MASTER EN SCIENCES INFORMATIQUES

Modélisation des processus de conception de bases de données

Aubert, Stéphane

Award date:
1994

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Rue Grangagnage, 21
B-5000 NAMUR

Modélisation des processus de conception de bases de données

Stéphane Aubert

Promoteur : J-L Hainaut

Mémoire présenté en vue
de l'obtention du diplôme de
licencié et maître en informatique

Année académique 1993-1994

Résumé

La modélisation des processus de développement a pris une place importante dans le domaine du génie logiciel depuis quelques années. Les modèles sont considérés comme des moyens efficaces pour comprendre et raisonner sur les activités de création et de maintenance des systèmes.

Ce travail cible un domaine particulier du génie logiciel : l'ingénierie des bases de données. Il propose un modèle de processus basé à la fois sur la représentation des activités et produits mis en oeuvre au cours de celui-ci, et sur les raisonnements qui ont conduit à sa réalisation. Le modèle présente ces concepts sur deux niveaux différents, le premier consacré à la représentation d'une méthode, le second à celle de son application dans une exécution réelle.

Au-delà de cette modélisation, les concepts introduits peuvent être utilisés pour la définition d'un moteur de méthode au sein d'un outil CASE. La spécification de ces objets permet de définir une architecture pour l'intégration des concepts méthodologiques dans l'outil de conception de base de données DB-MAIN.

Abstract

Recently, the Software Engineering community has begun to pay more attention to software process modeling. Process models are seen as important vehicles for understanding and reasoning about software creation and evolution activities.

This thesis focus on Database Engineering, a paticular domain of Software Engineering. A process model, based on the representation of activities, products and rationale of a development, is presented. These concepts are introduced on two levels corresponding to a method definition, and to its application in a process performance.

The objects presented allow methodological aspects to be defined into a Database CASE tool. A architecture that will be integrated to the repository of the DB-MAIN CASE tool is specified.

Remerciements

Je tiens tout d'abord à exprimer ma profonde gratitude à mon promoteur, monsieur Jean-Luc Hainaut pour les conseils et les encouragements qu'il m'a prodigués tout au long de la réalisation de ce travail.

J'exprime également toute ma reconnaissance à Madame Jeanine Souquières, ainsi qu'à tous les membres de l'équipe PROGRAIS du Centre de Recherche en Informatique de Nancy pour m'avoir permis d'effectuer, dans les meilleures conditions, un stage dans leur centre.

J'adresse aussi mes remerciements aux chercheurs du projet DBMAIN, et plus particulièrement à Didier Roland dont le travail est en étroite corrélation avec le sujet de ce mémoire.

Enfin, je remercie toutes les personnes qui ont participé de près ou de loin à l'achèvement de ce travail.

Merci.

Stéphane Aubert

Août 1994

Table des matières

Introduction

1. Motivations

Les coûts de développement et de maintenance des systèmes logiciels n'ayant cessé de croître depuis quelques années, la communauté des chercheurs en Génie logiciel a commencé à porter une attention considérable à l'amélioration des processus de création et de maintenance de ces systèmes, dont la complexité ne fait qu'ajouter au problème.

Cette attention s'est traduite par une recherche accrue dans le domaine de la modélisation des processus. Ces modèles sont considérés comme importants pour supporter la compréhension et le raisonnement sur les activités de développement. Ils peuvent en outre servir de base pour la structuration des environnements logiciels automatisés qui permettent d'aider les concepteurs, et qui font également l'objet de nombreuses recherches.

Malgré les efforts fournis dans ces domaines, certains problèmes restent toujours posés :

- aucun modèle vraiment satisfaisant n'a encore été proposé. Ils ne sont généralement concentrés que sur une partie des concepts nécessaires à la représentation complète du processus.

- les méthodes et outils utilisés pour simplifier, organiser et améliorer les activités de développement finissent généralement par augmenter la complexité du processus.

- l'assistance méthodologique est souvent laissée de côté dans les modèles et les outils d'aide au développement.

2. Notre objectif

Partant de ces considérations, l'objectif de notre travail est de développer un modèle qui puisse servir de base pour une compréhension et une amélioration des activités de création et de maintenance des systèmes.

Nous nous basons pour cela sur le fait que la représentation d'un développement doit prendre en charge principalement deux aspects :

- la description des différents produits, et des activités qui permettent de les élaborer.

- l'expression des raisonnements suivis pour construire ces produits.

Au-delà de cette modélisation, nous pensons qu'il est également important de fournir, grâce au modèle et à un langage de spécification, les moyens permettant de décrire des méthodes de développement, ainsi que l'historique du processus suivi avec ses différents composants.

Beaucoup de raisonnements peuvent être guidés par des méthodes. Bien que la spécification de celles-ci ne soit pas suffisante pour exprimer les raisonnements utilisés, leur utilité est évidente pour la conduite efficace d'un processus de développement, et elle peut donc constituer une aide à l'amélioration de celui-ci.

L'historique d'un processus est aussi utile car il peut, entre autre, servir de feed-back aux raisonnements afin d'améliorer le processus.

Enfin, nous sommes encore intéressés par l'assistance à apporter au concepteur, notamment pour l'aider à observer une méthodologie particulière. Cette aide peut être fournie principalement par l'utilisation d'un environnement supportant le processus et la méthode correspondante. Nous essayerons donc aussi de définir les éléments nécessaires à l'intégration de ces concepts dans un tel outil.

3. Approche suivie

Les pratiques systématiques de développement de systèmes logiciels sont normalement applicables à n'importe quelle classe de système qui a une durée de vie supérieure à son temps de développement, et qui demande plus qu'une personne pour conduire sa conception.

Notre propos est de nous focaliser sur le domaine particulier des bases de données (BD), que l'on peut considérer comme un sous-ensemble des nombreux systèmes possibles.

La communauté des chercheurs en bases de données doit faire face à la même crise que le reste de la communauté des développeurs de systèmes logiciels. Le fossé entre la collecte des informations et des requirements, et la représentation physique permettant leur implémentation s'est fortement élargi, et a impliqué la nécessité de définir des cadres et formalismes de travail.

Nous allons donc nous concentrer sur cette discipline en espérant que les efforts fournis pourront facilement être adaptés à d'autres cas dans le vaste domaine du génie logiciel.

Présentation de l'ouvrage

Le document est organisé de la manière suivante :

Dans le premier chapitre, nous proposons un état de l'art qui reprend les différents travaux concernant la modélisation des processus de développement de systèmes. L'analyse de ces réalisations permettra de mettre en exergue les éléments importants qu'ils utilisent, ainsi que les problèmes qu'ils présentent. Cette critique nous permettra alors de définir la problématique de notre travail.

Le chapitre 2 est consacré à la présentation du domaine de l'ingénierie des bases de données. Nous décrivons les principaux modèles utilisés, et donnons une vue générale des méthodes permettant de guider le processus de conception des bases de données. Nous définissons alors plus précisément nos objectifs en fonction de ce domaine particulier.

L'objet du chapitre 3 est de présenter un modèle de représentation de processus. Après en avoir donné les caractéristiques générales, nous donnons une description de ses différents composants. Nous proposons aussi la syntaxe d'un langage de description de méthode qui pourra être utilisé par l'ingénieur pour spécifier celle-ci.

Le chapitre 4 constitue une étude de cas qui consiste à utiliser le langage défini au chapitre 3 pour spécifier une méthode particulière de conception de bases de données. Nous proposons aussi un court exemple d'une partie d'historique, et de ses composants jugés nécessaires pour une représentation suffisante d'une trace d'exécution.

Le chapitre 5 concerne la définition et l'intégration des concepts de méthode au sein d'un atelier de génie logiciel. Après avoir donné une description générale des outils CASE, et plus particulièrement ceux de conception de bases de données, nous proposons un ensemble d'éléments nécessaires à la définition d'un moteur de méthode et qui serons intégrés au sein de l'outil DBMAIN. Les objets sont tirés du modèle de représentation décrit au chapitre 3, et viennent s'ajouter à ceux du méta-schéma de l'atelier.

La conclusion reprend les différentes réalisations du travail, et propose des possibilités d'extensions.

Les annexes se décomposent comme suit :

- L'annexe A présente des diagrammes de représentation utilisés par les modèles de données présentés dans le chapitre 2.
- L'annexe B propose la spécification des attributs et méthodes des objets représentant les concepts méthodologiques et constituant l'architecture proposée dans le chapitre 5.

Chapitre 1

Etat de l'art

1.1 Les modèles de processus

Les efforts en la matière étant fournis par une multitude de chercheurs, un certain nombre de modèles ont été proposés, basés sur des approches et des paradigmes distincts. Les processus sont souvent décomposés en différentes étapes qui forment ce qu'on appelle le cycle de vie du système. Les modèles de cycle de vie fournissent des cadres de travail pour appliquer différentes activités de développement théoriques ou intuitives, des techniques, des méthodes et des outils. Ils fournissent aussi la base nécessaire pour la planification et la conduite du processus de développement, ainsi que les moyens de raisonnement au sujet des processus organisationnels utilisés. Décrire et modéliser les processus de développement est crucial pour leur compréhension et pour aider à leur réalisation.

Depuis un certain nombre d'années, beaucoup de travaux ont été consacrés à la modélisation des processus (voir par exemple [Fou88], [Lehm87], [Ost87], [Rom89]), et plusieurs approches ont été proposées.

Certaines, plutôt **orientées vers les produits**, représentent le processus à travers une évolution d'un produit. Bien que ces modèles présentent l'avantage de lier l'activité de production à son résultat (le produit), ils ont beaucoup été critiqués parce qu'ils étaient trop concentrés sur les produits fournis par le processus, et pas sur le processus lui-même. Des exemples de tels modèles "orientés-produits" sont le "View Point Model" [Fin90], et le modèle de développement proposé dans le projet ESF (European Software Factory) [Peu91].

Un deuxième type d'approche, qui est suivie par la majorité des efforts dans le domaine consiste à se concentrer sur l'aspect fonctionnel des processus. Un des premiers modèles qui a suivi cette "**orientation-tâche**" est le "Waterfall Model" [Royce70]. Il est assez linéaire, et fonctionne avec des phases distinctes, généralement conduites en séquence. Chaque phase génère des documents qui peuvent être utilisés par la phase suivante, ainsi que pour les tests et la maintenance du système.

Le "Spiral Model" [Boehm86] est aussi de type orienté-tâche. Il décrit un processus comme une itération sur quatre phases d'activités, chaque itération impliquant une progression dans une même séquence d'étapes qui transforment les parties d'un produit à travers une série de niveaux d'élaboration. Il ajoute les concepts d'analyse de risque, prototypage et itération au cadre du Waterfall model, ce qui lui donne certains avantages.

Le Spiral model, ainsi que le "Software Certification Lifecycle" [Cur86] tentent de représenter une approche plus basée sur les risques, se concentrant sur les problèmes clés difficiles plutôt que sur les faciles, sur les activités essentielles plutôt que sur les "accidentelles". Ils ajoutent aussi une certaine flexibilité qui fait défaut dans le Waterfall Model.

Le modèle "contractuel" est né d'un besoin d'une approche plus générale capable de décrire les développements dans une grande variété d'environnements organisationnels, méthodes, et domaines d'application. Utilisé dans ISTAR ([Lehm85], [Dows87]), il permet une structure dynamique du processus, et considère que les activités peuvent être structurées en un ensemble de contractions spécifiant les activités individuelles et leurs relations, résultant en une structure hiérarchique de contractions et sous-contractions. Il satisfait à beaucoup de contraintes pour une représentation générale d'un processus, et fournit une base utile à la structuration d'un environnement logiciel. Il possède cependant un désavantage par les restrictions sur des interactions uniquement entre des paires d'activités, et la nature restrictive des contractions ou organisation des projets hiérarchiques.

Il existe encore d'autres modèles qui représentent les processus suivant cette approche "orientée-tâche", citons encore pour exemple le "Iterative Enhancement" [Basi75], mais la plupart d'entre eux et la manière de les penser sont basés sur le cadre du Waterfall model.

Certains ont essayé de remédier au problème des modèles orientés-tâche en proposant une représentation "**orientée-comportement**", qui consiste à se concentrer sur les effets produits par les activités plutôt que sur les procédures spécifiques utilisées. C'est par exemple le cas dans [Phil89], ou du SPM [Will88] qui décrit un développement de système comme une collection d'activités qui peuvent être réalisées de manière concurrente et asynchrone, et par différentes personnes. La communication et la synchronisation entre les activités est réalisée par l'envoi de messages dont la nature est déterminée par les activités et les besoins des développeurs. Il utilise des descriptions relativement proches de celles rencontrées dans le système DREAM [Ridd78] de modélisation.

Le modèle "TAP" [Yone93] suit aussi cette approche. Le concept central est celui de tâche liée aux différents produits et agents, mais il introduit une représentation du comportement sous forme de réseau de Pétri conditionnel.

Enfin, une dernière approche, plus récente, consiste à utiliser des modèles de représentation de processus "**orientés-décision**". Chaque étape de transformation d'un produit est vue comme la conséquence d'une décision. Les modèles tels que NATURE [Rol93], celui du projet DAIDA [Jar92], ou encore celui présenté dans [Pot89] appartiennent à cette catégorie. Ils représentent comment un développement est conduit, mais aussi pourquoi, ce qui les rend beaucoup plus puissants. En effet, ils fournissent plus d'informations sur le processus en ce qui concerne les raisonnements utilisés, ce qui s'avère fort utile pour la réutilisation de certains produits, ou la remise en cause de ceux-ci pour raison de modification des requirements.

1.2 Critique de cet état de l'art

La proposition de différentes orientations pour la modélisation des processus utilisés dans le développement de systèmes a permis d'ajouter un complément à l'habitude plus traditionnelle qui consistait à se concentrer sur les différents produits fournis par le processus. Celle-ci s'avérait effectivement insuffisante pour servir de base à un raisonnement sur les processus de développement de plus en plus complexes.

Cependant, parmi les différentes approches qui ont été proposées, et que nous venons de décrire, on peut aisément se rendre compte que la majorité d'entre elles ne suffisent pas non plus pour représenter tous les éléments intervenant dans un processus.

Comme nous l'avons dit, un grand nombre de modèles proposés ont été basés sur le cadre du "Waterfall Model". Si celui-ci a eu son utilité pour décrire certains types de processus, il a aussi présenté un certain nombre de défauts, par exemple :

- il implique une vision plutôt irréaliste des activités de développement comme des séquences uniformes et ordonnées,

- il ne s'accorde pas non plus facilement avec des développements récents comme le prototypage rapide et les langages avancés,

- il ne fait pas référence de manière adéquate à la propagation de modifications dans un développement,

- Enfin, il ne fournit pas assez de détails pour supporter l'optimisation du processus.

Cette confiance portée au Waterfall Model a eu quelques conséquences malheureuses, comme le fait de considérer que chaque étape du processus doit être terminée avant que la suivante ne commence. On sait très bien que certaines étapes doivent être réalisées de concert (par exemple la conception et la documentation avec l'implémentation).

De toutes manières, les approches centrées sur les activités, qui représentent des processus de développement comme des programmes, impliquent toujours certaines limitations parce qu'elles n'intègrent pas les aspects interactifs du développement [Rol93].

Les approches "orientées-comportement" ont permis de franchir un pas en ce qui concerne les éléments impliqués dans un processus de développement. Mais, si elles ont apporté des descriptions concernant la manière de conduire un processus, elles n'ont apparemment pas été suffisamment approfondies, et elles ont fini par être très proches d'une orientation-tâche par le fait qu'elles se concentraient sur les moments où les tâches étaient réalisées.

Le problème fondamental des modèles proposés est qu'ils ne représentent pas de manière assez précise les aspects comportementaux de ce qui est réellement réalisé. Ils sont beaucoup trop sensibles à la notion de séquence de tâches, et certains ajustements entraînent parfois une restructuration complète du modèle. Le fait de se concentrer sur la modélisation des tâches pour guider le concepteur limite la flexibilité et tend à imposer une certaine rigidité.

L'approche "orientée-décision" est celle qui nous semble aborder le mieux cet aspect comportemental des processus. Même si elle ne le traite pas encore de manière adéquate puisqu'elle ne fait que des allusions limitées sur quand et comment décider de quelque chose, elle introduit des éléments nouveaux qui sont aussi importants pour pouvoir opérer des raisonnements convenables sur la conduite d'un processus.

1.3 Notre problématique

L'objectif de notre travail tel que nous l'avons présenté dans le deuxième paragraphe de ce chapitre peut être atteint par les trois réalisations suivantes auxquelles nous allons nous attacher :

1. Définir une approche qui permette de prendre en compte les différents aspects d'un processus de développement.

2. Essayer de donner une représentation de ces éléments à différents niveaux pour que le modèle puisse être utilisé par plusieurs catégories de personnes intervenant dans le processus de développement (par exemple les ingénieurs méthodes, les développeurs d'applications, ...).

3. Apporter une aide au développeur en lui proposant des moyens de guidance et d'assistance méthodologique.

Ces objectifs découlent directement de la définition d'un modèle de représentation de processus. Le premier point a été abordé de différentes façons par les approches décrites dans l'état de l'art. Le second, s'il n'a pas toujours été mentionné de manière explicite, a aussi été traité dans certains modèles (NATURE, TAP). Le troisième a commencé à apparaître comme sujet de recherche avec notamment ce qui concerne le développement des outils CASE au sein desquels il peut être réalisé.

Les recherches effectuées sur les modèles de représentation sont à l'origine des idées proposées dans ce travail. Les différentes approches décrites seront utilisées en partie, et certains efforts seront aussi fournis pour établir des cadres de travail et des formalismes de représentation. Nous avons par exemple repris deux idées importantes abordées dans ces travaux :

- La représentation d'un processus sur plusieurs niveaux d'abstraction
- L'association des concepts d'activité et produit avec ceux de décision et d'alternative.

Ceci permet de représenter non seulement les activités réalisées durant le processus, mais aussi pourquoi elles le sont et quand.

Le modèle que nous allons développer a pour objectif de servir de moyen de compréhension et d'amélioration des activités de conception et de maintenance de systèmes. Pour cela, nous allons utiliser une approche qui constitue un "mélange" de celles que nous avons présentées. En effet, nous pensons que les différents éléments mis en valeur par chacune d'elles ont leur importance, et que le fait de tous les intégrer en un seul modèle nous permettra de donner une représentation beaucoup plus efficace.

De manière à prendre en compte les différents aspects d'un processus, nous essayerons d'associer les concepts principaux de tâches, produits et raisonnements, qui constituent à notre avis les éléments principaux permettant de le décrire complètement. Cette approche permet de ne pas se concentrer uniquement sur les activités et les produits fournis, mais introduit en plus la notion de raisonnement qui permet d'aboutir à un processus particulier. Elle permet donc de représenter les différentes formes d'informations intervenant dans celui-ci à savoir ce qui doit être fait, comment, quand et pourquoi.

En ce qui concerne les moyens d'aide à apporter au développeur, nous essayerons principalement de nous concentrer sur les aspects méthodologiques des processus de conception.

Les outils CASE sont efficaces pour l'enregistrement et la manipulation des modèles de systèmes, mais ne sont actuellement pas capables de fournir une véritable assistance aux développeurs pour les activités de création et de transformation.

Nous proposerons donc une représentation des éléments nécessaires à la définition d'un moteur de méthode, et essayerons de voir comment réaliser cette assistance de manière concrète en intégrant ces concepts au sein d'un atelier de génie logiciel (AGL).

Chapitre 2

Cas particulier : L'ingénierie des bases de données

Introduction

Il existe un grand nombre de définitions et de modèles mis en oeuvre dans le domaine de la conception des bases de données. Ce chapitre a pour but de donner un aperçu de ce qui y est le plus couramment utilisé.

La première section concerne une présentation du processus de conception, en quoi il consiste, et quels sont ses objectifs et caractéristiques. Elle propose aussi une étude générale des principaux modèles utilisés. Nous y présenterons ceux qui sont le plus fréquemment rencontrés dans la littérature, et terminerons avec une comparaison. Enfin, nous achèverons cette partie en faisant le point sur les méthodes de conception, voir pourquoi elles deviennent de plus en plus nécessaires, quel est leur rôle, et comment elles se présentent d'une manière générale.

La deuxième section de ce chapitre consiste à présenter nos objectifs pour ce travail. Ceux-ci constituent une précision de ceux que nous avons proposés dans l'introduction, en fonction du domaine particulier des bases de données auquel nous nous attachons.

2.1 Description du domaine

2.1.1 La conception des bases de données

Dans une entreprise, une administration, ou toute collectivité organisée, il n'y a pas de système d'informations et donc pas de communication possible sans clarification, description, organisation et manipulation des données. Il est donc important de définir le mieux possible cette collection de données concernant la partie de la réalité à laquelle s'intéresse l'organisation, en se rendant compte que ce qui paraît évident n'est pas toujours bien compris.

Le problème de la conception de bases de données peut être énoncé comme suit : "*Construire les structures logiques et physiques d'une ou plusieurs bases de données dans le but d'accommoder les besoins d'information de l'utilisateur dans une organisation en fonction d'un ensemble défini d'applications diverses*".[Elm89]

La décision d'implémenter une BD est motivée par ce besoin de partager des informations, ainsi que d'en intégrer d'autres qui permettent de supporter des applications plus compliquées.

Les buts de cette conception sont multiples : satisfaire les requirements de l'utilisateur ou des applications sur les informations, fournir une structuration naturelle et facile à comprendre de l'information, et observer les requirements techniques et de performances comme le temps de réponse, le temps d'accès, et l'espace de stockage.

Plusieurs éléments constituent un problème pour le concepteur d'application :

- Déterminer la "portion de réalité" à représenter, c'est-à-dire les requirements.
- Consolider les requirements de plusieurs applications en un seul ensemble intégré pour une "collection partagée".
- Anticiper les modifications de ce qui attire "présentement" l'intérêt.

Les objectifs sont d'autant plus difficiles à atteindre que le processus commence souvent avec des requirements peu formels et très mal définis. Ceux-ci, une fois déterminés doivent alors être traduits dans un modèle logique des données, et dans les structures de stockage physiques supportées par un SGBD (système de gestion de bases de données). Les traductions varient considérablement d'un modèle à un autre, ou d'un SGBD à l'autre, mais les critères généraux restent toujours les mêmes : minimiser la redondance, éviter les anomalies lors de mises à jour, obtenir des performances acceptables et une facilité de modification.

Ce sont ces problèmes qui doivent être l'objectif de toute aide à la conception pour n'importe quel modèle de données. Le concepteur a pour mission de développer un objet qui peut être implémenté de manière réaliste dans des environnements existants, tâche rendue encore plus difficile par le besoin de fournir des accès fiables et efficaces aux données. Les objectifs de la conception sont réalisés en partie en fournissant des systèmes de gestion permettant de relier les données en structures complexes grâce à des mécanismes comme les chaînes de pointeurs, les index et le positionnement séquentiel. Ils sont aussi atteints grâce au développement de méthodologies et de règles comme par exemple la volonté de stocker les données de manière non redondante, etc. ...

Le processus consiste typiquement en deux activités parallèles. La première comprend la conception de la structure des données et du contenu de la BD. La seconde concerne celle des applications qui la manipulent. Ces deux activités sont fortement liées et s'influencent l'une l'autre. Cependant, elles ne sont habituellement pas soulignées de la même manière dans les différentes méthodologies, la conception ayant tendance à mettre l'accent sur l'une ou l'autre ("data driven" ou "process driven"). Mais il est maintenant reconnu que les deux activités doivent être menées conjointement.

La conception de bases de données doit englober tous les aspects des données à stocker, depuis les détails concernant la manière de les présenter aux différents utilisateurs, jusqu'à la façon dont elles doivent être représentées sur le hardware d'une installation particulière. Pour réaliser ceci correctement, le processus est structuré en différentes phases (cfr. le paragraphe sur les méthodes de conception) qui permettent de passer de l'identification et la conception des interfaces des utilisateurs finaux jusqu'à la création des moyens physiques représentant la BD.

2.1.2 Les modèles de données

Une base de données est utilisée pour organiser de l'information au sujet de certains aspects du monde réel. Les concepteurs approchent et collectent les données qui décrivent le monde réel. Lorsqu'ils ont fait suffisamment d'observations, ils classent ces données suivant certaines abstractions dont ils établissent la valeur selon la manière dont elles les aident à manipuler l'univers avec un minimum d'exceptions.

L'outil qu'ils utilisent pour documenter les informations essentielles de la BD est le langage de spécification, appelé **modèle de données** dans le domaine des bases de données, dont le vocabulaire et la syntaxe représentent la manière dont les concepteurs perçoivent l'application qu'ils construisent. L'objet de tout modèle de données est de permettre à son utilisateur de décrire et manipuler les objets du monde réel qu'il veut stocker dans la BD. Il est important, pour avoir une bonne conception de choisir un bon modèle. Il est aussi fondamental de noter l'importance de fournir des directions sur la manière dont le concepteur doit utiliser les concepts du modèle pour représenter le monde réel. Un modèle trop simple produit une spécification difficile à comprendre et donc à utiliser et vérifier. Si certaines nuances d'une application sont faciles à exprimer et d'autres plus difficiles dans le modèle, l'utilisateur retrouvera aisément les premières dans ses applications, mais manquera les autres.

Nous décrivons les principaux modèles utilisés pour la représentation d'une BD, sans aller trop loin dans les détails. Pour plus d'informations concernant ceux-ci, nous renvoyons à la littérature correspondant à chacun d'eux. Une description des représentations graphiques qu'ils utilisent est également proposée dans l'annexe A.

2.1.2.1 Le modèle relationnel

Le modèle relationnel des données a été introduit par Codd [Codd70]. Parmi ceux dont nous allons parler, il a la structure la plus simple et la plus uniforme. Il représente l'ensemble des données d'une BD comme une collection de relations qui sont vues comme des tables de valeurs, chaque ligne contenant les valeurs des différents composants de la donnée qu'elle représente. Ces valeurs peuvent être interprétées comme un fait décrivant l'instance d'un objet ou d'une relation. Le nom de la table ou de la colonne est utilisé pour aider à interpréter la signification des valeurs de chacune des lignes.

Dans la terminologie des BD relationnelles, une ligne est appelée un **tuple**, le nom d'une colonne un **attribut**, et la table une **relation**. Le type de données qui décrit les types des valeurs de chaque colonne est appelé le **domaine**. On peut encore ajouter un **format** pour chaque donnée (par exemple, une donnée du domaine "Age-employe" peut être déclarée comme un nombre compris entre 16 et 70), ainsi que des informations pour interpréter les valeurs (par exemple le domaine Age-employe peut avoir "années" comme unité de mesure).

La valeur sémantique du modèle relationnel de données est en grande partie responsable de sa puissance pour la lecture et la manipulation des données. Mais elle joue en sa défaveur en ce qui concerne les opérations de mise à jour. Ce problème est reconnu depuis pas mal de temps [Codd71], et il a donné lieu à la théorie des formes normales [Bern76], [Fagi77].

Le modèle relationnel rassemble en tuples les attributs qui sont fonctionnellement dépendants. Il décrit les ensembles de tuples en utilisant la théorie mathématique des relations. Cette base mathématique du modèle relationnel, la représentation uniforme de toutes les structures comme des relations, et le peu de complexité fournissent d'importants avantages pour ce modèle et l'analyse des requêtes.

2.1.2.2 Le modèle Entités/Associations

Chen a développé un modèle basé sur le relationnel qui fait clairement la distinction entre les entités et les relations [Chen76], [Chen77]. Le modèle Entités/Associations (ERA) est un modèle conceptuel de représentation de données de haut niveau. Les concepts utilisés sont proches de la perception qu'a l'utilisateur des données, et n'ont pas comme objectif de décrire la manière dont elles seront stockées dans le système.

Le premier concept utilisé par le modèle ERA est l'**entité**, qui représente un objet du monde réel avec une existence indépendante. Chaque entité est décrite par un certain nombre d'**attributs** pour lesquels elle possède une valeur. Ces attributs peuvent eux-mêmes être composés de plusieurs attributs, il sont alors appelés décomposables, un attribut non divisible est appelé atomique. Une autre caractéristique d'un attribut est qu'il peut être multivalué, c'est-à-dire qu'il peut avoir un ensemble de valeurs pour la même entité.

Une BD possède en général un certain nombre d'entités qui sont similaires. Par exemple une compagnie rassemble plusieurs employés. Chaque entité "employé" possède les mêmes attributs, mais a ses propres valeurs pour chacun d'eux. Ces entités peuvent être groupées en un type d'entité (TE) qui est l'ensemble des entités qui ont les mêmes attributs. Un TE spécifie une structure commune partagée par les entités de ce type, et le schéma précise le nom de ce type, le nom et la signification des attributs, et certaines contraintes qui portent sur les entités.

Le deuxième concept de base du modèle est celui d'**association**, qui permet de représenter les relations entre les différentes entités. Un type d'association (TA) R entre les types d'entités E_1, E_2, ... , E_n est l'ensemble des instances d'associations entre les entités de ces types. Chaque relation représente le fait que les entités participantes sont mises en relation d'une manière correspondant à une situation du monde réel. Le **degré** d'un TA est le nombre de TE qui y participent, et qui jouent un rôle particulier dans cette relation.

Les TA ont habituellement un certain nombre de contraintes qui limitent les combinaisons possibles d'entités y participant (cardinalité, inclusion, exclusion, ...). Celles-ci sont déterminées à partir de la situation du monde réel qu'elles représentent. De la même manière que nous avons défini un attribut d'un TE, un TA peut aussi posséder des attributs.

Parmi les différents types de contraintes qui peuvent porter sur un schéma, celle d'identification est une des plus importantes. Un attribut identifiant est un attribut dont la valeur est distincte pour chaque entité d'un type donné, celle-ci peut être utilisée pour identifier l'entité de manière unique.

Les concepts du modèle ERA que nous avons présentés peuvent modéliser une grande variété d'applications de BDs. Cependant, il faut savoir que des nouvelles applications comme les BDs pour l' "engineering design" ou l'intelligence artificielle demandent des concepts supplémentaires pour pouvoir être modélisés avec précision.

D'autres modèles ont donc été proposés. Parmi ceux-ci, le modèle ERA étendu (ou EER pour 'Extended Entity Relationship') est le modèle ERA augmenté de concepts importants tels que ceux de **spécialisation** et **généralisation**, ainsi que des mécanismes d'**héritage** des attributs. Nous ne nous y attarderons pas car ce n'est pas notre propos, mais nous renvoyons à la littérature pour une description de ces extensions ([Ham81], [Sch79], [Smith77], [Teo86], ...).

2.1.2.3 Le modèle hiérarchique

Le modèle hiérarchique des données a été développé pour représenter les différents types d'organisations hiérarchiques qui existent dans le monde réel. Peu de documents originaux le décrivent [McGee77], [Bjo82], contrairement aux modèles relationnels ou en réseau. Pourtant, plusieurs systèmes de gestion d'information ont été développés en utilisant des structures de stockage hiérarchiques.

Le modèle hiérarchique utilise deux concepts principaux de structure de données : les **records** et les **relations "parent-enfants"** (RPE). Un record est une collection de valeurs de champs qui fournissent de l'information sur une instance d'une entité ou d'une relation. Les records d'un même type peuvent être regroupés en **types de record**, qui possèdent un nom et dont la structure est définie par une collection de champs nommés (ou items).

Un type de relation "parent-enfants" est une relation 1:N entre deux types de records. Le type de record du côté 1 est appelé le parent, celui du côté N l'enfant. Une occurrence d'un type de RPE consiste en un record du type de record parent, et un nombre de records (0 ou plusieurs) du type de record enfant.

La définition d'un schéma hiérarchique définit une structure de données en arbre dans laquelle un type de record correspond à un noeud et un type de RPE à un arc (voir en annexe).

Il existe aussi un certain nombre de contraintes inhérentes aux RPE qui existent quel que soit le schéma. Celles-ci concernent par exemple les occurrences de records et leurs relations avec un parent, la multiplicité des parents d'un record d'un même type ou d'un type différent, ... En plus, chaque schéma peut avoir ses propres règles d'intégration supplémentaires.

D'autres contraintes qui ne sont pas implicites dans le schéma hiérarchique devront être explicitement définies par les programmeurs dans les programmes de mise à jour de la BD.

2.1.2.4 Le modèle en réseau

Les premiers travaux sur le modèle en réseau ont été réalisés par C. Bachman [Bach64]. D'autres ont suivi avec par exemple Dodd [Dodd69], et en 1971, les standards sont définis dans un rapport du comité CODASYL (Conference on data systems language) [DBTG71].

Ce modèle utilise deux structures de base : les **records** et les **sets**. Les données sont stockées dans les records, qui consistent chacun en un groupe de valeurs de données reliées. Les records sont classés en types de records qui décrivent la structure d'un groupe contenant les mêmes types d'informations. Ils possèdent un **nom**, et chaque **attribut** (ou item) a un nom et un format (type de donnée).

Contrairement aux modèles relationnels et hiérarchiques, qui n'autorisent que des attributs simples, le modèle en réseau permet de définir des items de données complexes. Un **vecteur** est un item qui peut avoir plusieurs valeurs dans un seul record[1]. Un "**repeating group**" permet l'inclusion d'un ensemble de valeurs composées pour un item d'un même record[2] (le concept de repeating group n'est cependant pas essentiel étant donné qu'il peut être remplacé par deux types de records et un type de set).

Les records de différents types sont mis en relation. Pour représenter les relations possibles entre ceux-ci, le modèle utilise le concept de **type de set**. Chaque instance met en relation un type de **record propriétaire** avec un ensemble de **records membres**. Chaque occurrence de set est donc composée d'un record propriétaire, et d'un nombre (0 ou plusieurs) de records membres. Un record du type propriétaire ne peut pas exister dans plus d'une seule occurrence d'un type de set, ce qui introduit la contrainte qu'un type de set représente une relation 1:N entre deux types de records, il n'est donc normalement pas possible de représenter directement une relation M:N avec un seul type de set. Une méthode correcte pour y arriver est d'utiliser deux types de sets et un type de record supplémentaire appelé type de record de liaison.

Il existe aussi plusieurs contraintes portant sur la participation à un set. Elles sont habituellement divisées en deux catégories appelées **options d'insertion** et **options de conservation** dans la terminologie CODASYL. Ces contraintes sont déterminées durant la conception de la BD en sachant comment un set doit se comporter quand des records membres sont insérés, et quand un record est supprimé. Elles sont spécifiées au SGBD lors de la déclaration de la structure de la BD avec le langage de définition de données (LDD). Il est à noter que toutes les combinaisons de contraintes ne sont pas possibles, mais nous n'irons pas plus loin dans cette description.

2.1.2.5 Le modèle structurel

Le modèle structurel [Wie79] est un modèle relationnel étendu qui représente les relations jugées importantes pour la conception des structures de la BD tout en gardant les généralités de celui-ci. La plupart des modèles précédents ont été proposés comme des outils pour la définition du modèle logique des données. Le modèle structurel sert deux optiques :

1. Une représentation de la structure de la BD incluant certains concepts sémantiques, avec un ensemble limité de constructions de base.

2. Une guidance pour l'implémentation de la BD.

Dans ce modèle, les **relations** sont utilisées pour représenter les classes d'entités. Les relations simples (1:1 ou 1:N) entre celles-ci sont représentées par des **connexions**. Il est possible de représenter les relations M:N par une relation et deux connexions.

Pour des propos d'implémentation, les relations du modèle sont classées en différents types selon leurs interactions.

---

[1]Correspond à la notion d'attribut simple multivalué dans la terminologie ERA.
[2]Correspond à la notion d'attribut décomposable multivalué dans la terminologie ERA.

1. Une relation primaire définit un ensemble de tuples qui correspond à une classe d'entités du monde réel. Son but est de faire une correspondance la plus étroite possible entre les relations d'entités et les entités du monde réel.
2. Une relation emboîtée permet de résoudre le problème des attributs répétitifs qui n'est pas traité par le modèle relationnel.
3. Une relation référencée permet de représenter le fait que certaines entités du monde réel représentées dans le modèle sont référencées par d'autres.
4. Une relation 'lexicon' définit une correspondance 1:1 entre deux attributs.
5. Une relation d'association correspond à une relation entre des classes d'entités dans une situation du monde réel.

Associées à chaque type de connexion, il existe finalement des contraintes d'intégrité qui définissent la dépendance des tuples dans les deux relations connectées. Ces contraintes définissent les conditions pour la maintenance de l'intégrité structurelle du modèle.

2.1.2.4 Comparaison des modèles

Suite à l'analyse des différents modèles, on peut constater que le **modèle relationnel** possède une fondation mathématique plus formelle à la fois dans la description de ses structures, et dans le langage utilisé. Il fournit aussi généralement une plus grande flexibilité. Il est cependant beaucoup critiqué parce qu'il ne représente pas les relations entre les entités logiques de manière explicite.

Les **modèles en réseau** et **hiérarchique** ont été développés avant le modèle relationnel et sur base de systèmes commerciaux spécifiques. On peut donc comprendre que les langages qu'ils utilisent soient assez proches des commandes des systèmes de fichiers traditionnels.

Le **modèle en réseau** donne une représentation explicite des entités et de leurs relations. L'inconvénient est qu'il est fortement contraint par une attention initiale aux limitations d'implémentation, et que seules les relations représentées peuvent être implémentées et exploitées. Certains systèmes basés sur ce modèle ont des possibilités d'interrogation correspondant uniquement à des chemins d'accès implémentés, il est donc nécessaire d'implémenter des connexions pour servir aussi bien des interrogations rares de la BD que des importantes. En plus, étant données certaines contraintes d'implémentation, des relations sont parfois difficiles à exprimer, comme par exemple les ensembles récursifs qui sont des relations entre des instances d'entités appartenant à une même classe.

Souvent, les utilisateurs se sentent à l'aise avec le **modèle hiérarchique**. En général, il fonctionne bien pour les applications de BD qui sont naturellement hiérarchiques. Mais si ce n'est pas le cas, il est assez difficile d'adapter les relations sous cette forme, et les représentations qui en résultent sont souvent insatisfaisantes. L'inclusion de relations "parent-enfants" virtuelles le rapproche du modèle en réseau. Cependant, à cause de certaines restrictions sur ces relations RPE, le modèle réseau a de meilleures capacités de modélisation que le modèle hiérarchique.

Bien que le modèle relationnel et ses applications soient devenus très populaires actuellement, le modèle hiérarchique sera encore utilisé pendant un certain temps étant donnés les gros investissements qui ont été faits dans le monde commercial.

Le **modèle structurel** apparaît beaucoup moins puissant que le modèle relationnel original, car les mises à jour ne peuvent pas être faites librement. Dans le modèle relationnel, les connexions ne sont pas décrites, et sont donc laissées à découvrir à l'interrogation de la BD. Le manque de reconnaissance des relations peut simplifier les problèmes techniques, mais il n'élimine pas les inconsistances sémantiques. Le modèle structurel permet de se restreindre à la sémantique nécessaire à l'implémentation, et d'en laisser une partie à traiter à un niveau plus haut que celui auquel il correspond.

Les limitations des modèles relationnel et en réseau ont conduit à une recherche active dans les modèles de BD. Les nouveaux, plus complexes, utilisent en général des concepts sémantiques supplémentaires.

Le **modèle ERA** défini par Chen comme une représentation unique des données s'avère être un instrument très utile et performant pour la conception du schéma d'une BD. Utilisé surtout pour l'étape conceptuelle du processus, il est actuellement un des modèles qui sert principalement de référence, et la transformation d'un schéma ERA en schéma relationnel ou en réseau est assez systématique. De plus, les extensions et améliorations dont il a fait l'objet le rendent encore plus puissant dans ses capacités de représentation, et lui permettent de pouvoir faire face aux nouvelles applications plus complexes de la technologie des BDs. On s'attend à ce qu'un certain nombre de SGBD commerciaux basés directement sur lui, ou sur d'autres modèles similaires de haut niveau, soient disponibles dans un futur relativement proche.

2.1.3 Les méthodes de conception

La conception des bases de données est le processus qui utilise les informations pertinentes et les requirements des utilisateurs pour les transformer dans une représentation conforme à un SGBD.

Même sans tenir compte de requirements imposés ou d'innovations technologiques dans les différents composants, la complexité du processus de conception et production a commencé à inquiéter et parfois dépasser la portée intellectuelle de certains experts. La plupart des systèmes d'information sont extrêmement complexes, bien plus qu'on pourrait le croire à première vue. C'est pourquoi, il est devenu nécessaire de définir des méthodes (et des outils) pour essayer d'avoir des processus de conception efficaces.

Ces méthodes doivent s'appuyer sur certains concepts, comme entre autres :

1. La modularité, qui permet de subdiviser un problème compliqué en sous-problèmes de manière répétitive jusqu'à ce que les problèmes résultants deviennent moins complexes.
2. L'abstraction, qui permet de se concentrer sur un problème à différents niveaux de généralisation sans s'occuper de détails relevant d'un niveau inférieur. Elle permet aussi l'utilisation de concepts et termes familiers à l'environnement sans les transformer en une structure inconnue, ou non maîtrisée.
3. L'utilisation d'outils appropriés, la qualité du produit final est en grande partie déterminée par la qualité des outils utilisés pour la spécification, la conception, l'évaluation, et la description de celui-ci.

La qualité d'un produit est déterminée depuis le début de sa création. Un échec dans l'identification du problème ou des requirements résultera en une mauvaise spécification, qui entraînera un système inadéquat demandant beaucoup de modifications et d'améliorations menant à des coûts de maintenance élevés et à une insatisfaction de l'utilisateur. Cette raison, parmi beaucoup d'autres, fournit une des meilleures justifications pour l'utilisation d'une méthode systématique de conception d'une BD.

Construire une BD opérationnelle consiste à résoudre un certain nombre de problèmes, chacun de ceux-ci étant solutionné par un processus, ou une étape basée sur des concepts et des techniques spécifiques. Le rôle d'une méthode est de résoudre ces problèmes en définissant comment transformer un ensemble de spécifications en un schéma opérationnel de façon plus ou moins systématique.

Un certain nombre d'approches ont été formulées pour la conception des schémas de BD (voir [Curt78], [Kahn76], [Smith78], [Yao78], ...). Bien que celles-ci diffèrent dans les détails, les auteurs distinguent généralement quatre niveaux d'abstraction qui sont le niveau conceptuel, logique, physique et externe. Le processus peut être représenté en fonction de ces niveaux.

2.1.3.1 Le niveau conceptuel

Cette première phase consiste à examiner les requirements de l'utilisateur collectés et analysés pour produire un schéma conceptuel de la BD dans un modèle de haut niveau indépendant d'un environnement matériel ou logiciel. Le schéma permet une compréhension complète de la structure, la signification, les relations, ... des données, et doit être construit indépendamment d'un SGBD pour ne pas subir l'influence de ses particularités ou contraintes.

La bonne compréhension de ce schéma est cruciale pour les utilisateurs et les développeurs d'applications. Il est donc important d'utiliser un modèle de haut niveau, généralement plus expressif et général, et qui en plus doit être suffisamment simple, avec un petit nombre de concepts de base, et posséder une représentation graphique facilitant son interprétation.

Les deux modèles de référence souvent utilisés dans cette étape sont le modèle ERA et le modèle relationnel.

2.1.3.2 Le niveau logique

La conception logique a pour but de produire un schéma conforme au modèle de données d'un SGBD choisi. Elle est réalisée par la transformation du schéma conceptuel de haut niveau issu de la phase précédente en un modèle du SGBD. Le résultat est donc un schéma logique des données, utilisant les concepts habituels d'un SGBD et vérifiant ses propriétés. Celui-ci permet de représenter l'usage des données en décrivant les accès dont elles font l'objet.

2.1.3.3 Le niveau physique

Durant cette phase, les spécifications des données sont construites en termes de structures physiques de stockage, place des records, et chemins d'accès. Afin de réaliser les meilleures performances pour les différentes applications, le concepteur fait un choix quant à ces concepts et les options offertes par le SGBD. Les principaux critères utilisés pour guider le choix sont le temps de réponse, l'utilisation de l'espace, la capacité de transactions. Le résultat de cette conception physique est un produit issu d'une évaluation initiale des structures de stockage et accès aux fichiers. Il est

aussi souvent nécessaire de pouvoir accorder ce produit sur la base de ses capacités réelles observées après qu'il ait été implémenté. Certains systèmes offrent les possibilités de collecter des statistiques qui peuvent être analysées, et mener à certaines modifications ou réorganisations des fichiers (modification des index, méthodes d'accès, ...).

2.1.3.4 Le niveau externe

Cette étape permet de prendre en compte le fait que l'utilisateur du système d'information, ou un groupe d'utilisateurs, ne manipulent qu'un sous-ensemble de ces données. Elle définit les schémas externes qui concernent chacun une application particulière, ou une mise en oeuvre par un utilisateur et qui correspondent aux différentes contraintes de ceux-ci.

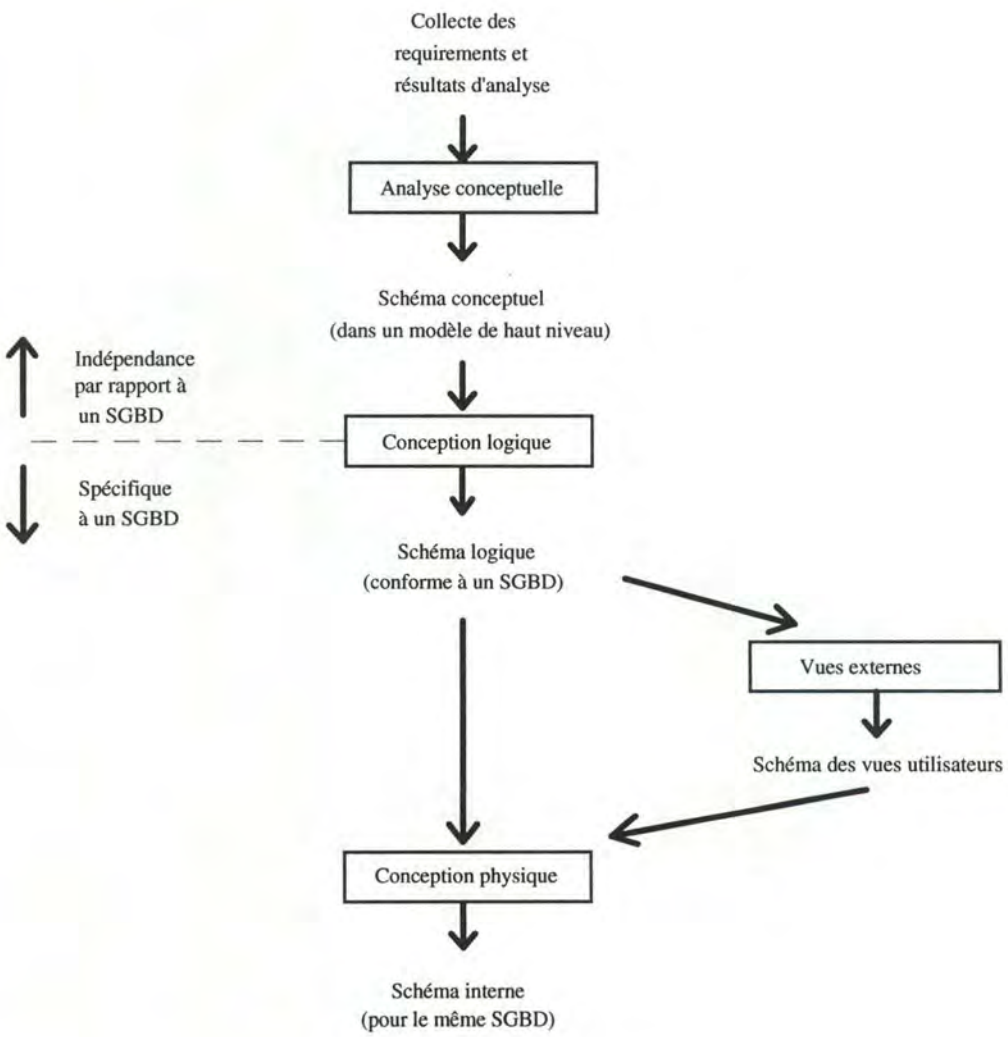


Fig. 2.1 Les étapes générales de la conception d'une BD

Ces différentes phases constituent les étapes générales de la plupart des méthodes standards actuellement utilisées pour la conception des bases de données, comme par exemple MERISE ([Tard83], [Tard85]), ou encore celles présentées dans [Tsao80] ou [Tard80]. Leur point de départ est l'analyse du réel "perçu" à partir duquel un schéma conceptuel est construit. Ensuite celui-ci est transformé en un schéma logique, qui sera à son tour traduit dans un schéma physique qui constitue le but principal de la conception.

Certaines méthodes peuvent aussi être spécialisées pour la conduite d'une étape particulière, avec parfois même une interprétation propre et plus spécifique pour celles-ci. Par exemple, certaines concerneront plus particulièrement l'élaboration du schéma conceptuel, comme par exemple celle présentée dans [Bat80], ou dans [Flo80]. D'autres seront plutôt destinées à l'étape logique comme celle décrite dans [Wong80] ou [NBS85], ou encore à l'étape physique comme dans [Schk78] et [Whang82].

Ces méthodes sont pour la plupart supportées par les modèles du type de ceux que nous avons présentés dans le paragraphe précédent, et principalement ceux de la famille du modèle ERA et du modèle relationnel.

2.2 Notre objectif

2.2.1 La modélisation du processus de conception

La plupart des processus de conception de bases de données sont encore conduits manuellement par des experts qui, bien qu'utilisant leur expérience et leurs connaissances dans le domaine, ne peuvent parfois faire autrement que de réaliser une série d'essais et erreurs. Afin d'éviter les désagréments dus à de telles pratiques comme les pertes de temps, les coûts occasionnés, etc... nous estimons qu'il est très important que le concepteur puisse suivre une stratégie bien définie plutôt que de permettre à son développement de s'étendre de manière "aléatoire".

Notre objectif à ce sujet est d'essayer de trouver un modèle de représentation qui permette de mettre en évidence les différents éléments décrivant la manière dont un processus particulier peut être exécuté. Nous considérons un développement comme une séquence d'opérations exécutées par le développeur pour obtenir une solution à son problème, et après avoir présenté les différents concepts intervenant dans cette exécution, nous essayerons d'en déduire une représentation des éléments nécessaires à la spécification d'une méthodologie. Ce modèle aura pour objet de définir comment un processus doit être réalisé par rapport à ses objectifs et aux contraintes d'un projet ou d'une organisation. Il permettra de décrire une façon de progresser dans les différentes étapes du processus en mettant l'accent sur deux types d'informations principales : les étapes-mêmes de la conception, et les liens entre celles-ci. La spécification de ces informations consiste à assurer que le bon travail est fait au bon moment, et impose une cohérence générale entre les requirements initiaux et le système obtenu comme résultat.

Nous essayerons d'obtenir des modèles suffisamment formels et rigoureux, qui offrent les moyens de raisonner plus facilement sur le processus, et fournissent une meilleure base pour que celui-ci puisse être développé ou supporté par des environnements de développement automatisés.

2.2.2 Langage de spécification de méthode

Il n'existe pas de notation unique idéale pour exprimer la conception d'un système. En fait, il y a probablement des centaines de notations différentes, chacune pouvant être intéressante à un certain niveau de détail de la conception ou dans un domaine d'application particulier.

L'existence de notations cohérentes et complètes est très importante pour la création d'objets aussi abstraits que les systèmes logiciels. Sans de telles notations, une conception ne pourra être évaluée, comparée, testée ni communiquée. Bien que le programme lui-même soit la spécification de conception absolue, son niveau de détail est tel qu'il ne convient pas à l'utilisation par un ingénieur logiciel. Ceci est particulièrement vrai pour les niveaux de conception plus élevés. Il est très difficile d'exprimer clairement la décomposition d'un grand système en unités fonctionnelles comme des sous-systèmes ou programmes en utilisant un langage de programmation. De plus, si cette approche présente l'avantage de produire directement un programme exécutable, elle possède néanmoins un certain nombre d'inconvénients :

1. Les langages de programmation ne sont pas facilement extensibles puisqu'ils doivent être compilables.

2. Les types de données, structures, et opérations qui peuvent être utilisées comme primitives dans les langages de programmation sont souvent d'assez bas niveau. La représentation de concepts intuitivement simples tels que les listes infinies peut très rapidement devenir complexe.

3. Etant donné que notre mode de pensée est contraint par le langage, plus le niveau du langage est bas, plus le travail du concepteur est susceptible d'être influencé par ce langage.

4. Dans le cas où la réalisation initiale d'une conception spécifiée dans un langage de programmation doit être reprogrammée, il devient difficile d'effectuer cette reprogrammation si le nouveau langage est de plus haut niveau que le langage de spécification de la conception.

Pour spécifier le processus de conception, nous essayerons de définir un langage de description de conception conçu pour documenter et communiquer la structure détaillée du système. (De tels langages ont d'ailleurs été proposés dans [Chu78], [Leer76], et [Linger79]).

Fondamentalement, ce type de langage de conception détaillée utilise les structures de contrôle habituelles des langages de programmation de haut niveau pour spécifier le flot de contrôle, mais il laisse une flexibilité considérable au concepteur en ce qui concerne la description des opérations. Un autre avantage à utiliser un tel langage est qu'il permet d'introduire et utiliser de nouveaux types abstraits de données, qu'il suffit de définir avant d'utiliser.

2.2.3 L'assistance via un atelier de génie logiciel (AGL)

Malgré la formulation de différentes approches pour la conception de bases de données, peu de méthodes formelles sont utilisées en pratique, et la plupart des développements sont opérés de manière "ad hoc". Une des raisons principales de cette situation est le manque d'outils de conception appropriés pour aider au processus.

Comme nous l'avons vu, l'assistance méthodologique fait souvent défaut, et trop peu d'efforts sont fournis pour essayer de donner au développeur un guide lui permettant d'orienter son travail afin de lui éviter au maximum les remises en questions de parties de développement.

Des outils adéquats peuvent aider à diminuer la complexité de la conception des BDs et la rendre plus facile à traiter. Bien que, de par sa nature, elle ne puisse être totalement automatisée (elle contient une grande part d'interactions utilisant le facteur humain, les jugements, l'interprétation, ...), une grande partie de l'aspect laborieux de la collection d'informations à synthétiser en un modèle unique et consistant peut être soulagée par une aide appropriée.

Notre objectif à ce sujet est d'essayer de trouver un moyen de fournir cette aide au concepteur grâce à un Atelier de Génie Logiciel. L'analyse de certains rapports de conférence sur le sujet permet de penser que les outils CASE (Computer Aided Software Engineering) peuvent offrir des fonctionnalités d'aide et d'apprentissage à leurs utilisateurs.

Nous concentrant principalement sur le problème de la conception, les fonctionnalités les plus importantes de l'environnement doivent :

- permettre au concepteur d'accéder à une aide avancée et structurée concernant la méthode de conception et le processus méthodologique associé (étapes, séquences, documentation, ...).

- offrir des exemples de conception pouvant être revus selon un processus méthodologique particulier.

- fournir une aide concentrée sur le processus méthodologique durant toute la session de conception au sein de l'outil, en vérifiant les actions de l'utilisateur.

- proposer des moyens d'évaluation de la qualité de la conception (par exemple des analyseurs statiques).

Nous essayerons de montrer comment fournir une assistance au développeur au sein d'un tel outil en y intégrant ce que nous appellerons les concepts méthodologiques de contrôle de processus. Le rôle de tels moyens est de forcer et d'aider l'utilisateur à respecter et suivre une méthode qui aura été spécifiée et intégrée au sein de l'outil.

Comme nous l'avons dit aussi, nous aurons auparavant décrit un langage de spécification de méthode. Il nous restera alors à définir une architecture permettant d'intégrer cet aspect méthodologique au sein d'un outil (en l'occurrence l'outil DB-MAIN) après en avoir décrit les principes fondamentaux nécessaires.

Conclusion

Le processus de conception d'une base de données a pour objet de récolter les informations pertinentes et les contraintes de fonctionnement relatives à une organisation, pour les transformer et les implémenter dans un SGBD. Après avoir décrit un peu plus précisément en quoi il consistait, dans la première partie du chapitre, nous avons présenté différentes réalisations permettant de proposer ce qui constitue l'outil principal du concepteur pour ce processus, à savoir le modèle de représentation de données. Nous avons analysé les principaux, mais il faut savoir qu'un certain nombre d'autres ont été proposés. Parmi ceux-ci, citons encore le "Role Model" [Bach77], TAXIS [Mylo78], le "Entity Model" [Pal78], le "Data Abstraction Model" [Smith77], ou le "Semantic Data Model" [Schm75]. Notre objectif n'étant pas de les décrire tous, la bibliographie permettra au lecteur intéressé de trouver les précisions supplémentaires qu'il souhaiterait obtenir.

La dernière partie de cette section consistait enfin à décrire les méthodes utilisées pour réaliser le processus de conception. Encore une fois, vu leur nombre et la difficulté de les décrire en détails, nous avons plutôt présenté l'approche générale utilisée par ces méthodes, qui consiste à découper le processus de conception en différentes étapes. Ici aussi la littérature permettra d'apporter des informations complémentaires.

Dans la deuxième section du chapitre, nous avons présenté les objectifs qui guideront le reste de ce travail, à savoir principalement la modélisation du processus de conception, ensuite la définition d'un langage de spécification de méthode et l'intégration des concepts de méthode au sein d'un atelier de génie logiciel.

Chapitre 3

Modèle de représentation d'un processus

Introduction

Ce chapitre est consacré à la description d'un modèle de représentation de processus.

La première section du chapitre introduit le sujet en précisant quelles sont les raisons et les objectifs d'une modélisation de processus, ainsi que certaines caractéristiques à observer pour avoir un modèle efficace. A la suite de ces considérations, nous proposons une vue globale du modèle de représentation tel que nous le concevons. Celui-ci est composé de deux parties concernant d'une part la **représentation de méthodes** et d'autre part la **représentation des historiques**. Les deux sections qui suivent ont pour objet de présenter chacun des modèles correspondant à ces niveaux de représentation.

Enfin la dernière section concerne l'étude d'un langage de description de méthode. Elle a pour but de décrire la syntaxe et la sémantique permettant de donner à l'ingénieur méthode les moyens de spécifier un plan de travail pour guider le concepteur dans son développement.

3.1 Processus et modélisation

Le développement de systèmes logiciels à partir des besoins initiaux est souvent rempli de problèmes, qui se manifestent par des plans d'exécution non suivis, des excès dans les budgets alloués, des systèmes qui ne correspondent pas, dans une plus ou moins large mesure, aux véritables exigences du client, ... Bien qu'ils proviennent parfois aussi d'une définition inadéquate, incomplète, erronée ou ambiguë de ces besoins, ces problèmes sont souvent attribués à une mauvaise définition et à une compréhension erronée des processus utilisés pour les analyser, les spécifier et les transformer en produits correspondants.

Le "software process modeling" est considéré comme une technique relativement nouvelle qui a comme objectif de contribuer à l'amélioration des processus de développement et de leurs produits correspondants. Le modèle permet aussi d'aider à faire d'un développement une activité plus fiable et plus productive. Il n'est pas un produit final, mais un outil qui permet de représenter la réalisation ou l'instanciation de son objet avec tous les aspects qu'elle comporte.

Etant donné que c'est une discipline encore évolutive, l'introduction et le développement de nouveaux modèles peut conduire à des analyses qui permettent souvent de tirer des leçons intéressantes et d'apporter des éléments importants sur la façon de représenter au mieux les éléments

constituant l'objet de la modélisation. Il est donc intéressant d'examiner les nouveaux concepts introduits par ces modèles pour essayer d'en retirer des renseignements utiles concernant l'essence et les défauts d'une modélisation.

Un système peut être défini comme "*un ensemble organisé de doctrines, idées, ou principes habituellement utilisés pour expliquer l'aménagement ou le fonctionnement d'un tout systématique*" [Web85].

Un processus de conception est défini comme un cadre technique et de décision établi pour appliquer des gens, des méthodes, des outils et des pratiques au développement et à l'évolution de logiciels ou de produits. La définition d'un processus ne fait pas que préparer à des éventualités probables, elle fournit aussi des mécanismes pour un apprentissage organisé.

De ces définitions, il est évident que l'on peut considérer un processus comme un système. Les principes de conception tels que la modularité, l'abstraction, la structure, etc... sont bien établis comme des mécanismes d'importance fondamentale pour la spécification et la conception de systèmes. Il est donc raisonnable de supposer que ces principes seront également valides dans la spécification et la définition des modèles des processus.

Une architecture de processus établit la structure, les standards, et les relations entre les différents éléments intervenant dans le processus. Elle constitue "*un cadre de travail au sein duquel chaque processus peut être représenté*" [Hum88]. Il est donc possible dans une telle architecture de définir différents processus particuliers, et notre modèle de processus constitue une concrétisation spécifique d'une telle architecture.

Alors que des modèles peuvent être construits à n'importe quel niveau d'abstraction, l'architecture doit fournir les éléments, les standards et le cadre de travail structurel pour un affinement à tous les niveaux de détails.

Modéliser un processus de développement consiste à donner une représentation purement descriptive sous forme abstraite, généralement simplifiée de celui-ci. Dans le but d'assister la compréhension et le raisonnement sur les activités de création et de développement d'un système, un modèle doit représenter les différents composants d'une catégorie de processus particuliers, et être suffisamment spécifique pour permettre de raisonner sur ceux-ci [Wile86].

Ceci implique qu'il doit non seulement représenter la manière dont les choses se passent, comment les étapes d'un développement sont exécutées, mais aussi pourquoi elles le sont. Beaucoup de modèles décrivent comment un processus est exécuté mais ne parlent pas du pourquoi de cette exécution. La notion de justification qui permet d'exprimer les raisons d'une manière d'agir et les buts poursuivis a aussi de l'importance et doit avoir sa place dans le modèle.

D'un point de vue théorique, le modèle abstrait expose les différents éléments qui permettent de représenter l'exécution d'un processus de développement, les différentes étapes du processus grâce à la définition des objets qu'elles utilisent, ainsi que la manière dont elles sont réalisées (quand et comment).

Il est important de voir qu'un modèle complet d'un processus complexe peut lui-même être complexe. L'approche traditionnelle "orientée-tâche" est une conséquence naturelle de la vision que l'on a en général du travail (c'est ce qui a conduit à la définition du 'Waterfall Model'). Mais bien que cette structure soit appropriée et relativement facile à comprendre quand les tâches sont simplement connectées, elle devient rapidement dépassée lorsque le nombre de séquences possibles de tâches augmente.

Le danger est que, en essayant d'utiliser ce type de modèle dans de telles situations, il doit être simplifié pour permettre la compréhension, et ces simplifications tendent à limiter les possibilités d'agencement des tâches.

Un autre point concerne l'exécution réelle du processus. Lorsque le modèle est utilisé pour guider ou contrôler l'exécution détaillée du processus, il doit être suffisamment clair et contenir plus qu'une optique orientée-tâche. Un modèle complet doit reprendre les aspects fonctionnels, comportementaux, structuraux et conceptuels des éléments du processus. Pour le traitement du processus, un modèle plus simple peut être approprié tant qu'il n'implique pas de contraintes artificielles sur son exécution.

Notre travail est guidé par le fait que la qualité d'un produit est largement déterminée par la qualité du processus qui a permis de le développer. L'amélioration d'un produit peut donc effectivement être réalisée par une amélioration du processus de construction. Par conséquent, un des principaux objectifs du travail est de donner un moyen qui peut avoir pour conséquence de faciliter et améliorer l'évolution d'un processus vers une plus grande efficacité, capacité et fiabilité. Cette évolution conduit directement à une amélioration des produits correspondants.

Modéliser et analyser les processus de développement est devenu une activité très importante parce que l'arrivée des nouvelles technologies force les développeurs à décider comment les utiliser au mieux. La modélisation supporte les objectifs d'amélioration des processus en fournissant un mécanisme pour :

- Enregistrer et comprendre le processus de base.
- Evaluer, communiquer et promulguer les améliorations de celui-ci.

A un niveau de détails plus élevé, et avant d'établir des critères permettant de définir une bonne approche de modélisation, il est nécessaire de définir les différents rôles que peut jouer le développement d'un modèle de représentation de processus.

1. Permettre une communication effective au sujet du processus.
2. Faciliter la réutilisation du processus.
3. Supporter l'évolution du processus.
4. Faciliter la conduite du processus.

Le premier rôle permet une véritable communication en ce qui concerne le processus : la communication entre les utilisateurs, les développeurs, les directeurs, les chercheurs,... Il facilite la compréhension et fournit une base précise pour l'exécution du processus et son automatisation. Il est aussi crucial que le personnel soit capable de voir le processus à différents niveaux d'abstraction.

La réutilisation, le deuxième rôle, permet à un processus spécifique d'être exécuté de manière répétitive et fiable dans plusieurs projets. Osterweil considère dans [Ost87] que " ... *the most important benefit of process [modeling] is that it offers the hope that software processes themselves can be reused*". Le développement d'un processus prend du temps et coûte relativement cher, de telle sorte que peu d'équipes de recherche peuvent se permettre le temps et les ressources pour en développer complètement un nouveau.

Le troisième rôle consiste à supporter l'évolution du processus (1) en servant de "magasin" pour les modifications, les leçons apprises, ... et (2) en analysant l'efficacité de modifications en laboratoire ou en environnement simulé avant de les implémenter. Des définitions précises, facilement comprises, extensibles, et réutilisables fournissent des moyens réels pour l'apprentissage du processus. Un bon modèle de processus est donc un élément important pour l'amélioration de celui-ci.

Le dernier rôle a pour but de faciliter le planning, le contrôle et la mise en oeuvre du processus. Ceci est réalisé par une compréhension accrue du processus, la conformité aux définitions, des simulations, des analyses, ... Une conduite effective demande une bonne compréhension des plans et l'habileté pour précisément caractériser leurs statut par rapport au processus lui-même. Le modèle fournit potentiellement un cadre de travail pour définir de façon précise les critères de statut du processus et les mesures d'évaluation.

De manière à réaliser ces objectifs, nous pouvons déduire que le modèle de représentation doit posséder certaines caractéristiques sous principalement trois aspects. Ils doivent :

1. Permettre une représentation du travail qui est ou qui doit être réellement réalisé.

2. Fournir un cadre flexible et facilement compréhensible, ainsi que relativement puissant pour représenter et améliorer le processus.

3. Pouvoir être raffiné jusqu'à n'importe quel niveau de détail souhaité.

Nous anticipons déjà en disant que ces caractéristiques peuvent aussi être vérifiées de manière efficace dans une approche d'automatisation.

Les deux premières apparaissent généralement comme évidentes, mais la troisième est souvent négligée. Il est nécessaire d'avoir des définitions précises des tâches pour permettre un développement efficace d'un outil et d'un environnement. Comme pour toute application manipulant des données, des tâches mal définies entraînent une mauvaise définition des requirements et un système inefficace. L'automatisation d'un processus consiste également en un développement d'application et doit donc se baser sur des modèles de processus précis à un niveau relativement élevé de détails.

Enfin, nous pouvons dire que pour réaliser les quatre objectifs présentés ci-avant, un modèle de processus doit aller plus loin que la simple représentation. Il doit permettre une analyse étendue du processus, ainsi que des prédictions en ce qui concerne les conséquences de modifications ou améliorations potentielles. L'étude des prévisions permet de voir le comportement du processus en réponse à différents événements ou circonstances, par exemple de prévoir certaines réponses aux questions posées lors de modifications de tâches, ou d'insertions de nouvelles technologies. Les moyens d'analyse contiennent aussi des tests en ce qui concerne la consistance, la complétude et la correction. Ces objets sont critiques pour la validité du modèle et du processus réel qu'il représente.

Afin de réaliser au mieux les objectifs présentés et observer les caractéristiques principales qui en découlent, nous pouvons encore ajouter que le modèle doit répondre aux critères suivants :

Tout d'abord la clarté : la complexité croissante des processus conduit à une complexité visuelle des modèles. Une fois que cette complexité atteint une certaine limite, les modèles résultants deviendront difficiles voire impossible à comprendre et à traiter. La charge d'information est aussi importante dans un modèle. Souvent, les développeurs de modèles ont tendance à montrer tous les détails du processus sur un même diagramme. Bien que cela soit désirable pour les processus assez simples, cette approche peut parfois conduire à une surcharge d'information pour ceux qui utilisent le modèle lorsqu'ils traitent des processus complexes.

Ensuite la connexion entre les concepts : La corrélation entre la description comportementale et la partie fonctionnelle du processus doit être visible. Les descriptions comportementales doivent montrer explicitement le contrôle des activités, la description fonctionnelle doit aussi décrire l'information et ses caractéristiques. Les deux parties doivent être mises en correspondance étroite (pas d'associations vagues, informelles, ...) pour éviter à l'utilisateur de devoir faire de trop gros efforts pour fournir les données nécessaires à la réalisation de sa tâche.

3.2 Modèle de représentation d'un processus

Les techniques de modélisation utilisées doivent être applicables pour la modélisation de processus de n'importe quelle complexité. Elles doivent permettre d'améliorer l'esthétique et la maniabilité des modèles complexes sans introduire d'inconvénients sérieux.

A la suite des considérations que nous venons de faire, nous avons de bonnes raisons de croire qu'un modèle doit contenir une approche orientée-tâche, mais ne peut s'y limiter sans s'avérer insuffisant et incomplet dans le domaine traité ici. Nous proposons par conséquent un modèle de représentation qui utilise, en plus de l'aspect "tâche" du processus, une description comportementale de celui-ci.

Une caractéristique principale d'un modèle est son niveau d'abstraction et les propriétés qui sont formalisées. Un programme en langage machine par exemple peut être considéré comme un objet formel et abstrait, mais son niveau d'abstraction est plus faible que celui d'un programme en langage de programmation de haut niveau. Aussi, différents types de modèles sont appropriés pour la formalisation de la correction fonctionnelle, l'efficacité algorithmique, le comportement interactif,... ou pour la spécification, la conception, l'analyse et la vérification de systèmes.

La description des concepts permettant de modéliser un processus de conception telle que nous allons la présenter propose une représentation de ceux-ci sur deux niveaux différents.

Le premier niveau concerne une représentation des concepts nécessaires à la définition d'une méthode. Ce **modèle des méthodes** donne une description générale qui établit la structure et les relations entre les objets permettant à un ingénieur méthode de prescrire une manière de travailler.

Le deuxième reprend l'ensemble des éléments décrivant les phénomènes réellement mis en oeuvre au cours d'un processus. Ce **modèle des historiques** permet de donner une représentation des instances des différents objets intervenant dans une exécution concrète d'un processus, qui peut constituer une application d'une méthode spécifiée selon le modèle du premier niveau, et qui a

conduit à la définition de produits faisant partie de l'univers réel comme un schéma conceptuel, un texte de programme, ...

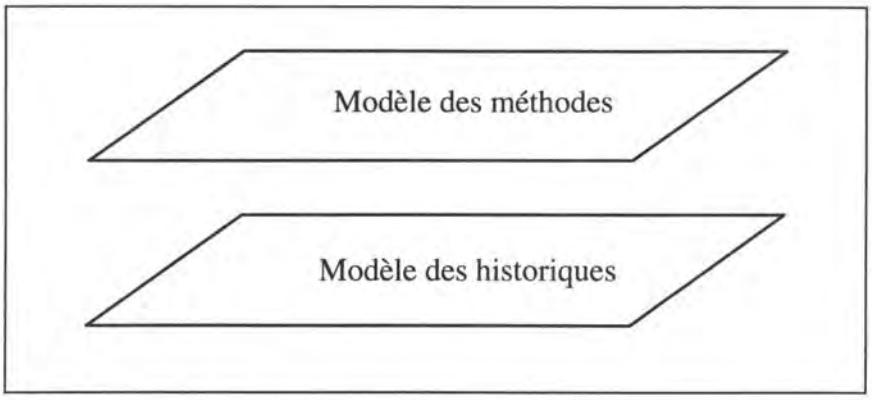


Fig. 3.1 Les deux niveaux de représentation d'un processus

Les éléments proposés au niveau des instances doivent être conformes à ceux définis au niveau des méthodes, qui doivent aussi couvrir les caractéristiques d'un développement réel. Sans cela, les deux modèles risquent d'être incompatibles, et il sera difficile de faire la concordance entre la partie concernant la méthode et celle concernant son application dans un processus réel.

Une telle distinction entre des éléments d'un niveau abstrait utilisés pour décrire un processus générique couvrant différentes situations possibles de développement, et ceux d'un niveau d'instances qui représentent les objets d'une situation concrète, permet d'analyser la relation entre le modèle de méthode et la réalisation d'un processus. Par exemple, un modèle de méthode avec lequel la plupart des réalisations concrètes ne sont pas concordantes ou ne peuvent être représentées est certainement insuffisant, et devra être amélioré. Un autre exemple est que la représentation d'un certain nombre de réalisations de processus peut donner lieu à un modèle de méthode quelque peu différent de celui défini au départ, et fournir des renseignements supplémentaires utiles pour l'amélioration de celui-ci.

Les deux sections qui suivent ont pour but de donner une description des modèles de représentation aux deux niveaux proposés. La première concerne le modèle des méthodes et donne une architecture générale des objets permettant de les décrire. La seconde traite du modèle des historiques, qui représente les objets nécessaires à la spécification de la trace de l'exécution d'un processus.

3.2.1 Le modèle de définition de méthode

3.2.1.1 Méthodes et conception

Les principes de conception, dans quelque domaine que ce soit, ne peuvent être appliqués au petit bonheur. Chaque système doit être construit de manière disciplinée. Les critères qui permettent de faire d'un processus de conception quelque chose d'ordonné et de cohérent sont généralement incorporés dans ce qu'on appelle des méthodes de conception.

Une méthode regroupe des procédés habituels qu'on peut observer et définir par induction pour les pratiquer ensuite plus sûrement. Elle est le fruit d'une réflexion permanente, construite selon une démarche rationnelle et empirique, déductive et inductive. Une bonne méthode doit s'appuyer à la fois sur les concepts créés par la technologie et sur l'observation et l'analyse scientifique de sa pratique. De plus, elle ne peut servir des buts fondamentalement divergents, ce qui implique des méthodes spécialisées en fonction du type de produit à construire, ou du type de projet.

Une méthode est en grande partie fondée sur la combinaison d'un cadre procédural et de techniques de conduite des tâches qui a pour objectif la réalisation d'un produit qui n'est pas déterminé à l'avance. Le cadre procédural précise les étapes de la méthode et leur enchaînement, les résultats attendus et les arguments nécessaires pour les obtenir, les modèles et langages à utiliser pour la spécification, la conception, le réalisation, l'assurance de la qualité, ... et la définition des termes et des concepts employés. Les techniques de conduite précisent la manière d'exécuter les tâches utilisables durant une ou plusieurs étapes pour obtenir les produits attendus (par exemple technique d'analyse sémantique du discours pour la construction d'un modèle conceptuel des données).

Une méthode de développement supporte normalement deux aspects de la conception. Elle fournit un ensemble de représentations et analyses de différents types de produits, et un support d'heuristiques pour la réflexion au sujet de la conception (décider quoi faire et pourquoi). La plupart des méthodes, même si elles diffèrent dans la rigueur de leurs produits et analyses, et dans la richesse de leur heuristiques, fournissent un certain degré de support pour les deux aspects de la conception.

En résumé, une méthode fournit aux professionnels du développement le cadre technique dont ils ont besoin pour faire un travail de bon niveau, consistant et uniforme. Son rôle est d'essayer d'améliorer la réalisation d'un projet et de faciliter la réalisation des tâches clés. Elle peut en plus être incorporée dans un répertoire de "scénarios" disponibles pour chaque projet.

3.2.1.2 Cadre de processus standardisé

Bien qu'il soit souvent nécessaire de définir les processus spécifiquement à un projet, il existe aussi des raisons pour lesquelles il est intéressant d'avoir un cadre de processus standardisé :

1. La standardisation d'un processus est nécessaire pour permettre son apprentissage, sa direction, son étude et le support par un outil.

2. Avec des méthodes, l'expérience sur chaque projet peut contribuer à une amélioration générale des processus dans l'organisation.

3. Les standards fournissent une base structurée pour l'évaluation.

4. Etant donné que les définitions de processus demandent du temps et des ressources, il est impossible d'en définir des nouveaux pour chaque projet.

5. Parce que certaines tâches de base sont communes à beaucoup de projets, un processus standardisé aura besoin de peu d'effort d'adaptation pour rencontrer les besoins spécifiques à un projet.

3.2.1.3 Modèle de représentation

a. Présentation générale

Présenter un modèle de définition de méthode consiste à mettre en lumière les différents éléments nécessaires à l'ingénieur méthode pour lui permettre de prescrire une manière de travailler. Il doit donner une vue qui puisse montrer comment les développeurs peuvent ou même doivent procéder pour obtenir progressivement un système efficace et fiable qui corresponde aux requirements initiaux. L'ensemble des concepts repris dans le modèle sont utilisés pour décrire la technique qui permet d'obtenir les produits en fonction des arguments, des requirements et des spécifications de départ. Une méthode est en réalité un processus discipliné qui permet de fabriquer des produits, et englobe les différentes traces possibles d'un développement (différents comportements).

La spécification d'une méthode consiste à donner un modèle générique de processus qui couvre différentes situations de développements possibles. Nous allons donc essayer de représenter des objets qui sont une description abstraite de quelque chose qui peut exister lors d'une exécution réelle d'un processus.

Cette spécification peut être considérée comme une collection de morceaux de processus qui seront stockés dans une base de connaissances d'un outil CASE, pour pouvoir servir d'aide automatisée au processus (voir le chapitre 5).

Le modèle a donc un rôle au niveau opérationnel puisqu'il permet d'apporter une aide pour les ingénieurs méthode et les développeurs d'application. Il définit en effet un ensemble de concepts qui sont génériques, c'est-à-dire indépendants de toute méthodologie particulière. L'ensemble de ceux-ci fournissent une base à la définition d'une méthodologie, et peuvent servir pour une

éventuelle aide automatisée à l'exécution du processus. Il peut donc être considéré comme une théorie des plans qui permet de générer la représentation de méthodes pouvant elles-mêmes être exécutées.

Le modèle décrit ci-après reprend les concepts que nous jugeons nécessaires de représenter pour la définition d'une méthode, quelle que soit celle-ci. Il peut paraître minimal et relativement simple, mais nous estimons qu'il est suffisant, du moins dans cette première approche, pour permettre à l'ingénieur de définir les éléments nécessaires pour fournir un plan de travail à suivre par le concepteur.

La figure 3.2 ci-après montre la façon dont ces éléments sont reliés.

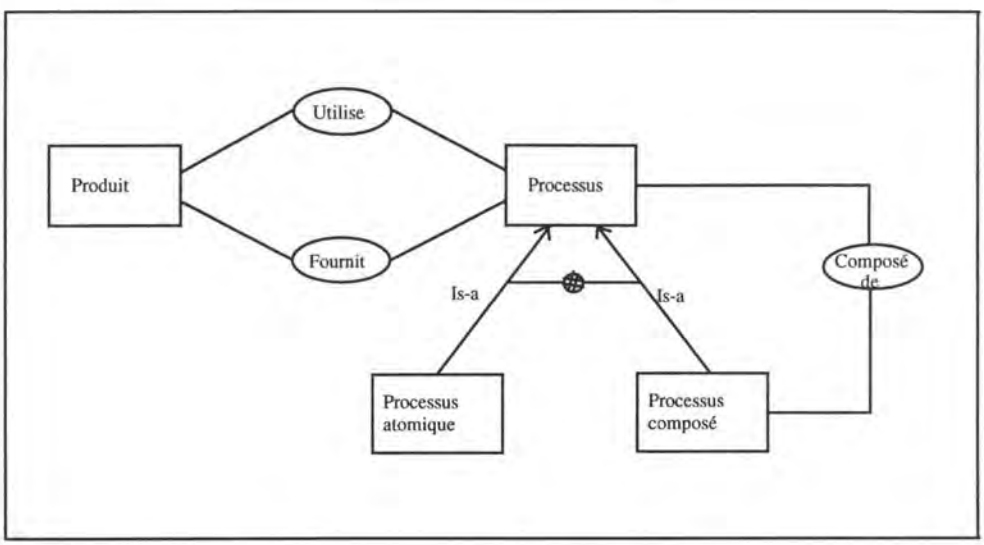


Fig. 3.2 Le modèle de définition d'une méthode

- Un *processus* est la description d'une transformation appliquée sur des produits. Il a pour but de faire évoluer leur spécification de manière à ce qu'elle se rapproche de la description finale, imposée par la méthode et observant les contraintes introduites. Le développement consiste à exécuter un certain nombre de ces processus pour faire évoluer les produits vers leur spécification finale.

- Un *produit* est la description d'un objet réel qui peut être manipulé tel quel ou subir des transformations. L'objectif premier d'un développement est de construire un ou des produits qui correspondent à l'interprétation des spécifications initiales.

Chaque objet ou concept décrit dans le modèle de la figure 3.2 possède un certain nombre d'attributs qui sont des éléments de données, et sa propre valeur pour chacun d'eux. Une classe contient les objets possédant les mêmes noms d'attributs, la valeur de ceux-ci permet de distinguer les objets au sein d'une même classe. Définir une méthode revient finalement à définir une collection d'objets "produits" et "processus" qui seront instanciés lors de l'exécution réelle du processus qui donnera une valeur précise à chacun de leurs attributs.

b. Approche utilisée

L'approche utilisée ici est de type transformationnel. Elle adopte la vision traditionnelle d'un processus comme une fonction entre produits input et output. Une telle approche est abordée dans [Hai92c], et nous estimons que les principes qui y sont proposés s'avèrent intéressants pour notre propos. Un processus est vu comme un ensemble de transformations appliquées successivement à un ou des produits afin que ceux-ci satisfassent à certaines contraintes définies soit par une méthode, soit par le développeur lui-même.

Le but des systèmes transformationnels est de transformer un état initial en un état final. Ils utilisent la logique basée sur une formule de la forme :

$$\{P\}S\{Q\}$$

P représente la situation à partir de laquelle le processus va être exécuté.
Q représente la situation après l'exécution du processus.
S représente le processus, c'est-à-dire l'ensemble des transformations qui permettent de produire Q à partir de P.

P et Q ne constituent pas seulement la description de produits. Ils sont aussi l'expression des états dans lesquels se trouvent ceux-ci et peuvent être considérés respectivement comme préconditions et postconditions du processus S. L'interprétation de la formule précédente est :

"Le processus S ne peut être appliqué que sur des produits possédant les caractéristiques définies par P, et doit fournir des produits tels qu'ils sont décrits par Q".

Dans l'exemple d'une implémentation transformationnelle, les différents produits ont une sémantique formelle et les règles de dérivation sont strictement définies.

La conception d'un produit est un effort de traduction de notre représentation mentale de ce produit en une représentation adaptée de telle manière que d'autres puissent la comprendre aisément. Etant donné un ensemble de requirements comme des propriétés, des comportements,... nous devons construire et représenter, sous forme plus ou moins standard, les différents objets qui satisfont à ces contraintes.

Le processus de conception est en fait un mécanisme relativement complexe dans lequel le concepteur tente de construire les produits, et opère, à la suite de décisions, toutes sortes de transformations pour le raffiner et l'améliorer jusqu'à ce que celui-ci satisfasse aux requirements.

3.2.1.4 Description des éléments du modèle

a. Processus

Description

Un processus est la description abstraite d'une partie de travail à exécuter. Il représente une transformation globale appliquée sur un produit fourni en entrée avec l'intention de fournir un produit résultat. Cette description est faite indépendamment de toute situation spécifique dans une exécution réelle d'un processus. Elle ne contient aucune information concernant le moment de l'exécution, ni l'agent qui va la réaliser, et ne représente qu'un processus de manière générique, qui fera l'objet d'une instanciation.

Cette caractéristique n'empêche quand même pas l'introduction de relations temporelles ou causales entre les différents processus spécifiés, ainsi que la définition de conditions qui peuvent restreindre leur application.

Nous verrons comment, dans la définition d'un processus, il est possible d'introduire la notion de procédure qui peut imposer un certain ordre sur l'exécution de processus, et donc définir des relations entre ceux-ci pour leur mise en oeuvre. Nous verrons aussi comment imposer certaines conditions à l'exécution d'un processus, notamment grâce à la définition de préconditions.

Un processus est réalisé grâce à un ensemble d'opérations qui sont exécutées de manière à former les différentes étapes permettant de transformer un ou des produits utilisés comme arguments.

Définir un processus de transformation de produit est une tâche qui peut être réalisée de façon relativement "systématique" :

1. Identifier les produits utilisés comme arguments.
2. Identifier les produits qui constitueront les résultats.
3. Définir la procédure qui permet de transformer les arguments en résultats.

Cette procédure paraît relativement simple conceptuellement, mais peut devenir assez compliquée pour des processus pour lesquels il peut avoir un grand nombre d'actions à définir.

Un processus est complètement spécifié lorsqu'une valeur est fournie aux attributs suivants :

- le nom
- la description informelle
- la description des produits input
- la description des produits output
- la procédure de réalisation

Le *nom* du processus permet d'identifier celui-ci parmi l'ensemble de tous les processus qui sont définis au sein d'une méthode.

La *description informelle* permet d'expliquer en langage naturel ce que fait le processus. Le choix d'un nom suffisamment explicite n'étant pas toujours facile, cette description permet de donner des renseignements supplémentaires sur son rôle.

La *description des produits input* permet de spécifier les types de produits que le processus utilise en entrée. Cette partie de la spécification constitue une précondition à l'exécution de celui-ci, si les produits fournis au cours de l'exécution ne sont pas du type spécifié (ne vérifient pas les propriétés de celui-ci), le processus ne pourra pas être exécuté.

La *description des produits output* permet de spécifier les types de produits que le processus fournit comme résultat. Elle constitue sa postcondition puisqu'elle contient les propriétés que posséderons ces résultats.

La *procédure de réalisation* du processus est la description des transformations utilisées pour obtenir les produits résultats à partir de ceux fournis comme arguments. Il s'agit d'un algorithme qui spécifie les différentes instructions permettant d'exécuter la tâche. Ces instructions peuvent être soit des fonctions directement fournies par un système quelconque, soit les noms d'autres processus qui seront à leur tour spécifiés de la même manière.

Décomposition d'un processus

La définition d'un processus consiste à préciser les différentes transformations utilisées pour obtenir les produits résultats à partir des arguments. Le modèle introduit la spécialisation d'un processus en "processus atomique" et "processus composé". La concrétisation d'un objectif par l'application d'un processus sur les produits peut représenter un travail relativement important. Il peut alors être décomposé en sous-processus qui, exécutés un à un, permettent d'obtenir le résultat attendu par le premier.

Par exemple, la création d'une relation entre deux types d'entité dans un schéma ERA pourra être réalisée par la création du type d'association, la définition du premier rôle, la définition du deuxième rôle, ...

C'est pour cette raison que nous avons introduit dans le modèle cette spécialisation d'un processus. Un processus atomique correspond à une transformation d'un produit qui ne doit pas être décomposée en plusieurs phases. Le processus composé est réalisée par une séquence d'autres actions.

Cette décomposition est un avantage pour la définition des processus mis à la disposition du concepteur. En effet, nous verrons que le niveau de précision dans la définition de ceux-ci est importante pour l'utilisateur.

La spécification d'un processus introduit donc un certain nombre de sous-processus, et l'idée est de le spécifier par une approche déductive ou descendante qui est un raffinement itératif dans lequel chaque étape est graduellement décomposée en sous-étapes plus simples. Le principe est le suivant :

1. Spécifier le processus initial, celui qui correspond à fournir les produits finaux comme résultat. Cette description met à jour des étapes intermédiaires qui peuvent elles-mêmes être considérées comme des sous-processus.

2. Appliquer itérativement cette étape à chaque processus défini jusqu'à ce que la décomposition fournisse des sous-processus dont la spécification soit suffisamment simple. (cfr. "granularité et précision" ci-après)

La spécification d'un processus consiste finalement à décrire un ensemble d'actions qui sont structurées en arbre dont le sommet est le processus original, et qui est décrit comme la transformation des spécifications initiales en produits finaux. A chaque niveau de l'arbre, un processus représente une étape qui, associée aux autres de même niveau, permet de réaliser le processus du niveau supérieur.

Chacun de ces composants peut être considéré comme un processus et être spécifié de la même manière par une nouvelle décomposition au niveau inférieur.

A chaque niveau de l'arbre, nous avons donc un ensemble d'unités qui décrivent chacune un sous-processus qui représente un composant d'une unité définie au niveau supérieur. Les arguments et les résultats utilisés et fournis par le sous-processus appartiennent à des ensembles issus des domaines déjà définis au niveau supérieur. De manière itérative, chaque sous-processus est décrit par l'introduction d'autres plus précis et de données ou produits intermédiaires.

La décomposition d'un processus s'arrête lorsque les feuilles du sous-arbre décrivant ce processus correspondent à une transformation terminale d'arguments en résultats, qui n'a plus besoin de subir une expansion à travers d'autres processus. Un processus peut être considéré comme une transformation terminale s'il constitue un processus de base tel qu'il est défini ci-après. Le spécifieur peut aussi décider d'arrêter la décomposition s'il considère les produits utilisés par le processus comme élémentaires.

Les relations entre les différentes unités issues de la décomposition d'un processus peuvent avoir plusieurs interprétations, elle peuvent être définies au moyen d'opérateurs tels que ou, et, puis,...

Il est donc possible de donner une structure et une sémantique beaucoup plus riche à l'arbre que celle d'une simple décomposition ayant pour but un plus grand niveau de détail. Une représentation graphique d'une telle structure permet de donner une vue d'ensemble d'une décomposition, ainsi que de sa signification. L'exemple ci-dessous est tiré de la description du processus de conception de bases de données qui fera l'objet de l'étude de cas au chapitre 4.

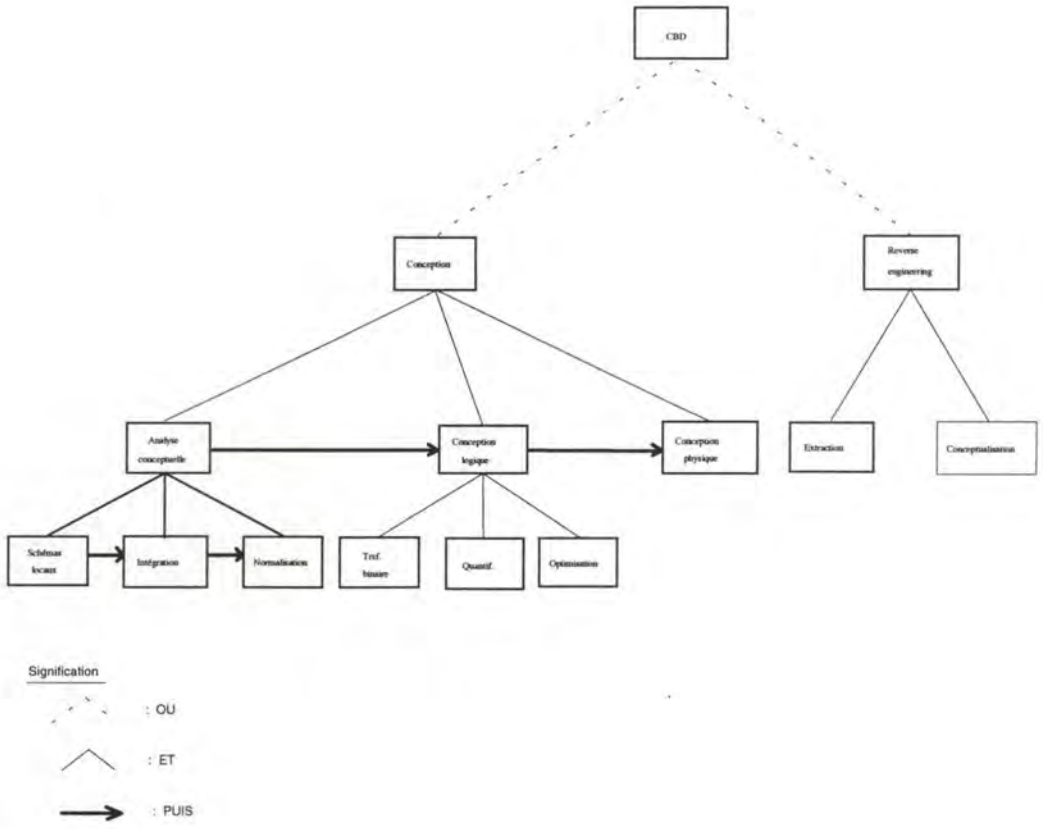


Fig. 3.3 Représentation graphique d'une partie de l'arbre de décomposition du processus de conception de BD

Cette décomposition en sous-processus nous permet d'introduire ici la distinction entre un processus de base et un processus "user defined".

Un processus de base est une fonctionnalité offerte directement par un atelier de conception. Sa spécification est connue et elle ne doit donc plus être décrite par l'ingénieur méthode. Sa stratégie n'est pas explicite en ce sens que l'utilisateur n'a pas accès à sa définition et ne voit donc pas comment il est réalisé. Un processus de base est une opération neutre, il ne concerne aucune méthodologie particulière et peut être simplement considéré comme un service mis à la disposition des autres processus, ou de l'utilisateur.

Un processus "user defined" est une procédure construite par le spécialiste méthode qui spécifie le processus de conception. Elle est spécifique à une méthode ou à une famille de méthodes, et sa spécification et la stratégie utilisée pour la réaliser sont connues et peuvent être consultées.

Granularité et précision

Un processus est décomposé en sous-processus de plus en plus simples. La taille de ces éléments permet d'introduire la notion de granularité dans la description d'un processus. La volonté de définir une granularité de plus en plus importante permet d'obtenir une plus grande précision dans la manière dont les différentes étapes d'un processus sont décrites.

Lorsque le niveau d'abstraction de la représentation des procédures est trop élevé, il ne permet pas de fournir suffisamment de détails pour guider correctement son exécution, et l'effort de l'utilisateur devra être plus important pour exécuter des actions précises et appropriées suivant l'état du système.

D'un côté, il peut être intéressant d'avoir une description moins détaillée, ce qui permet une plus grande flexibilité à l'utilisateur, notamment lorsqu'il est amené à prendre des décisions compliquées pour choisir le processus à exécuter. Mais d'autre part, modéliser un processus de façon plus précise permet de représenter les différentes alternatives qui s'offrent à l'utilisateur, ainsi que de l'aider à mieux choisir parmi celles-ci.

Curtis, Kellner et Over affirment dans [Curt92] que les processus automatisés demandent une représentation plus précise, alors que ceux devant être exécutés par une personne peuvent être décrits avec moins de détails.

b. Produit

Description

Un produit dans la définition d'une méthode est une description abstraite des instances potentielles ou réelles de produits. Il représente tout objet utilisable par un processus et constitue un concept générique qui pourra faire l'objet d'une instanciation lors de l'exécution d'un processus réel.

La spécification des données est très importante dans la description d'un processus de conception ou même d'un système quelconque. Les spécifications des opérations sont en partie dérivées de la définition des données, et celle-ci implique certaines structures et autres caractéristiques sur ces données.

Certains modèles considèrent qu'une tâche utilisant des produits doit être terminée avant que la suivante ne commence, et que les produits mis en oeuvre ont la durée de vie correspondant à la durée d'exécution de la tâche. Dans notre modélisation, un produit utilisé à quelque moment que ce soit existe durant tout le développement, même s'il n'est plus utilisé après un certain temps.

Dans l'exemple d'une conception de base de données, le schéma conceptuel sert de base à la construction du schéma logique, et il se peut qu'aucun processus ne l'utilise plus par la suite. Ce produit "schéma conceptuel" existera cependant durant tout le développement et pourra être considéré comme disponible dans toutes les étapes.

La définition d'un produit au sein d'une méthode permet de préciser les préconditions et postconditions d'un processus tel que nous l'avons vu dans la définition de celui-ci. En effet, celle-ci sert à la fois à introduire un type de produit qui sera instancié au cours d'un processus réel, mais elle permet aussi de définir des préconditions à l'exécution du processus, ainsi que les postconditions sur ses résultats, grâce notamment aux différents attributs qu'elle utilise et en particulier les propriétés.

Les produits doivent être spécifiés de manière rigoureuse, et nous allons montrer une façon de les représenter en décrivant les différents attributs qui les constituent, à savoir :

- Un nom
- Une description
- Une structure
- Des propriétés

Le *nom* du produit permet de désigner celui-ci parmi tous ceux appartenant à l'ensemble des produits relatifs à la méthode.

La *description* permet de donner en langage informel une idée du contenu du produit. Il n'est en effet pas toujours facile d'être suffisamment explicite dans le choix d'un nom pour un produit, et cet attribut donne plus de détails sur son contenu. Cette description peut aussi donner quelques précisions sur la structure ou les propriétés du produit.

L'attribut de *structure* permet d'introduire les différents composants utilisés dans la définition du produit.

Les *propriétés* sont la description des contraintes qui établissent les caractéristiques que doit observer chaque instance du produit. Ces propriétés permettent entre autres d'exprimer les requirements imposés par la méthodologie. Nous avons vu que la construction d'un produit est constituée d'un ensemble de transformations qui doivent permettre d'aboutir à un produit possédant certaines caractéristiques. Une partie de ces caractéristiques peuvent être définies dans les propriétés du produit, et tous les produits qui en sont une instance doivent observer ces propriétés et donc les requirements imposés sur celui-ci avant et après chaque opération qui les concerne.

Les propriétés d'un type de produit peuvent concerner deux domaines différents de sa définition. D'une part, elles peuvent porter sur la structure du type de produit. Dans ce cas, elles indiquent des conditions sur les différents concepts utilisés pour définir cette structure, ou sur les relations entre ceux-ci. D'autre part, elles peuvent introduire des précisions sur la valeur ou le contenu des constituants d'un type de produit.

Dérivation d'un produit à partir d'un autre

Il peut être intéressant pour la définition d'un produit d'avoir des mécanismes qui permettent de récupérer certains éléments qui ont déjà fait l'objet d'une spécification, et qui peuvent être réutilisés tels quels dans une autre définition.

Nous introduisons ici quatre mécanismes que nous jugeons utiles pour pouvoir dériver la définition d'un produit à partir de celle d'un autre dont il partage certaines caractéristiques.

1. Récupération des définitions

Ce mécanisme est le plus général. Il permet d'importer toutes les définitions concernant les éléments (propriétés, concepts, ...) qui ont été introduits dans le premier produit. La définition d'un produit à partir d'un autre se fait toujours par la récupération des définitions, ce n'est que par la suite que les éléments indésirables sont "élagués", ou qu'on rajoute des concepts supplémentaires.

2. Modification de définition

Certaines parties de la définition récupérées d'un produit pour en définir un autre peuvent ne pas correspondre à ce que l'ingénieur souhaite pour ce dernier. Il doit donc avoir la possibilité de les modifier et les adapter pour obtenir la spécification voulue.

Dans le cas où les définitions récupérées sont trop générales, et pas suffisamment précises, la modification consistera en une restriction portant sur les éléments concernés. Il s'agira par exemple d'imposer un nombre de rôles égal à 2 par type d'association dans un schéma ERA alors qu'il pouvait aller jusqu'à 5 dans le premier produit.

Un autre cas est celui où les définitions du premier produit sont un peu trop précises ou restrictives. L'ingénieur a alors la possibilité de les modifier par généralisation. Le premier produit pouvait par exemple imposer que tous les rôles soient joués par un seul type d'entité. Une généralisation possible concernant cette propriété consistera à autoriser dans le nouveau produit des rôles "multi-types d'entité".

La modification qui consiste à donner une nouvelle définition à une caractéristique héritée lorsque celle-ci se révèle inadéquate porte principalement sur les propriétés des produits. En effet, si les concepts du premier produit doivent être modifiés, il est peut-être préférable d'en définir un nouveau plutôt que de récupérer les définitions de celui-ci.

3. Ajout de définitions

Il est clair que récupérer les définitions d'un produit est nettement insuffisant, car il risque de manquer un certain nombre de concepts ou propriétés. L'ingénieur doit donc pouvoir ajouter de nouveaux concepts à la structure de son produit, ou de nouvelles propriétés à celles qu'il possède déjà.

On peut prendre par exemple le cas du modèle logique, qui serait défini à partir du modèle conceptuel auquel on aurait ajouté le concept d'accès.

Le nouveau type est doté d'autres concepts et/ou propriétés qui représentent ses caractéristiques propres.

4. Renommage

Ce quatrième mécanisme permet à l'ingénieur de modifier l'appellation de certains concepts qu'il a récupéré d'un produit précédent. Il se peut que le nouveau produit défini utilise les mêmes concepts ou la même structure que le premier, mais que ceux-ci portent un nom différent. Considérons par exemple le cas d'un modèle Entités/Associations dont un concept utilisé est le "type d'entité". Le concept équivalent dans un modèle CODASYL portera le nom de "type de record".

Héritage

L'utilisation du mécanisme de dérivation d'un produit à partir d'un autre dans la définition d'une méthode permet d'introduire la notion d'héritage entre les produits.

Un produit dont la définition est dérivée d'un autre hérite des caractéristiques de celui-ci, qui peuvent être modifiées par la suite. On ne peut cependant pas donner à cette relation une interprétation habituelle telle que l'interprétation logique, ensembliste, ... comme définies dans [Bra83]. En général, celles-ci considèrent qu'un produit défini à partir d'un autre ne peut en constituer qu'une spécialisation, ou une restriction, ce qui n'est pas le cas pour la dérivation de produits telle que nous l'avons définie. Mais le fait de dériver la définition d'un produit P1 pour spécifier un produit P2 nous permet quand même de considérer ce dernier comme "une sorte de" P1 "à quelques modifications près" (qui correspond dans une plus ou moins large mesure au point de vue conceptuel de l'interprétation de la relation d'héritage toujours décrite dans [Bra83]). Nous conserverons donc quand même cette notion d'héritage, tout en étant conscient de faire un certain abus de langage.

Cette interprétation que nous en donnons est un peu plus flexible et a l'avantage d'éviter certains problèmes. En effet, , pour reprendre l'exemple de P1 et P2, rien n'empêche à priori une même contrainte d'apparaître à la fois dans la spécification de P1 et dans celle de P2 défini comme une spécialisation de P1, mais avec un paramètre différent.

On pourrait par exemple définir un type de produit "ERA" avec une contrainte spécifiant un nombre d'accès égal à zéro, alors que le type de produit "ER-binary-logical" qui est défini à partir de celui-ci permet la définition des accès.

Définir P2 comme un sous-type de P1 ne signifie pas nécessairement que les produits de type P2 sont aussi de type P1, et l'interprétation (informelle) donnée à la relation d'héritage permet de rester cohérent face à ce genre de problème.

Cette façon de voir la relation entre les différents produits nous permet de considérer une structuration en graphe de ceux-ci, de telle sorte qu'un sommet représente un produit, et qu'il existe un arc entre deux sommets représentant les produits P1 et P2 si P2 est défini à partir de P1. Cette structuration entraîne une modularité importante, la description des produits peut être divisée en parties relativement indépendantes regroupant ceux-ci par "affinité", puisqu'il sont issus d'un même sommet. Chaque produit appartenant à un tel groupe partage la majorité des caractéristiques de ce sommet, et une modification de ce sommet n'entraîne que des modifications mineures, et uniquement sur les produits représentés par les sommets adjacents, c'est-à-dire qui en sont dérivés.

3.2.2 Le modèle des historiques

3.2.2.1 L'historique d'un processus

Etant donnés certains facteurs tels que les coûts, les horaires, les ressources, etc..., il est souvent préférable d'améliorer des anciens systèmes plutôt que d'en développer de nouveaux. Les besoins initiaux sont réutilisés, entièrement ou en partie, pour être spécifiés et mis à jour en vue d'obtenir une description du nouveau système correspondant aux nouvelles contraintes. D'autres peuvent aussi venir s'ajouter, il n'est en effet pas rare que les contraintes de départ ne soient plus en concordance avec l'évolution du système, et il est alors nécessaire d'ajouter de nouvelles contraintes à l'ensemble des précédentes pour mettre le système à jour.

Il est donc nécessaire d'avoir une bonne organisation de l'information transmise entre les différentes étapes de développement du système. Parce que les personnes impliquées dans ce développement seront probablement différentes au cours de celui-ci et dans ses modifications, l'information le concernant ne doit pas demeurer la propriété des développeurs mais doit être intégrée à la spécification du système pour l'accompagner dans toutes les étapes de transformation.

Un système est rarement construit en une seule itération, c'est pourquoi il est important que les documents le concernant contiennent aussi la description des raisonnements qui ont permis d'aboutir à sa définition. Les personnes qui ont travaillé précédemment pourraient avoir développé d'autres versions moins prioritaires ou abandonnées, celles qui ont documenté les contraintes initiales pourraient être indisponibles,...

Si les raisonnements ne sont pas conservés avec les requirements originaux, certains problèmes peuvent surgir :

1. Le code doit être utilisé pour découvrir la philosophie qui se cache derrière les décisions prises, ce qui risque de prendre beaucoup de temps et d'occasionner un certain nombre d'erreurs.

2. De nouvelles contraintes sont ajoutées pour une nouvelle version du système, et elles sont en contradiction avec les raisonnements utilisés pour le système existant. Si les décisions ne sont pas enregistrées, ces conflits peuvent passer inaperçus.

Il est donc important d'avoir des techniques qui permettent de conserver la trace des raisonnements, des informations concernant les décisions prises et des différentes alternatives proposées dans les discussions, avec les arguments supportant ou objectant à ces positions.

En réalité, certains de ces documents s'avèrent parfois plus utiles pour la maintenance que le code final dans lequel les effets d'une décision prise relativement tôt peuvent devenir difficiles à repérer. Sans ces enregistrements, le responsable de la maintenance peut répéter certaines erreurs qui ont été faites dans le développement original. Le fait de ne posséder que le code final peut aussi l'empêcher de retrouver et défaire certaines décisions qui ne lui paraissent pas nécessaires.

Il est aussi très intéressant, par exemple lors d'une remise en cause d'une partie de développement ("backtracking"), de voir comment un produit a été construit et de pouvoir remonter à différents points du développement où l'information qu'il contient a été introduite.

Un autre avantage de l'enregistrement de la documentation sur l'exécution d'un processus réside dans le fait qu'il permet une réutilisation de certaines parties d'un développement. Le concepteur peut vouloir mettre en oeuvre un processus similaire à celui qui a déjà été fait, et en "récupérer" certaines parties. Il lui suffira de parcourir la documentation, de vérifier si les différentes décisions prises correspondent bien à son objectif, et modifier uniquement celles qui ne l'intéressent pas.

Enfin, l'historique d'un processus et de ses différents composants réellement utilisés peut aussi servir de feed-back à l'ingénieur méthode. L'observation des différents enregistrements concernant des processus peut pousser celui-ci à améliorer la manière de travailler prescrite par une méthodologie particulière.

On voit donc que le rôle de l'enregistrement de l'exécution réelle d'un processus est d'une utilité multiple qui concerne à la fois la planification du processus, son contrôle et son amélioration. A partir de cette information historique, on peut extraire et organiser un certain nombre de connaissances dans le but d'aider le développeur.

C'est dans toutes ces raisons qui viennent d'être exposées que l'existence de ce niveau des instances trouve sa justification.

3.2.2.2 Modèle de représentation

La plupart des activités d'analyse des requirements, de spécification, conception, transformations, ... sont fortement dépendantes du travail intellectuel des développeurs, et il semble difficile de pouvoir définir clairement la manière exacte qui leur permet de faire évoluer les spécifications initiales en produits finaux qui leur correspondent. Cependant, même s'il n'est pas possible de représenter de façon précise tous les raisonnements utilisés, nous pouvons quand même identifier certaines étapes et mécanismes empruntés par le concepteur dans son processus de développement.

Le concepteur part d'un produit initial constitué de l'ensemble des requirements, spécifications, besoins de l'utilisateur, ... Son objectif est de construire d'autres produits qui sont la traduction de ces spécifications dans un formalisme bien déterminé. Pour cela, il utilise différents processus qui permettent de faire évoluer les produits. L'exécution d'un processus, ou d'une série de processus, peut être la conséquence d'une hypothèse avancée par le concepteur et qui constitue un essai dans une voie de développement particulier. Elle peut aussi faire suite à une décision qu'il a prise et qui lui dicte la manière d'agir pour arriver à son but.

La mise en oeuvre d'une stratégie peut l'obliger à définir différentes étapes qui permettent de la réaliser. Elle sera concrétisée par l'application de transformations correspondantes sur les produits.

A tout moment du développement, le concepteur peut proposer des hypothèses de développement, ou prendre des décisions sur la manière de continuer. La seule contrainte à ce niveau est que la proposition de plusieurs alternatives pour résoudre un même problème nécessitera par la suite la prise d'une décision pour n'en choisir qu'une seule.

Pour pouvoir organiser toutes ces informations de manière efficace, il est nécessaire d'utiliser un modèle et des techniques de représentation qui permettent à toutes les personnes impliquées dans le développement du système de comprendre ce que les descriptions des contraintes, dont les imperfections sont inhérentes au langage naturel, signifient réellement.

Ce modèle peut alors être aussi utilisé pour les futures itérations sur le système, encourageant et facilitant l'enregistrement des informations et diminuant le temps de spécification.

Le modèle de l'historique d'un processus a tout d'abord comme objectif de permettre d'enregistrer et maintenir de l'information concernant les différents produits mis en oeuvre et les actions qui ont permis de les obtenir. Ces enregistrements contiennent de l'information concernant l'histoire des processus réellement exécutés et des produits développés. Ensuite il permet aussi de garder la trace des décisions et des raisonnements suivis au cours de ce développement. Ces enregistrements reprennent la description des décisions en tant que telles, mais aident aussi à les documenter en contenant les différentes alternatives qui ont été proposées, ainsi que les arguments supportant ou rejetant certains points de vue.

Il est clair que l'enregistrement de tous les produits, même dans les différents états successifs de leur développement, ne constitue pas une trace suffisante. La documentation concernant un processus particulier est constituée de plusieurs parties.

Tout d'abord une documentation sur les résultats du processus, c'est-à-dire les différents produits qui ont été élaborés, dans les différents états par lesquels ils sont passés avant d'être complètement terminés.

Ensuite une documentation qui porte sur le processus en tant que tel avec les raisonnements c'est-à-dire les différentes hypothèses et décisions prises au cours de celui-ci, et les actions concrètes auxquelles elles ont donné lieu.

Le fait de faire une différence entre l'enregistrement des résultats fournis, et le raisonnement qui a permis de les obtenir possède un double avantage. Le concepteur ne doit pas rechercher dans les produits des commentaires ou des explications qui ne s'y trouvent peut-être pas, et les considérations qui ont un effet sur plusieurs produits ne sont enregistrées qu'à une seule place. Même s'il est utile de présenter les résultats avec des notes concernant leur construction, ces deux éléments sont cependant séparés au niveau conceptuel. Les produits enregistrent les résultats de la conception, le raisonnement les décisions prises pour produire la conception.

L'approche utilisée représente le concept central d'instance de processus qui est mis en relation avec les différents concepts supplémentaires nécessaires à la représentation complète d'un processus. Les produits sont les objets sur lesquels portent les processus, et les raisonnements utilisés sont représentés grâce aux concepts d'hypothèse et de décision.

Le modèle introduit ici permet de représenter le raisonnement qui a été utilisé dans une conception, et de le mettre en relation avec l'élaboration des différents produits qui est elle-même soumise à l'observation d'une méthode spécifique. Il utilise des concepts modélisant des objets qui sont des phénomènes observables réels.

La partie concernant l'élaboration des produits en tant que telle est représentée par les concepts d'instances de processus et de produits, qui permettent de spécifier grâce à quels mécanismes le concepteur a concrétisé ses décisions et a pu obtenir ses résultats.

La représentation du raisonnement est basée sur les concepts d'hypothèse et de décision qui, de manière générale, montrent les différents éléments introduits par le concepteur dans sa réflexion pour l'avancement de son développement.

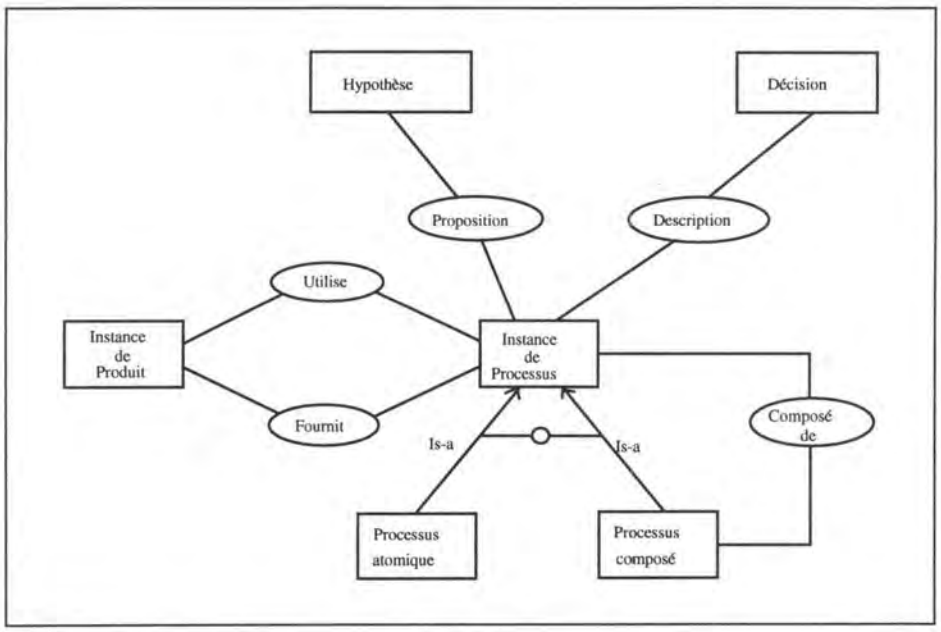


Fig. 3.4 Le modèle de représentation d'un historique

- Une *instance de processus* représente soit la réalisation d'un processus défini par une méthode, elle a alors comme objectif la transformation de produits dans le but de faire évoluer ceux-ci vers leur spécification finale (en fonction des hypothèses, décisions ou contraintes), soit la prise d'une décision à un moment du développement. Dans ce dernier cas, elle ne représente pas la transformation d'un produit mais la définition d'une conduite à suivre pour la suite du développement.

- Un *instance de produit* est la description d'un objet sur lequel agissent les différentes actions du développeur. Elles permettent de définir l'état courant du développement, et peuvent aussi être analysées pour être comparées aux préconditions d'un processus et définir son "activabilité".

- Une *hypothèse* est introduite par le développeur et permet à celui-ci de présenter différentes possibilités de développement qui ont pour but de faire évoluer son état courant.

- Une *décision* permet de décrire un choix qu'a fait le développeur relativement à un contexte précis. Elle a pour but d'apporter des précisions supplémentaires à celles déjà contenues dans la description de l'instance du processus par lequel elle a été introduite.

A tout moment de son développement, le concepteur se trouve dans un état particulier qui constitue une étape de la transformation des spécifications initiales en produits finaux. A partir de ce contexte, il peut émettre différentes hypothèses portant sur la manière de continuer son développement et qui trouvent leur fondement dans la perception qu'il a du système à construire et des résultats qu'il souhaite obtenir. Chacune de ces alternatives lui permet de partir du contexte courant et de le développer d'une manière différente.

Il peut aussi appliquer des transformations directement sur les produits. Il concrétise alors les objectifs qu'il avait auparavant définis dans une hypothèse ou une décision en les répercutant sur les produits.

A chaque étape du développement, le concepteur peut aussi prendre certaines décisions quant à la façon de le poursuivre. Ces décisions peuvent porter sur les différents éléments définis dans le modèle comme les hypothèses ou les actions. Elles auront par exemple comme objet l'abandon d'alternatives proposées dans le contexte, ou l'utilisation d'un processus ou d'une suite de processus particuliers à appliquer sur les produits.

Ces décisions sont des moments relativement importants dans l'exécution du processus car elles permettent d'avoir des points de repère en ce qui concerne les étapes du développement, et constituent, avec les hypothèses, la définition de son architecture. Nous verrons dans l'étude de cas au chapitre 4 que celui-ci peut être représenté par un arbre dont chaque branche constitue un développement particulier portant sur l'ensemble des produits.

3.2.2.3 Description des éléments du modèle

a. Instance de processus

Le concept d'instance de processus permet de représenter une opération exécutée par le concepteur pour faire évoluer l'ensemble des produits qui définissent l'état courant du développement vers le résultat attendu correspondant à la spécification initiale.

Une instance de processus peut correspondre à plusieurs types. Tout d'abord, elle peut constituer une instance d'un processus défini par une méthode, et la concrétisation ou la mise en oeuvre réelle par modification des produits d'une hypothèse proposée ou d'une décision prise dans un état du développement. Elle est alors associée aux différents produits qu'elle met en oeuvre, et permet de faire évoluer ceux-ci de façon concordante à la stratégie choisie par le concepteur. Quand elle est exécutée, des instances de produits sont créées, modifiées et consommées. Elle a donc comme effet de modifier l'ensemble des produits utilisés dans un développement et de le faire évoluer vers ce que le concepteur s'est défini comme objectif.

Une instance de processus peut aussi représenter le fait que le concepteur a pris une décision au cours du développement. Elle ne correspond donc pas à l'exécution d'un processus défini par la méthode, mais plutôt à celle d'un processus standard fourni par l'outil. Elle a pour but de proposer une stratégie pour orienter la suite du développement, et qui sera répercutée sur l'ensemble des processus exécutés par la suite.

L'exécution d'un processus peut donc représenter à la fois la proposition d'une stratégie de développement par prise de décision, et aussi la transformation de l'ensemble des produits, dictée par la méthode, et qui constitue la concrétisation de ces décisions permettant de faire évoluer l'état des produits courants en fonction des arguments et objectifs de celles-ci.

Une instance de processus est complètement spécifiée grâce aux attributs suivants :

- Un *nom* qui identifie l'action parmi toutes les autres qui sont exécutées au cours du développement.

- Un *type* qui permet de spécifier à quelle catégorie appartient le processus exécuté et selon le cas de quel processus défini dans la méthode il constitue une instance.

- Le *nom* d'un processus que compose celui-ci. De la même manière qu'un processus peut être décomposé en plusieurs sous-processus dans la définition de la méthode, l'exécution de celui-ci peut être réalisée par l'exécution de plusieurs autres. Il est donc nécessaire de rattacher ceux-ci au premier pour justifier en partie leur exécution. Cet attribut est aussi utile pour le "backtracking" ou le "replay", la réutilisation ou la remise en cause d'un processus impliquant évidemment la réutilisation ou la remise en cause de tous ceux qui le composent.

- Une *liste des produits* que l'action a utilisés en *entrée*. Ceux-ci font normalement partie de l'ensemble des produits qui définissaient l'état courant du développement dans lequel l'action a été exécutée.

- Une *liste des produits* que l'action a fournis comme *résultat*. Ils définissent un nouvel état à partir duquel d'autres décisions ou hypothèses peuvent être introduites, et d'autres processus exécutés pour continuer le développement.

b. Instance de produit

Les produits sont des objets réels utilisés par les processus exécutés au cours du développement. Si l'on considère le cycle de vie de tout le système, un certain nombre des objets produits pendant sa construction deviennent extrêmement utiles, et la définition propre d'un produit utilisé dans un processus est donc importante. Pour être considéré comme un produit dans le développement, un objet doit :

1. Exister dans le monde réel et pas simplement dans le modèle ou le processus.
2. Etre identifiable et identifié par son nom.
3. Etre transformé par le processus à travers un ensemble défini d'états.

Un produit représente n'importe quel objet sur lequel peut porter un processus, et qui a une durée de vie déterminée au cours du développement. Il s'agit de choses qui sont introduites dans le processus et qui persistent un certain temps. Des exemples de ces objets sont les requirements, des programmes, des documentations, des schémas,... L'état des produits en cours de construction définit à tout moment l'état d'avancement du travail dans lequel l'utilisateur peut proposer différentes hypothèses pour continuer son développement. Le produit peut donc être indirectement considéré comme un objet de décision, puisque c'est par rapport à ce qu'il a déjà défini et en fonction de ce

qu'il doit obtenir que le concepteur prendra des décisions concernant l'orientation de son travail. Le but du concepteur est de définir le ou les produits finaux correspondant aux requirements initiaux.

Etant donné que la logique suivie est de séparer les produits-mêmes utilisés dans le processus des différents éléments et mécanismes mis en oeuvre pour les construire, leur spécification ne reprend pas ces éléments.

Un produit utilisé dans un développement est complètement spécifié s'il possède une valeur pour les attributs suivants :

- Un *nom* de produit qui identifie celui-ci parmi l'ensemble de tous les produits utilisés dans le développement.

- Une *description*, qui spécifie de manière informelle à quoi correspond le produit, quelles sont ses caractéristiques, ... par rapport au problème.

- Un *type de produit*, qui permet de spécifier à quel type défini dans la méthode il appartient (de quel objet défini dans la méthode il constitue une instance).

- Une *spécification* plus formelle ou une description technique du produit qui doit être interprétée.

On peut aussi ajouter que c'est principalement en fonction des produits et de ce qu'il veut obtenir que le concepteur raisonne et prend des décisions. Nous pouvons donc encore considérer les produits comme des objets principaux de raisonnement à partir desquels il introduit des hypothèses ou prend des décisions.

L'avantage de représenter les produits comme des objets indépendants est que chacun possède sa propre définition et sa propre séquence de transformations (les différents états dans lesquels il passe au cours de son développement). Même si elles dépendent parfois de l'état des autres produits, les modifications sont le résultat de causes bien définies, et la séquence des différents états par lesquels passe le produit est contenue dans la propre évolution des définitions de celui-ci. Chaque chaîne de transformation d'un produit peut donc être considérée comme (relativement) indépendante et peut être étendue ou raccourcie sans perte de "consistance" ou logique.

Selon la granularité désirée, on peut alors "ajuster" cette chaîne et ne considérer que les aspects du processus impliqués dans la transition entre les états ainsi introduits.

c. Décision

Le concept de décision permet de représenter un choix opéré par le développeur par rapport à l'état courant du développement pour définir l'évolution de son travail. Elle peut porter sur les différentes hypothèses qu'il a émises, et qu'il décide de prendre en compte ou d'abandonner, ou peut encore constituer la définition d'une ligne de conduite qu'il adoptera dans la suite du développement.

Un objet représentant une décision peut être décrit grâce aux attributs suivants :

- Un *nom* qui identifie la décision prise parmi toutes celles appartenant au développement.

- Un *objet* qui spécifie sur quoi porte la décision, à quoi elle fait référence, ...

- Une *description* qui explique le choix fait ou en quoi consiste la décision prise, et quels sont ses objectifs et ses conséquences éventuelles.

- Une *justification* qui permet de donner les arguments supportant la décision prise.

Une décision représente une intention de transformation des produits selon les critères ou les contraintes définies, et constitue un concept important pour le suivi d'une stratégie. Elle est prise dans un contexte en fonction des différentes hypothèses qui ont été émises, et en tenant compte aussi de la manière et des résultats auxquels le concepteur désire aboutir.

Une décision permet tout d'abord au concepteur d'opérer un choix parmi les différentes alternatives proposées. La prise d'une telle décision a comme objectif de choisir parmi un ensemble d'hypothèses qui correspondent à la même intention d'action, c'est-à-dire à utiliser la bonne stratégie. Les alternatives proposées traitent toutes d'un même objectif mais proposent des manières différentes de résolution. Chacune correspond donc à un chemin différent de décision, et le fait d'opter pour l'une d'entre elles implique donc de choisir pour la stratégie qui lui est associée. Ce type de décision est appelé stratégique dans [Rol93].

Par exemple, lorsqu'il commencera l'élaboration du schéma conceptuel de la base de données, l'ingénieur décidera de définir un seul schéma dès le départ, contenant toutes les informations, plutôt que de commencer à en définir plusieurs qu'il devra intégrer par la suite, et qu'il trouve plus compliqué.

Le choix d'une telle alternative ne provoque pas directement une transformation de produit. Elle ne fait qu'introduire un contexte plus restreint en ce sens qu'il possède des précisions supplémentaires sur la façon de procéder, il correspond à une description ou une étape nécessaire pour obtenir les produits désirés, et sera lui-même éventuellement une nouvelle source d'hypothèses ou sujet à des précisions.

Une décision peut aussi être prise pour sélectionner un ensemble d'actions possibles qui ont pour but l'application de transformations sur les produits. Sa conséquence est l'exécution des actions choisies. Elle se rattache aux concepts des différents modèles supportés par la méthodologie utilisée, et est dérivée de la manière de travailler prescrite par la méthodologie dont tiennent compte les différentes actions sur lesquelles elle peut porter. Elle concerne directement le produit en cours de construction et a pour but de faire évoluer sa spécification.

Ce type de décision, qualifiée de tactique dans [Rol93], peut être considérée comme la concrétisation des décisions stratégiques par l'application de transformations sur les produits. Elle permet en effet de répercuter directement sur les produits les hypothèses et les choix opérés lors de la sélection d'une stratégie.

Par exemple, dans le cadre de la définition du concept de livre dans une bibliothèque, la conséquence d'une décision prise par le concepteur peut être de définir les différentes étapes permettant de construire la partie de schéma correspondante. Il construira d'abord le concept de livre, puis celui d'auteur, et créera une association entre les deux. Ensuite, il ajoutera le concept d'emprunteur qu'il reliera aussi à livre.

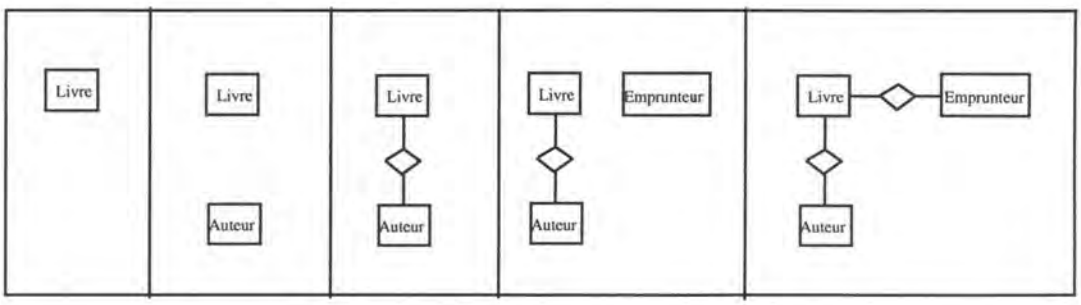


Fig. 3.5 Exemple de séquence d'actions dictée par une décision.

Ces transformations du produit et l'ordre dans lequel elles sont effectuées sont dictés par la décision prise.

L'orientation d'un processus de conception est déterminée par un ensemble de décisions qui sont basées à la fois sur la vision propre qu'a le concepteur du système à construire, l'idée qu'il se fait de la manière dont il va procéder, et sur certaines contraintes imposées par la méthodologie suivie dans le cadre de ce développement.

d. Hypothèse

Le concept d'hypothèse permet de représenter une proposition que fait le concepteur à un moment du développement. Elle consiste à faire certaines considérations sur la manière de conduire le processus, sur les produits à construire, ... et elle peut constituer une alternative à d'autres hypothèses faites pour résoudre un même problème (dans ce cas, le concepteur ne pourra en choisir qu'une parmi celles proposées). Les hypothèses représentent en quelque sorte des "essais" que fait le concepteur. A chaque étape de son développement, il peut avoir la possibilité de définir différentes voies qui lui permettront d'obtenir ses résultats. Chacune de ces voies est introduite par une hypothèse qui est formulée en fonction des contraintes, des spécifications ou encore de la perception que le développeur a du système à construire.

Par exemple, le concepteur qui veut développer la base de données d'une bibliothèque pourra envisager deux façons de travailler. D'un côté, il choisira de commencer à développer l'aspect emprunt de la bibliothèque en s'occupant d'abord des abonnés, des mécanismes de prêt, ... D'un autre côté, il estimera qu'il est préférable de commencer par la construction de la bibliothèque elle-même avec les concepts de livres, auteurs, ...

Ce sont deux possibilités qui peuvent être envisagées et qui constituent chacune une stratégie différente de développement. Le concepteur sera obligé par la suite de faire un choix entre les deux.

Un objet hypothèse est défini par les attributs suivants :

- Un *nom*, qui permet de l'identifier parmi l'ensemble de toutes les autres hypothèses proposées dans le développement.

- Une *description* qui permet d'expliquer en quoi consiste l'hypothèse proposée et ce qu'elle implique sur le développement.

- Une *justification* qui permet de donner des arguments supportant la validité et la pertinence d'une telle hypothèse et de préciser quels sont ses avantages.

Une hypothèse est proposée par rapport à un état courant. Elle ne constitue pas une action de transformation mais introduit une stratégie de développement. Elle constitue simplement la création d'un noeud dans l'histoire du développement qui permet au développeur l'exploitation de plusieurs chemins possibles d'évolution.

Chaque alternative peut être développée. Le choix de l'une d'entre elles n'empêche pas le concepteur de revenir à l'état courant du développement dans lequel elles ont été proposées et de commencer à en développer une autre. Mais si les hypothèses faites à ce moment portent sur un même problème et qu'elles ont chacune pour objectif d'obtenir un résultat qui résout celui-ci, il arrivera un moment où le concepteur sera obligé de faire un choix et d'en écarter certaines pour ne garder que celle qui le satisfait le plus.

e. Résumé

Le tableau ci-dessous reprend les différents objets nécessaires à la spécification de l'historique d'un développement, avec leurs composants principaux.

Objet	Attributs	Description
Instance de processus	nom type compose input output	Identification du processus. Nom du processus méthodologique dont celui-ci est une instance. Nom du processus dont celui-ci est un composant. Produits utilisés par le processus. Produits fournis par le processus.
Instance de produit	nom description type spécification	Nom du produit. Description informelle du contenu du produit. Type du produit. Spécification formelle du produit.
Décision	nom objet description justification	Identification de la décision. Objet de la décision. Explication du contenu de la décision et de ses conséquences. Arguments qui supportent la décision.
Hypothèse	nom description arguments	Identification de l'hypothèse. Description informelle du contenu de l'hypothèse. Justification de la pertinence de la proposition.

3.3 Langage de spécification de méthode

3.3.1 Introduction

Cette partie du travail a pour objet de décrire la syntaxe et la sémantique d'un langage de description de méthode. Nous utilisons le terme "langage de description", et non pas "langage de programmation" de méthode pour mettre en évidence le fait que la spécification d'une méthode avec ce langage ne sera pas traduite en un texte en code objet exécutable par une machine. Il doit plutôt être compris comme la description d'une structure et du comportement d'un processus et de ses produits, qui pourra être analysée ou interprétée, par exemple par un moteur de méthode dans un atelier logiciel.

La définition de ce langage est évidemment basée sur ce que nous avons vu dans les parties précédentes concernant la modélisation d'un processus, et plus particulièrement le modèle de représentation d'une méthode. Son utilisation permet à l'ingénieur de définir l'ensemble des différentes activités et produits relatifs à la méthode et qui serviront de modèle à une exécution réelle.

Les éléments qui le constituent représentent un ensemble nécessaire mais suffisant, et qui permet une bonne compréhension des mécanismes utilisés pour la spécification d'une méthode. Il pourra également être complété ou adapté par la suite pour permettre son utilisation dans des cas plus particuliers, ou pour permettre une interprétation plus efficace par un agent quelconque.

3.3.2 Notes sur la syntaxe

3.3.2.1 Syntaxe BNF

Les règles de grammaire peuvent être décrites avec des grammaires BNF. L'intérêt d'utiliser une telle grammaire est de pouvoir donner une spécification précise et facile à comprendre de la syntaxe d'un langage de programmation, et de pouvoir facilement adapter une grammaire au fur et à mesure de l'évolution du langage.

Une grammaire BNF est définie comme un ensemble constitué de symboles terminaux, de classes syntaxiques, d'une variable distinguée et de productions où :

- Les symboles terminaux sont les symboles de base à partir desquels les chaînes sont formées. Ce sont tous les symboles qui peuvent intervenir dans la représentation du texte du programme; ils sont considérés syntaxiquement comme atomiques.

- Les classes syntaxiques désignent une concaténation d'autres classes syntaxiques et de symboles terminaux. Parmi celles-ci, une classe syntaxique est appelée la "variable distinguée".

- La variable distinguée est la classe syntaxique associée au texte complet du programme.

- Les productions d'une grammaire sont des combinaisons de symboles et de classes syntaxiques pour former des chaînes. Une production a la forme :

"Classe syntaxique ::= chaîne de classes syntaxiques et/ou symboles terminaux"

Nous utiliserons la syntaxe BNF avec les conventions suivantes :

1. La barre verticale (" | ") sépare des alternatives.
2. Les accolades ("{" et "}") indiquent des composants qui peuvent être répétés 0, 1 ou plusieurs fois. Les crochets ("[" et "]") indiquent des composants facultatifs.
3. Les symboles terminaux sont encadrés par des guillemets (" "), ceux-ci n'appartiennent pas au symbole.
4. Aucune distinction n'est faite entre les lettres majuscules et minuscules.
5. Les commentaires commencent par un double tiret (--). Tout le texte entre ces caractères et la fin de la ligne sont ignorés.
6. Les identificateurs sont des séquences de lettres, chiffres et underscores ("_"). Ils doivent commencer avec une lettre et se terminer par une lettre ou un chiffre. Il ne peut y avoir deux underscores consécutifs dans un identificateur.

3.3.2.2 Mots-clés

Le langage utilise évidemment un certain nombre de mots-clés, qui sont des unités syntaxiques ayant une sémantique bien définie dans le texte de spécification. Ces mots sont mis en évidence en étant écrits en majuscule et en gras dans le texte. La liste de ces mots est précisée avec leur sémantique dans les parties concernant la spécification des processus et des produits.

3.3.2.3 Expressions

Les expressions booléennes sont des formules utilisées par le langage, notamment pour spécifier les propriétés des produits, ou encore certaines conditions d'exécution des processus.

```
Formula  ::=  Disjunction

Disjunction  ::=  Disjunction  "OR"  Conjunction  |
                  Conjunction

Conjunction  ::=  Conjunction  "AND"  Primary  |  Primary
```

```
Primary  ::=  "NOT"  Primary  |  AtomicFormula


AtomicFormula  ::=  PredicateExpr          |
                    Relation               |
                    "("  Formula  ")"


Relation  ::=  Value  "="  Value    |          -- Egal
               Value  "<>"  Value   |          -- Différent
               Value  "<"  Value    |          -- Plus petit
               Value  "<="  Value   |          -- Plus petit
                                                --  ou égal
               Value  ">"  Value    |          -- Plus grand
               Value  ">="  Value   |          -- Plus grand
                                                --  ou égal
```

3.3.2.4 Les commentaires

La spécification des éléments de la méthode peut contenir une description informelle de leur signification. Celle-ci est incluse dans le texte sous forme de commentaires qui sont introduits à partir des caractères "--" jusqu'à la fin de la ligne.

3.3.3 Spécification des éléments de la méthode

3.3.3.1 Spécification de la méthode

La spécification d'une méthode est divisée en deux parties importantes, la description des produits et la description des processus.

```
MethodSpec  ::=  ProductDescriptionList
                                  -- Description des produits

                 ProcessDescriptionList
                                  -- Description des processus
```

La description des processus est faite après celle des produits, puisque nous avons vu qu'ils utilisent les caractéristiques de ceux-ci comme pré- et post-conditions.

Chacune de ces parties est traitée individuellement dans les deux sections qui suivent.

3.3.3.2 Syntaxe et sémantique de la spécification des produits

a. Mots-clés de la spécification des produits

PRODUCT	Début de la spécification d'un produit
IS	Suivi du produit dont celui-ci est dérivé
WITH	Suivi des concepts utilisés par le produit
AS	Permet de renommer les concepts dérivés
PROPERTIES	Début de la spécification des propriétés du produit
END-PRODUCT	Fin de la spécification du produit

b. Spécification des produits

La déclaration des produits est une unité qui définit la structure et les propriétés des objets utilisés et fournis par les processus. Elle est constituée d'une liste dont chaque élément est la déclaration d'un produit particulier.

```
ProductDescriptionList  ::=  ProductDescription
                                    { ProductDescription }
```

La spécification d'un produit consiste en deux parties principales :

- La structure du produit qui permet de décrire ses composants. Cette définition peut utiliser des types de données connus de la méthode, ou d'autres produits définis précédemment. C'est dans la spécification de cette structure qu'est utilisé le mécanisme de dérivation du produit à partir d'un autre.

- Les propriétés qui représentent des invariants sur le produit. Celles-ci sont spécifiées de manière relativement informelle.

Syntaxe

```
ProductDeclaration  ::=  "PRODUCT"  ProductName
                                    "%"  InformalDescription
                                    "IS"  StructureDefinition
                                    "WITH"  ConceptList
                                    "PROPERTIES"  PropList
                                "END-PRODUCT"  ProductName

StructureDefinition  ::=  ProductName  |
                                    -- Produit dont il est déduit

                                DataType
                                    -- Type de donnée connu par
                                    -- la méthode
```

```
ConceptList  ::=  ConceptDeclaration
                      { ConceptDeclaration }  |

                  ["all"]            -- Récupération de tous
                                        -- les concepts
                  { ConceptDeclaration }


ConceptDeclaration  ::=  ConceptName
                                     -- Concept utilisé

                             ["AS" ConceptName ]
                                     -- Renommage des concepts

ConceptName  ::=  Identifier

PropList  ::=  PropDecl
                  { PropDecl }


PropDecl  ::=  PropName                    -- Nom de la propriété
                  "%"  InformalDescription
                                           -- Description de sa
                                           -- signification

PropName  ::=  Identifier
```

3.3.3.3 Syntaxe et sémantique de la spécification des processus

La spécification des processus consiste à définir une liste des différents processus mis à la disposition du concepteur par la méthode.

```
ProcessDescriptionList  ::=  ProcessDescription
                                  { ProcessDescription }
```

La spécification d'un processus au sein d'une méthode particulière consiste à décrire comment celui-ci transforme un ensemble de produits utilisés comme arguments en un autre ensemble de produits fournis comme résultat. Le module de spécification d'un processus contient principalement trois parties qui sont la description des produits input, la description des produits output, et la description de la partie procédurale.

Chaque processus fait partie d'un réseau dans lequel il est connecté à d'autres processus. Cette coordination entre les différentes activités est contenue dans la propre définition de celles-ci et plus particulièrement dans la partie procédurale.

Dans une première approche, nous pouvons déjà introduire certaines restrictions sur ce réseau de processus :

1. Il doit être cohérent, et il ne peut pas y avoir de "deadlock".

2. Chaque objet utilisé comme input doit exister avant d'être utilisé par une activité. Il a donc dû être créé auparavant (par une autre activité ou fourni tel quel par le concepteur).

3. Il ne doit normalement pas y avoir d'actions qui ne pourront jamais être utilisées.

4. Il ne peut pas y avoir d'actions concurrentes qui produisent le même résultat.

Ce qui suit est une spécification plus ou moins détaillée de la syntaxe de la déclaration des processus. La sémantique est incluse sous forme de commentaires, et est complétée par un texte au cas où ceux-ci ne suffiraient pas.

a. Mots-clés de la spécification des processus

PROCESS	Début de la spécification d'un processus
SUMMARY	Suivi de la description informelle du processus
INPUT	Suivi de la liste des produits utilisés comme arguments
OUTPUT	Suivi de la liste des produits fournis comme résultat
PROCEDURE	Début de la spécification de la partie procédurale
LOOP	Début d'une instruction de boucle
END-LOOP	Fin de l'instruction de boucle
EXIT LOOP	Instruction permettant de sortir d'une boucle
IF	Suivi d'une condition
THEN	Suivi d'opérations pouvant être exécutées si la condition est vérifiée
ALTERNATIVE	Début d'une liste des différentes possibilités d'opérations à réaliser
END-ALTERNATIVE	Fin de la liste des différentes possibilités d'opérations à réaliser
END-PROCESS	Fin de la spécification d'un processus

Rappelons qu'en plus de ces mots-clés, toutes les unités alphanumériques commençant par une lettre et se terminant par une lettre ou un chiffre sont reconnus comme des identificateurs.

b. Spécification des processus

La spécification d'un processus est comprise entre les deux mots-clés "PROCESS" et "END-PROCESS". Elle consiste à énumérer la liste des produits utilisés comme arguments, la liste des produits fournis comme résultats, et la description de la procédure qui décrit les différentes étapes d'exécution du processus.

La définition d'une liste de produits peut être constituée de plusieurs sous-listes séparées par des 'OU'. Cela signifie que le processus met en oeuvre (utilise ou fournit) tous les produits de l'une, ou de l'autre sous-liste.

Syntaxe

```
ProcessDescription  ::=  "PROCESS"  ProcessName
                              "SUMMARY"  InformalDescription
                              "INPUT"  ":"  ProductList
                              "OUTPUT"  ":"  ProductList
                              "PROCEDURE"
                                    ProcedureDescription
                         "END-PROCESS"  ProcessName


ProductList  ::=  {ProductName  ["*"] }          |
                  ProductList  "OR"  ProductList
                                    -- '*' signifie que ce produit
                                    -- peut avoir plusieurs
                                    -- instances utilisées par le
                                    -- processus.

ProductName  ::=  Identificateur
```

c. Spécification de la partie procédurale

La description procédurale du processus consiste à spécifier l'algorithme d'exécution de celui-ci, qui permet de transformer les arguments en résultat. C'est aussi dans cette partie qu'est spécifiée la coordination et les connexions entre les différents processus de la méthode.

Le langage utilisé facilite la compréhension du processus, il supporte la décomposition hiérarchique telle que nous l'avons décrite précédemment. Il fournit également les mécanismes d'exécution et utilise les structures de contrôle de base comme la séquence, le contrôle conditionnel, et le contrôle itératif.

- La séquence est spécifiée par un " ; " séparant chacune des instructions.
- Le contrôle conditionnel est réalisé grâce au test "IF", permettant d'introduire des conditions à l'exécution d'une instruction.
- Le contrôle itératif est réalisé grâce à la notion de boucle avec "LOOP".

Nous ajoutons encore un mécanisme supplémentaire qui permet de spécifier un certain nombre d'instructions qui doivent être réalisées, mais sans imposer d'ordre sur cette exécution. N'oublions pas que la méthode sert de modèle à une exécution réelle et que c'est l'utilisateur qui déclenche l'exécution des différents processus. Ce nouveau contrôle permet d'exprimer le fait que la méthode laisse le choix à l'utilisateur de l'ordre dans lequel il souhaite exécuter les processus. Il est spécifié grâce au caractère " | " qui séparera les instructions.

Enfin, l'instruction "ALTERNATIVES" permet de spécifier un ensemble d'instructions parmi lesquelles l'utilisateur doit faire un choix unique. Dans les processus qui apparaissent dans cette clause, un seul peut être exécuté .

Syntaxe

```
ProcedureDescription  ::=  Op-Unit

Op-Unit  ::=  ProcessName      |     -- Nom du processus à exécuter
              LoopUnit         |     -- Instruction itérative
              CondUnit         |     -- Instruction conditionnelle
              SeqUnit          |     -- Séquence d'instruction
              SetUnit          |     -- Instructions exécutables
                                       -- dans un ordre quelconque
          AltUnit                    -- Alternatives


ProcessName  ::=  Identifier  |
                  "EXIT LOOP"

LoopUnit  ::=  "LOOP"
                  Op-Unit
               "END-LOOP"


CondUnit  ::=  "IF"  Formula  "THEN"  Op-Unit


SeqUnit  ::=  Op-Unit  ";"
                  Op-Unit
                  { ";"  Op-Unit }


AltUnit  ::=  "ALTERNATIVE"
                  Op-Unit  ["IF"  Formula ]
                  Op-Unit  ["IF"  Formula ]
                  { Op-Unit  ["IF"  Formula ] }

              "END ALTERNATIVE"


SetUnit  ::=  Op-Unit  "|"
                  Op-Unit
                  { "|"  Op-Unit }
```

Conclusion

L'objet de ce chapitre était de décrire un modèle de processus qui prenne en compte un certain nombre d'aspects nécessaires à sa représentation complète.

Nous avons utilisé une approche qui, bien qu'étant basée sur les activités et les produits du processus, prenait également en charge l'aspect comportemental de celui-ci. Le modèle constitue en quelque sorte une "mise en commun" de tous les concepts ciblés dans les différents modèles rencontrés dans la littérature, et conserve donc les avantages de ceux-ci, en évitant les inconvénients dus à l'insuffisance et la limitation du nombre d'éléments utilisés pour représenter un développement.

Il est composé de deux parties destinées chacune à une représentation d'un niveau particulier. Le premier fournit un moyen de représentation des informations permettant la définition d'une méthode, ce qui permet à l'ingénieur de prescrire un "plan de travail", ce qui doit se passer, quand et comment. Le deuxième niveau présente une architecture des différents éléments intervenant au cours d'un développement concret et qui constituent l'historique de celui-ci.

Cette approche permet de représenter un développement aussi bien en termes d'événements qu'en termes d'états de produits construits. Sa relative simplicité et la séparation entre les concepts facilitent la compréhension des décisions prises durant la conception, avec les arguments qui les supportent et les résultats qu'elles ont engendrés.

La représentation relativement formelle du modèle, ainsi que le langage de description de méthode que nous avons décrit permettent de supporter des visions multiples et complémentaires d'un processus (fonctionnelle, comportementale, conceptuelle, l'implémentation, ...). Ils donnent une description de la même réalité selon différents points de vue, et supportent des niveaux d'abstraction multiples, ce qui permet d'avoir une image assez générale du processus aussi bien que les détails de bas niveau et des niveaux intermédiaires. Ils permettent enfin des capacités d'analyse concernant par exemple la consistance, la complétude, ou encore la correction de la définition d'un processus, ainsi que la représentation et l'analyse de contraintes sur le processus comme des régulations, des standards, ...

L'approche suivie définit donc un cadre de travail pour la représentation d'un processus, et fournit un moyen permettant de décrire différents modèles et méthodes de conduite de développement. Elle permet de faciliter les raisonnements concernant le processus, ainsi que d'analyser les descriptions des événements.

Chapitre 4

Etude de cas

Introduction

Ce chapitre constitue une étude de cas de ce que nous avons vu dans le chapitre précédent, et plus particulièrement ce qui concerne la spécification d'une méthode de conception, ainsi que l'historique d'un développement. Il a pour but de montrer que le langage qui a été défini peut être utilisé dans le cas particulier d'une méthode de conception de bases de données, et que les éléments introduits pour "tracer" un processus permettent bien d'en spécifier l'historique.

La première section est une présentation générale de la méthode que nous spécifierons avec le langage. Elle en définit le fonctionnement et les grandes étapes qui la constituent.

La deuxième section concerne la spécification proprement dite de la méthode. Dans la première partie, elle présente la spécification des différents produits utilisés par celle-ci, la seconde traite des processus qu'elle met à la disposition du concepteur.

La troisième et dernière section propose une description succincte d'une partie de l'historique d'un développement exécuté suivant la méthode spécifiée. Elle a simplement pour objectif de montrer sur un exemple l'utilité des différents concepts que nous avons présentés.

4.1 Présentation générale de la méthode

Comme nous l'avons vu dans le deuxième chapitre, la conception d'une base de données est une définition des structures logiques et physiques d'une ou plusieurs bases de données pour servir les besoins d'information de l'utilisateur dans une organisation en fonction d'un ensemble d'applications.

Les objectifs sont multiples et pas toujours faciles à atteindre. Le problème est d'autant plus compliqué que le processus commence souvent avec des requirements relativement informels et plutôt mal définis. D'une manière générale, il consiste à transformer la perception qu'a un utilisateur du monde réel en un ensemble de produits qui représentent la base de données, et en tenant compte de certaines contraintes.

La méthode que nous allons présenter ici est constituée d'un certain nombre de phases utilisant chacune un ensemble de produits définis dans une phase précédente. Chaque phase fait subir des transformations aux produits avant de les fournir comme résultat. Cette méthode peut être considérée comme relativement générique étant donné qu'elle englobe les différentes approches actuellement utilisées. Elle est supportée par le modèle Entités/Associations (dont nous avons rappelé les principaux concepts dans le deuxième chapitre) que nous utiliserons comme référence pour la représentation des produits manipulés par les processus.

Les phases principales que nous avons identifiées pour cette méthode de conception de bases de données sont :

- *L'analyse conceptuelle* : elle conduit à élaborer un schéma conceptuel, qui est une description complète du système d'information indépendante de tout outil informatique. Cette phase utilise souvent un modèle de représentation de données de haut niveau qui décrit les structures sémantiques représentant le réel perçu.

- *La conception logique* : Durant cette phase, le schéma conceptuel issu de l'analyse conceptuelle est transformé en un modèle de données conforme à un SGBD (système de gestion de bases de données). Cette phase peut commencer après le choix d'un modèle d'implémentation et ne pas attendre de connaître le SGBD spécifique. On peut par exemple décider d'un modèle relationnel sans savoir lequel en particulier et développer le schéma correspondant. Le résultat de cette phase est un schéma logique des données qui décrit celles-ci selon les concepts habituels des SGBD, sous l'angle de la sémantique qu'elles expriment et des accès dont elles peuvent faire l'objet.

- *La définition des vues externes* : Un utilisateur final ou un groupe d'utilisateurs ne manipulent qu'un sous-ensemble des données du système d'information. Cette phase a pour objectif de définir plusieurs schémas issus du schéma logique qui concernent chaque application ou mise en oeuvre par un utilisateur.

- *La conception physique* : Elle a pour but d'adapter le schéma logique à la configuration matérielle et logicielle en optimisant les performances du système. A la fin de cette étape, nous obtenons une structure de données qui soit correcte, efficace et exécutable sur une machine réelle dans un SGBD donné, par l'intermédiaire d'un langage de description de données (LDD).

- *Le processus de reverse engineering* peut aussi être considéré comme un processus de conception qui transforme des produits d'entrée (généralement du texte de code) en une description conceptuelle de la base de données.

Chacune de ces phases peut être considérée comme un sous-processus permettant de réaliser le processus global de conception. Elles peuvent être elles-mêmes décomposées en plusieurs étapes qui permettront de les exécuter.

Dans cette section, nous allons donner une spécification des différents processus qui seront mis à la disposition du concepteur par la méthode et qui lui permettront de développer les différents produits. Ces spécifications sont bien évidemment basées sur ce que nous avons vu précédemment à savoir principalement une vision transformationnelle, et des mécanismes de décomposition d'un processus.

4.2 Spécification de la méthode

4.2.1 Spécification des produits

Cette partie contient la description des produits qui peuvent être utilisés ou construits par le concepteur. Ils seront utilisés dans la deuxième partie pour spécifier les arguments et les résultats des différents processus. Comme nous l'avons vu, leurs propriétés servent de précondition à l'exécution de ceux-ci.

Les deux produits "TEXT" et "GER" sont des produits standards définis pour toute méthode. Un produit de type TEXT est constitué de caractères et peut être utilisé pour des produits tels que les spécifications, le code, ... Un produit GER représente un schéma Entités/Associations le plus général. La définition d'un produit représentant un schéma utilisant les mêmes formalismes peut être déduite de celui-ci.

```
PRODUCT  Requirements
        SUMMARY :  "Les requirements sont les spécifications du système
                        à construire en fonction des besoins de l'utilisateur"
        IS  TEXT
        WITH
                Mots
        PROPERTIES

END-PRODUCT  Requirements


PRODUCT  Text-Code
        SUMMARY :  "Texte représentant le schéma exécutable d'une
                        base de données"
        IS  TEXT
        WITH
                Mots
        PROPERTIES
                Syntaxe-OK
                        %  Les mots forment un texte syntaxiquement correct

END-PRODUCT  Text-Code


PRODUCT  ERA
        SUMMARY :  "Le schéma ERA permet de représenter l'univers  du discours avec les concepts
                        de type d'entité, type d'association, attribut, contraintes d'intégrité."
        IS  GER
        WITH
                Type-d-entite
                Type-d-association
                Attribut
                Contrainte-d-integrite
        PROPERTIES
                Nom-id-TE
                        %  Les types d'entité doivent avoir des
                                noms différents

                Nom-id-TA
                        %  Les types d'association doivent avoir des
                                noms différents

                TA-deg-min-2
                        %  Le degré minimum d'un type d'association
                                est 2

END-PRODUCT  ERA
```

```
PRODUCT  ER-conceptuel
       SUMMARY :  "Le schéma ER-conceptuel est une spécialisation du schéma ERA dans lequel
                     les concepts sont exprimés sous une forme abrégée et les contraintes de
                     structure  sont plus restrictives."
       IS  ERA
       WITH
              Type-d-entite                  AS      T-entite
              Type-d-association             AS      T-assoc
              Attribut                       AS      attribut
              Contrainte-d-integrite         AS      contr-int
       PROPERTIES
              TA-deg-max-4
                     %  le degré maximum d'un type d'association
                                   est 4

END-PRODUCT  ER-conceptuel


PRODUCT  ER-conceptuel-normalise
       SUMMARY :  "Définition du schéma conceptuel normalisé"
       IS  ER-conceptuel
       WITH  all
       PROPERTIES
              Forme-norm-3
                     %  le schéma est sous troisième forme normale

END-PRODUCT  ER-conceptuel-normalise


PRODUCT  ER-logique-binaire
       SUMMARY :  "définition du schéma logique binaire comme une spécialisation du schéma ERA"
       IS  ERA
       WITH
              Type-d-entite                  AS      T-entite
              Type-d-association             AS      T-assoc
              Attribut                       AS      attribut
              Contrainte-d-integrite         AS      contr-int
       PROPERTIES
              TA-deg-max-2
                     %  Le degré maximum d'un type d'association
                                   est 2

              TA-nbr-attr-0
                     %  Un type d'association ne peut pas
                                   avoir d'attributs

END-PRODUCT  ER-logique-binaire


PRODUCT  ER-logique-quantifie
       SUMMARY :  "Définition du schéma logique avec les accès quantifiés"
       IS  ER-logique-binaire
       WITH  all                      -- Le modèle utilise les mêmes concepts
             Acces                    -- avec en plus celui d'accès
       PROPERTIES
              Acces-quantif
                     %  Tous les accès doivent être quantifiés

END-PRODUCT  ER-Logique-quantifie


PRODUCT  ER-logique-optimise
       SUMMARY :  "Définition du schéma logique optimisé par rapport aux accès"
       IS  ER-logique-binaire
       WITH  all                      -- Le modèle utilise les mêmes concepts
       PROPERTIES
              Acces-Optim
                     %  le schéma est optimisé par rapport aux accès

END-PRODUCT  ER-Logique-Optimise
```

```
PRODUCT  Relationnel
        SUMMARY :  "Définition du schéma conforme au relationnel"
        IS  GER
        WITH
                Table
                Cle-Acces
        PROPERTIES
                ID-Cle-Acces
                        %  Tout identifiant est une clé d'accès

END-PRODUCT  Relationnel
```

4.2.2 Spécification des processus

4.2.2.1 Le processus global de conception

Comme nous venons de le voir, ce processus est décomposé en plusieurs phases. Au départ, le concepteur a le choix entre deux possibilités. La première concerne un processus de conception qui lui permet, à partir de spécifications ou requirements initiaux, de construire un schéma d'une base de données opérationnelle. La deuxième possibilité est un processus de reverse engineering qui, à partir des textes de code, permet de retrouver la structure des données.

Spécification

```
        PROCESS  CBD
                SUMMARY :  "Processus de conception de BD"

                INPUT :  Requirements          --  Pour la conception
                            OR
                         Text-Code             --  Pour le reverse engineering

                OUTPUT : ER-Conceptuel-Normalise
                            ER-Logique-Optimise
                            Text-Code
                            OR
                            ER-Conceptuel

                PROCEDURE :
                        ALTERNATIVES
                            Conceptuel;Logique;Vues-Locales;Physique
                                                --  Conception de la BD à partir
                                                --  des requirements initiaux

                            Reverse-engineering
                                                --  Reconstruction du schéma
                                                --  conceptuel de la BD à partir
                                                --  du code.
                        END-ALTERNATIVES

        END-PROCESS  CBD
```

4.2.2.2 L'analyse conceptuelle

Ce processus consiste, à partir de l'analyse des besoins de l'utilisateur et des contraintes, à produire le schéma conceptuel des données. Il peut être réalisé suivant deux approches :

- Les requirements des différents utilisateurs et des applications sont regroupés en un seul ensemble avant de commencer la construction du schéma qui correspondra à l'ensemble des requirements mélangés. Les schémas externes correspondant aux applications seront alors extraits de ce schéma général par la suite.

- Dans la seconde approche, les requirements ne sont pas réunis, et un schéma est construit pour chaque groupe d'utilisateurs et application, basé sur les requirements individuels. Le résultat est donc un ensemble de schémas qui seront ensuite intégrés pour obtenir le schéma conceptuel global de toute la BD. Les vues externes pourront être construites à partir des sous-schémas conceptuels initiaux.

Après la construction du schéma conceptuel, celui-ci doit encore être normalisé afin d'observer différents critères imposés par la méthodologie, ou même définis par le concepteur lui-même.

```
PROCESS  Conceptuel
        SUMMARY :  "Construction du schéma conceptuel de la BD"

        INPUT :  Requirements
        OUTPUT : ER-Conceptuel-Normalise *
                                -- Plusieurs schémas peuvent être définis
        PROCEDURE :
                LOOP
                        ALTERNATIVES
                                C-Constr-Schema
                                        -- Conception de la BD à partir
                                        -- des requirements initiaux
                                C-Integration IF plusieurs-vues-definies
                                C-Normalisation IF NOT vide(current)
                                EXIT LOOP IF normalise(current)
                        END-ALTERNATIVES
                END-LOOP;
                Production-Schema
                        -- Validation et génération du code du schéma.
END-PROCESS  Conceptuel
```

Rien ne distingue les deux approches dans la spécification de la procédure. S'il choisit la première, l'utilisateur construira son schéma grâce au processus "C-Constr-Schema" qu'il n'exécutera qu'une seule fois. S'il opte pour la deuxième, il construira ses différents schémas en exécutant ce processus autant de fois qu'il y a de schémas. Il ne pourra évidemment utiliser le processus d'intégration que s'il a défini plusieurs schémas.

Enfin, il utilisera le processus "C-Normalisation" afin d'opérer les modifications nécessaires pour que le schéma réponde aux contraintes imposées.

Le processus d'analyse conceptuelle ne peut être terminé que si le schéma construit est bien normalisé, d'où la condition imposée pour sortir de la boucle.

```
PROCESS  C-Constr-Schema
        SUMMARY :  "Construction d'un schéma conceptuel"

        INPUT :     Requirements
        OUTPUT :    ER-Conceptuel

        PROCEDURE :
                ALTERNATIVES
                        Ouvrir
                        Creation
                END-ALTERNATIVES;
                LOOP
                        ALTERNATIVES
                                Ouvrir
                                Creation
                                Transformation
                                Evaluation
                                Rapport
                                EXIT LOOP
                        END-ALTERNATIVES
                END-LOOP;
                Production-Schema
                        --  Validation et génération du code du schéma.

END-PROCESS  C-Constr-Schema


PROCESS  C-Integration
        SUMMARY :  "Integration de différents schémas conceptuels"

        INPUT :  ER-Conceptuel  *
        OUTPUT :   ER-Conceptuel

        PROCEDURE :
                Ouvrir IF empty(current);
                LOOP
                        ALTERNATIVES
                                Ouvrir
                                Transformation  IF NOT empty(current)
                                Correspondance  IF NOT empty(current)
                                Integrer  IF NOT empty(current)
                                Rapport  IF NOT empty(current)
                                EXIT LOOP
                        END-ALTERNATIVES
                END-LOOP;
                Production-Schema
                        --  Validation et génération du code du schéma.

END-PROCESS  C-Integration


PROCESS  C-Normalisation
        SUMMARY :  "Normalisation du schéma conceptuel"

        INPUT :  ER-Conceptuel
        OUTPUT :   ER-Conceptuel-Normalise

        PROCEDURE :
                LOOP
                        ALTERNATIVES
                                C-Transform
                                C-Evaluate
                                C-Rapport
                                EXIT LOOP IF normalise(current)
                        END-ALTERNATIVES
                END-LOOP;
                Production-Schema
                        --  Validation et génération du code du schéma.

END-PROCESS  C-Normalisation
```

4.2.2.3 La conception logique

Ce processus permet d'obtenir, à partir du schéma conceptuel, un schéma dans le modèle du SGBD choisi. Il est réalisé en deux étapes principales :

- Transformation du schéma conceptuel en schéma logique sans considérer de caractéristiques spécifiques concernant l'implémentation dans le SGBD choisi.

- Modification du schéma en fonction d'un SGBD spécifique avec les caractéristiques et les contraintes correspondantes.

```
PROCESS  Logique
        SUMMARY :  "Construction du schéma logique des données"

        INPUT :  ER-Conceptuel-Normalise
        OUTPUT :   ER-Logique-Optimise

        PROCEDURE :
                ALTERNATIVES
                        L-Expert
                        L-Automatique
                        L-Binaire
                END-ALTERNATIVES;
                L-Quantification;
                L-Optimisation;
                L-Traduction;
                Production-Schema
                        --  Validation et génération du code du schéma.

END-PROCESS  Logique


PROCESS  L-Expert
        SUMMARY :  "Transformation du schéma conceptuel en schéma logique des données
                    conforme au SGBD choisi en expert"

        INPUT :  ER-Conceptuel-Normalise
        OUTPUT : ER-Binaire-Logique

        PROCEDURE :
                ALTERNATIVES
                        L-DB2-Expert
                        L-Oracle-Expert
                        L-Sybase-Expert
                        L-IDS-2-Expert
                        L-UDS-2-Expert
                        L-Cobol-Expert
                END-ALTERNATIVES

END-PROCESS  L-Expert


PROCESS  L-Automatique
        SUMMARY :  "Transformation automatique du schéma conceptuel en schéma logique
                    des données conforme au SGBD choisi"

        INPUT :  ER-Conceptuel-Normalise
        OUTPUT : ER-Binaire-Logique

        PROCEDURE :
                ALTERNATIVES
                        L-DB2-Automatic
                        L-Oracle-Automatic
                        L-Sybase-Automatic
                        L-IDS-2-Automatic
                        L-UDS-2-Automatic
                        L-Cobol-Automatic
                END-ALTERNATIVES

END-PROCESS  L-Automatique
```

```
PROCESS  L-Binaire
        SUMMARY :  "Transformation manuelle du schéma conceptuel en schéma binaire"

        INPUT :  ER-Conceptuel-Normalise
        OUTPUT : ER-Binaire-Logique

        PROCEDURE :
                LOOP
                        ALTERNATIVES
                                Transform-binaire
                                Transform-global
                                Transform-elem
                                L-Report
                                EXIT LOOP IF ER-Binaire(current)
                        END-ALTERNATIVES
                END-LOOP

END-PROCESS  L-Binaire


PROCESS  L-Quantification
        SUMMARY :  "Quantification des accès du schéma logique des données"

        INPUT :  ER-Logique-Binaire
        OUTPUT : ER-Logique-Quantifie

        PROCEDURE :
                LOOP
                        ALTERNATIVES
                                Quant-update
                                L-Transform
                                L-Evaluation
                                L-Rapport
                                EXIT LOOP
                                        IF ER-logique-quantifie(current)
                        END-ALTERNATIVES
                END-LOOP

END-PROCESS  L-Quantification


PROCESS  L-Ind-Optimisation
        SUMMARY :  "Optimisation du schéma logique des données indépendamment
                    d'un SGBD spécifique"

        INPUT :  ER-Logique-Quantifie
        OUTPUT : ER-Logique-Ind-Optimise

        PROCEDURE :
                LOOP
                        ALTERNATIVES
                                Denormalisation
                                Redondances-Struct
                                Restructuration
                                L-Evaluation
                                L-Report
                                EXIT LOOP
                        END-ALTERNATIVES
                END-LOOP;
                Production-Schema
                        --  Validation et génération du code du schéma.

END-PROCESS  L-Ind-Optimisation


PROCESS  L-Traduction
        SUMMARY :  "Transformation du schéma logique optimisé en un schéma conforme
                    à un type de SGBD"

        INPUT :  ER-Logique-Ind-Optimise
        OUTPUT :  Conforme-General          OR
                  DB2-Conforme              OR
                  Oracle-Conforme           OR
                  Sybase-Conforme           OR
                  IDS-2-Conforme            OR
                  UDS-2-Conforme            OR
                  Cobol-Conforme
```

```
        PROCEDURE :
                ALTERNATIVES
                        Traduction-generale
                        DB2-Trad
                        Oracle-Trad
                        Sybase-Trad
                        IDS-2-Trad
                        UDS-2-Trad
                        Cobol-Trad
                 END-ALTERNATIVES;
                Production-Schema
                        -- Validation et génération du code du schéma.
END-PROCESS  L-Traduction
```

4.2.2.4 La définition des vues externes

Elle permet de définir plusieurs schémas issus du schéma logique qui concernent chaque application de l'utilisateur. Cette étape consiste à produire, à partir du schéma logique des données et en fonction des différentes applications, plusieurs schémas qui concernent chacun une de ces applications particulières.

```
PROCESS  Vues-Locales
        SUMMARY :  "Définition des vues utilisateur dans le SGBD choisi"

        INPUT :  ER-Logique-Optimise *
        OUTPUT :  DB2-Conforme *              OR
                  Oracle-Conforme *           OR
                  Sybase-Conforme *           OR
                  IDS-2-Conforme  *           OR
                  UDS-2-Conforme  *           OR
                  Cobol-Conforme

        PROCEDURE :
                ALTERNATIVES
                        Vue-DB-2
                        Vue-Oracle
                        Vue-Sybase
                        Vue-IDS-2
                        Vue-UDS-2
                        Vue-Cobol
                END-ALTERNATIVES;

END-PROCESS  Vues-Locales


PROCESS  Vue-DB2
        SUMMARY :  "Définition des vues utilisteur en DB2"

        INPUT :  ER-Logique-Optimise *
        OUTPUT : DB2-Conforme *

        PROCEDURE :
                LOOP
                        Definition-Vue-DB2;
                        Generation-Vue-DB2
                        EXIT LOOP
                END-LOOP

END-PROCESS  Vue-DB2
```

La spécification est exactement la même pour les autres SGBD.

4.2.2.5 La conception physique

Cette étape produit le schéma exécutable de la base de données et des vues utilisateur. Le schéma fait d'abord l'objet d'une optimisation relativement au SGBD choisi s'il ne l'est pas déjà. Le schéma optimisé, ainsi que les vues utilisateur sont ensuite exprimés dans un texte de définition utilisable par les programmeurs et directement exécutable.

Par exemple, la démarche de conception appliquée à la production d'une base de données relationnelle (ex. SQL) produira un texte avec la définition de la base de données, et la définition des index exprimant les clés d'accès spécifiées dans le schéma logique.

```
PROCESS  Physique
        SUMMARY :  "Construction du schéma exécutable de la base de données et
                    des vues utilisateur"
        INPUT : ER-Logique-Optimise *                OR
                  Conforme-General *                 OR
                  DB2-Conforme    *                  OR
                  Oracle-Conforme *                  OR
                  Sybase-Conforme *                  OR
                  IDS-2-Conforme  *                  OR
                  UDS-2-Conforme  *                  OR
                  Cobol-Conforme *

        OUTPUT :  DB2-Text                      OR
                  Oracle-Text                   OR
                  Sybase-Text                   OR
                  IDS-2-Text                    OR
                  UDS-2-Text                    OR
                  Cobol-Text

        PROCEDURE :
                ALTERNATIVES
                        PH-Expert
                        PH-Automatic
                        PH-Manuel
                END-ALTERNATIVES;

END-PROCESS  Physique


PROCESS  PH-Manuel
        SUMMARY :  "Construction 'manuelle' du schéma exécutable de la base de données"

        INPUT :  ER-Logique-Optimise *               OR
                  Conforme-General *                 OR
                  DB2-Conforme    *                  OR
                  Oracle-Conforme *                  OR
                  Sybase-Conforme *                  OR
                  IDS-2-Conforme  *                  OR
                  UDS-2-Conforme  *                  OR
                  Cobol-Conforme  *

        OUTPUT : DB2-Text                       OR
                  Oracle-Text                   OR
                  Sybase-Text                   OR
                  IDS-2-Text                    OR
                  UDS-2-Text                    OR
                  Cobol-Text

        PROCEDURE :
                LOOP
                        ALTERNATIVES
                                DB2-Manuel
                                Oracle-Manuel
                                Sybase-Manuel
                                IDS-2-Manuel
                                UDS-2-Manuel
                                Cobol-Manuel
                                EXIT LOOP
                        END-ALTERNATIVES
                END-LOOP

END-PROCESS  PH-Manuel
```

4.2.2.6 Le reverse engineering

Il a pour but de retrouver la structure des données de la base à partir des textes de code. Une méthode permettant d'obtenir ce résultat est de diviser ce processus en deux étapes.

La première produit la description complète des structures de données selon le modèle du SGBD (Cobol, CODASYL, relationnel, ...). Les parties non conformes (procédurales,...) sont éliminées. Cette étape, appelée extraction des données, peut être perçue comme l'opération inverse de la conception physique.

La deuxième étape essaye de retrouver la sémantique des données. Elle consiste à retirer les constructions techniques (comme l'optimisation) et retrouver le sens des structures de données. Elle peut être considérée comme l'opération inverse de la conception logique.

```
PROCESS  Reverse-Engineering
        SUMMARY :  "Construction de la structure des données à partir du code exécutable"

        INPUT :  Text-Code
        OUTPUT : ER-Conceptuel

        PROCEDURE :
                Extraction-donnees;
                Conceptualisation-donnees;
                Production-Schema
                        --  Validation et génération du code du schéma.

END-PROCESS  Reverse-Engineering


PROCESS  Extraction-donnees
        SUMMARY :  "recherche de la structure logique des données à partir du code"

        INPUT :  Text-Code
        OUTPUT : ER-Logique
        PROCEDURE :
                LOOP
                        ALTERNATIVES
                                Analyse-code
                                Integration-schema
                                L-Transform
                                L-Update
                                L-Rapport
                                EXIT LOOP
                        END-ALTERNATIVES
                END-LOOP

END-PROCESS  Extraction-donnees


PROCESS  Conceptualisation-donnees
        SUMMARY :  "Construction du schéma conceptuel des données extraites du code"

        INPUT :  ER-Logique
        OUTPUT : ER-Conceptuel

        PROCEDURE :
                LOOP
                        ALTERNATIVES
                                DeTraduction
                                DeOptimisation
                                C-Reconstruction
                                C-Integration
                        END-ALTERNATIVES
                END-LOOP

END-PROCESS  Conceptualisation-donnees
```

4.3 Historique d'une conception

4.3.1 Introduction

Cette deuxième partie du chapitre a pour but de présenter la spécification d'une partie de l'historique de l'exécution d'un processus de conception d'une BD. La définition des éléments décrits a été faite dans le chapitre précédent, et nous montrons ici une représentation concrète de leur mise en oeuvre.

L'exemple reprend la conception d'une base de données destinée à la gestion d'une bibliothèque. Pour ne pas entrer dans les détails, nous ne considérerons qu'un système simple, à savoir une collection d'ouvrages qui peuvent posséder plusieurs exemplaires, et qui ont été écrits par un auteur. Ces exemplaires peuvent être empruntés à la bibliothèque par un abonné (emprunteur).

Nous ne donnons pas la spécification complète de tout le processus, ce qui serait trop fastidieux. Nous nous contentons de présenter succinctement certaines parties des processus et produits, ce qui permet de donner une description générale de tous les concepts dont nous avons parlé, et de voir ce que doit contenir l'historique pour garder de manière efficace la trace d'un développement.

Enfin, le formalisme utilisé pour cette description n'a pas été présenté de manière rigoureuse. Nous décrivons les éléments d'une manière simple qui, à notre avis, ne laisse place à aucune équivoque possible quant à son interprétation étant donnée la présentation qui a été faite précédemment.

4.3.2 Architecture du développement

La manière dont le processus est exécuté, et l'évolution des différents produits en fonction des transformations qu'ils subissent et des décisions dont ils font l'objet nous permet de donner une représentation graphique du profil général du développement, constituée principalement des actions qui désignent les transformations appliquées sur les produits. Elle est inspirée de celle proposée dans [Souq93] qui représente un développement grâce à un arbre et/ou. Elle utilise aussi un arbre qui possède deux types de noeuds principaux :

- Les premiers types correspondent aux actions et aux produits. Une transition entre deux de ces noeuds exprime le fait que soit le produit est utilisé comme départ du processus ou de la séquence de processus, soit il est le résultat de son exécution.
- Le deuxième type de noeuds sont les noeuds 'ou' qui représentent la proposition d'hypothèses à ce moment du développement.

La figure 4.1 ci-dessous montre la représentation de l'exemple que nous proposons. Le processus démarre à partir des spécifications du système, à partir desquelles le concepteur commence à construire un schéma conceptuel. Arrivé à un certain point, il propose deux possibilités de représentation pour des concepts dans le schéma. Les deux hypothèses sont exploitées, mais il décide que la deuxième lui convient mieux et abandonne donc la première. Continuant le développement du produit issu de la deuxième alternative, il exécute différentes transformations pour obtenir un schéma logique qui fait aussi l'objet de deux possibilités de transformation. Les

deux ayant été développées, il choisit la seconde qui, après transformation, permet d'obtenir le schéma logique des données.

Ce schéma sert d'argument à deux processus différents. Le premier concerne la définition des vues externes, le second est la conception physique qui utilise en plus du schéma logique, les deux vues externes pour produire le code de la BD et un rapport de conception.

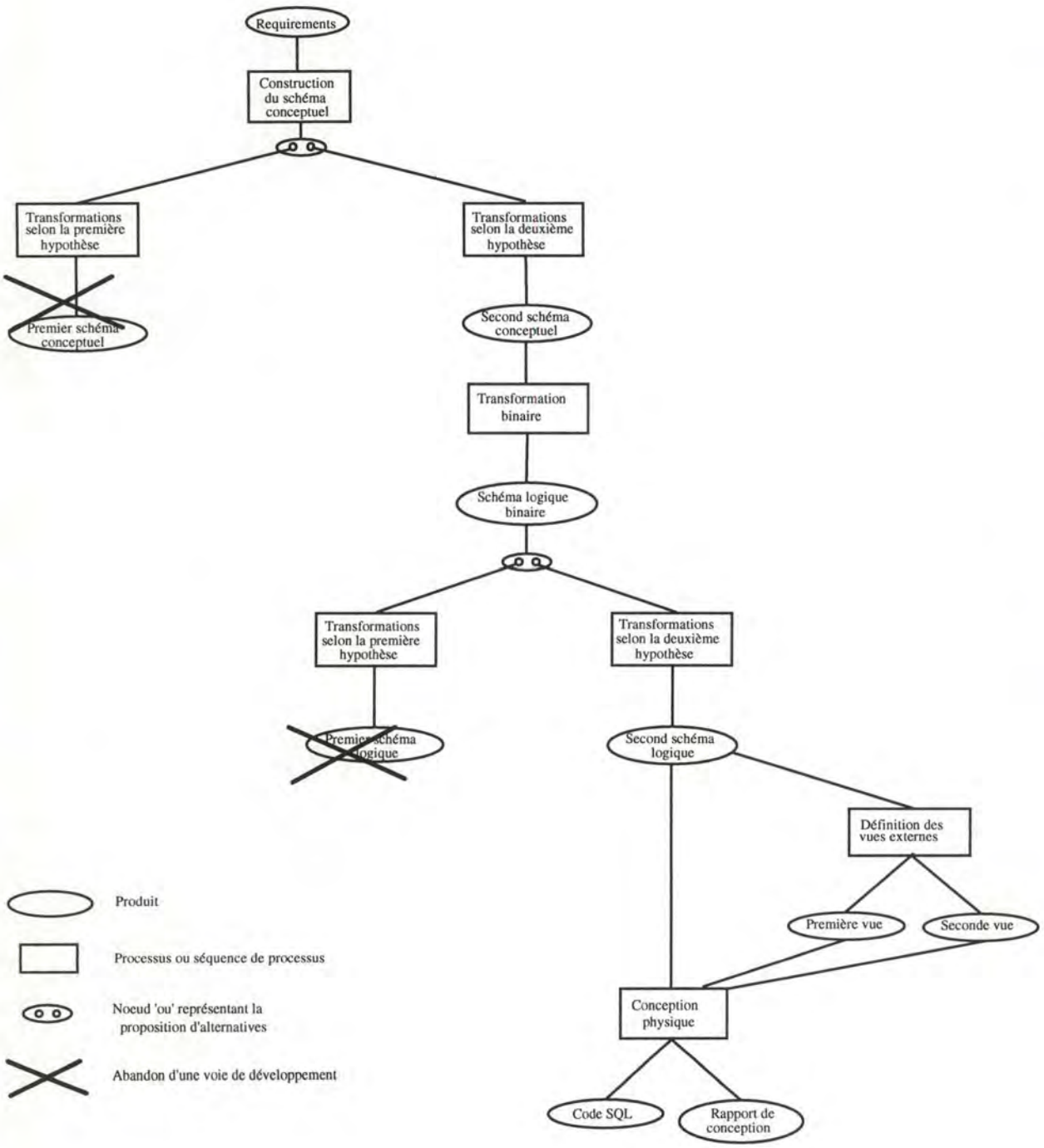


Fig. 4.1 Représentation graphique de l'exemple de développement

Cet exemple permet de montrer les différentes caractéristiques marquant la progression d'un développement. Nous y avons repris les concepts présentés, et montré la façon dont ils interviennent pour l'évolution du processus.

Cette structuration est évidemment insuffisante pour décrire complètement le processus puisqu'elle ne permet pas, par exemple, d'exprimer l'ordre d'exécution complet de toutes les actions, ni même de préciser les décisions utilisées pour l'obtenir. Cependant, la représentation d'un développement de cette manière permet de donner une idée de la stratégie suivie pour obtenir son état courant.

4.3.3 Spécifications

La spécification des concepts est constituée de plusieurs parties. La première concerne les processus exécutés. C'est principalement sur elle que nous nous concentrerons car c'est elle qui permet le mieux de se rendre compte des activités menées par le concepteur. Associés à cette description, les raisonnements utilisés seront spécifiés grâce aux concepts d'hypothèses et de décisions que nous présenterons dans des cadres à côté de l'endroit dans le développement où ils ont été introduits, ceci permettant de faciliter la lecture et la compréhension du processus. Dans la seconde partie, nous donnerons la description de quelques produits caractéristiques utilisés au cours du développement.

4.3.3.1 Historique des processus

```
Process  ConcBD-15081055
        Type :  CBD
        Assumption :
        Input : Biblio_req
        Output :

Process  Concept-15081056
        Type :  Conceptuel
        Assumption :
        Input :  Biblio_req
        Output :

Process  Constr_sch-15081059
        Type :  C-Constr-Schema
        Assumption :
        Input :  Biblio_req
        Output :

Process  Create_sch-15081059
        Type :  Creation
        Assumption :
        Input :
        Output :  Biblio
```

```
Process  Create_ET-15081101
        Type :  Creation_TE
        Assumption :
        Input :
        Output :  Auteur

...

Process  Save-15081132
        Type :  Enregistrer
        Assumption :
        Input :  Biblio
        Output :  Biblio-1

Process  Create_ET-15081140
        Type :  Creation_TE
        Assumption :  Hyp-15081137
        Input :
        Output :  Personne

...

Process  Save-15081155
        Type :  Enregistrer
        Assumption :
        Input :  Biblio
        Output :  Biblio-2

Process  Create_ET-15081200
        Type :  Creation_TE
        Assumption :  Hyp-15081159
        Input :
        Output :  Reservation

...

Process  Save-15081222
        Type :  Enregistrer
        Assumption :
        Input :  Biblio
        Output :  Biblio-3

Process  Dec-15081224
        Type :  Decision
        Assumption :
        Input :  Hyp-15081137
                 Hyp-15081159
        Output :

Process  Ouvr_sch-15081225
        Type :  Ouvrir
        Assumption :
        Input :
        Output :  Biblio-1

Process  Create_RT-15081227
        Type :  Creation_TA
        Assumption :
        Input :
        Output :  Ouvaut

...
```

Hypothese Hyp-15081137
Descr. : On peut créer un type d'entité "Personne" dont les types d'entité "Auteur" et "Emprunteur" seront des spécialisations.
Justif. : Ceci permettrait de considérer "Personne" comme un type avec ses propres composants, rassemblant les caractéristiques communes à Auteur et Emprunteur.

Hypothese Hyp-15081159
Descr. : On peut ajouter au schéma des concepts représentant la réalisation de demande de réservation d'ouvrages.
Justif. : Ceci permettrait d'introduire une fonction supplémentaire sur la BD.

Decision Dec-15081224
Subject : Hypothèses faites sur le schéma conceptuel et concernant l'introduction d'un type d'entité personne dont on dérive les types d'entités auteur et emprunteur, ainsi que celles concernant les réservations.
Descr : Eliminer les deux hypothèses proposées.
Justif. : Nous préférons avoir un schéma simple, et le problème des réservations n'est pas considéré comme nécessaire pour l'instant.

```
Process  Normal-15081242
        Type :  C-Normalisation
        Assumption :
        Input :  Biblio-1
        Output :

Process  Product_Sch-15081256
        Type :  Production-Schema
        Assumption :
        Input :  Biblio-1
        Output :  Conc_norm_Biblio-1

Process  Log-15081412
        Type :  Logique
        Assumption :
        Input :  Biblio-1
        Output :

Process  L-Bin-15081414
        Type :  L-Binaire
        Assumption :
        Input :  Biblio-1
        Output :  Biblio-1

...
```

Hypothese Hyp-15081452
Descr. : L'attribut répétitif "mots-clés" du type d'entité "Livre" peut être représenté par une seule chaîne de caractères dans laquelle ils seraient concaténés.
Justif. : Ceci permet d'avoir une structure plus simple pour cet l'attribut.

```
Process  Trsf-15081459
        Type :  Transformation
        Assumption :  Hyp-15081452
        Input :  mot-clé
        Output :  mot-clé

...

Process  Save-15081532
        Type :  Enregistrer
        Assumption :
        Input :  Biblio-4
        Output :  Biblio-5

Process  Ouvr_Sch-15081536
        Type :  Ouvrir
        Assumption :  Hyp-15081534
        Input :
        Output :  Biblio-4

Process  Trsf-15081540
        Type :  Transformation
        Assumption :  Hyp-15081534
        Input :  mot-clé
        Output :  Mot-clé

...
```

Hypothese Hyp-15081534
Descr. : L'attribut répétitif "mots-clés" du type d'entité "Livre" peut être représenté par un type d'entité.
Justif. : Ceci permettrait de traiter mot-clé comme un concept relativement indépendant.

```
Process  Dec-15081556
        Type :  Decision
        Assumption :
        Input :  Hyp-15081452
                 Hyp-15081534
        Output :

...
```

Decision Dec-15081556
Subject : La représentation du concept de mot-clé pour le type d'entité livre.
Descr : Nous choisissons de représenter les mots-clés par un type d'entité.
Justif. : Cette représentation du concept paraît la mieux adaptée par rapport à sa signification.

Process Save-15081632
Type : Enregistrer
Assumption :
Input : Biblio-1
Output : Biblio-4

Process L-Quant-15081638
Type : L-Quantification
Assumption :
Input : Biblio-4
Output : Biblio-4

Process L-Opt-15081646
Type : L-Optimisation
Assumption :
Input : Biblio-4
Output : Biblio-4

Process L-Trad-15081705
Type : L-Traduction
Assumption :
Input : Biblio-4
Output : Biblio-4

Process Product_Sch-15081712
Type : Production-Schema
Assumption :
Input : Biblio-4
Output : Log_Opt_Biblio-1

Process Vues-16080925
Type : Vues-Locales
Assumption :
Input : Biblio-4
Output :

Process Dec-16080927
Type : Decision
Assumption :
Input :
Output :

Decision Dec-16080927
Subject : La génération des vues externes.
Descr : Les vues externes seront traduites en DB2.
Justif. : SGBD choisi.

Process Vue-DB2-16080930
Type : Vue-DB2
Assumption :
Input : Biblio-4
Output :

Process Dec-16080935
Type : Decision
Assumption :
Input :
Output :

Decision Dec-16080935
Subject : Définition d'une vue externe.
Descr : Nous définissons une vue externe pour l'enregistrement d'un livre, et qui reprend les concepts d'ouvrage, exemplaire et auteur.
Justif. : Celle-ci correspond à une fonction d'utilisation

Process Def-Vue-16080940
Type : Definition-Vue-DB2
Assumption :
Input : Biblio-4
Output : Vue-Biblio-1

Process Gen-Vue-16081006
Type : Generation-Vue-DB2
Assumption :
Input : Vue-Biblio-1
Output : DB2-Vue-Biblio-1

Process Dec-16081025
Type : Decision
Assumption :
Input :
Output :

Decision Dec-16081025
Subject : Définition d'une vue externe.
Descr : Nous définissons une vue externe pour le prêt d'un exemplaire qui reprend les concepts d'exemplaire et emprunteur.
Justif. : Celle-ci correspond à une fonction d'utilisation.

```
Process  Def-Vue-16081027
        Type :  Definition-Vue-DB2
        Assumption :
        Input :  Biblio-4
        Output :  Vue-Biblio-2

Process  Gen-Vue-16081052
        Type :  Generation-Vue-DB2
        Assumption :
        Input :  Vue-Biblio-2
        Output :  DB2-Vue-Biblio-2


Process  Phys-16081124
        Type :  Physique
        Assumption :
        Input :  Biblio-4
        Output :


Process  PH-Man-16081127
        Type :  PH-Manuel
        Assumption :
        Input :  Biblio-4
        Output :

Process  DB2-Man-16081132
        Type :  DB2-Manuel
        Assumption :
        Input :  Biblio-4
        Output :  DB2-Text-Biblio-1
```

4.3.3.2 Quelques produits de l'historique

```
Product  Biblio-req
Type :  Requirements
Desc :  "Texte des requirements et spécifications de la BD Biblio"


Product  Auteur
Type :  Type-d-entite
Desc :  "Type d'entité représentant l'auteur d'un livre dans la BD Biblio"


Product  Conc-Norm-Biblio-1
Type :  ER-Conceptuel-Normalise
Desc :  "Schéma conceptuel normalisé de la BD Biblio"


Product  Log-Opt-Biblio-1
Type :  ER-Logique-Optimise
Desc :  "Schéma logique optimisé de la BD Biblio"


Product  Vue-Biblio-1
Type :  DB2-Conforme
Desc :  "Vue externe de la BD Biblio utilisée pour l'enregistrement d'un livre"


Product  DB2-Text-Biblio-1
Type :  Text-Code
Desc :  "Code DB2 de la BD Biblio"
```

Conclusion

L'objet de ce chapitre concernait une application des concepts introduits dans le chapitre précédent à la description d'une méthode particulière de conception de bases de données.

Les principales étapes de la méthode, qui possède le profil général de la majorité de celles utilisées dans ce domaine, sont décrites dans la première section.

La spécification présentée dans la deuxième section va évidemment plus loin dans la décomposition des étapes qui la constituent. Elle reprend tout d'abord la spécification des produits, ensuite celle des différents processus qui les utilisent ou permettent de les construire. Ces processus ont été décomposé jusqu'à un degré de précision que nous estimions suffisant pour l'exemple. Il est clair que dans le cadre d'une spécification concrète en vue d'une interprétation ou d'une éventuelle intégration au sein d'un outil, ils devront être spécifiés de manière plus rigoureuse jusqu'à une relation directe avec des fonctionnalités de cet outil.

La description de l'historique clôturant le chapitre n'est pas non plus complète. Nous n'avons présenté que les principaux processus et produits mis en oeuvre, ainsi qu'un tout petit nombre des éléments de raisonnement utilisés. Nous espérons que cette simplicité, délibérément choisie pour ne pas surcharger le travail, n'empêchera pas le lecteur de comprendre ce qui est représenté, ni de voir l'importance des éléments qui sont spécifiés.

Chapitre 5

Intégration dans un atelier de génie logiciel

Introduction

Ce chapitre concerne l'étude de l'intégration du modèle présenté dans le chapitre 3 au sein d'un atelier de génie logiciel (AGL). En particulier, il a pour objectif de définir les concepts nécessaires à la définition d'un moteur de méthode qui doit être ajouté au sein d'un outil particulier.

La première section a pour but de définir et de présenter les caractéristiques des outils d'aide à la conception (outils CASE). Nous nous attarderons un peu plus sur la présentation des outils d'aide à la conception des bases de données, ainsi que sur l'aspect méthodologique de ces outils.

La deuxième section concerne l'intégration au sein de l'atelier DB-MAIN (que nous aurons présenté) des éléments représentant les concepts du moteur de méthode. Après avoir décrit le rôle et les objectifs de celui-ci, nous spécifierons plus en détail les différents objets qui viendront s'ajouter à l'architecture actuelle de l'outil.

5.1 Les outils CASE

5.1.1 Description générale

Face à la complexité croissante des systèmes et à la difficulté de les concevoir, l'emploi d'outils automatisés s'avère de plus en plus utile au concepteur pour l'aider à obtenir des résultats corrects et correspondant bien à l'objectif fixé.

Les environnements logiciels d'aide au développement, présentés sous le nom de Computer Aided Software Engineering (CASE) fournissent un ensemble d'outils qui permettent d'aider le concepteur à faire de son développement une activité efficace. Le concept d'outil CASE recouvre les outils d'aide supportant toute activité de développement, depuis la planification et l'analyse des spécifications du système à construire jusqu'à son codage, son implémentation, les tests de sa validité, et sa maintenance.

La fonction de ces outils a aussi pour but d'assurer le contrôle sur les différents produits, comme par exemple la vérification et la validation. Les tâches dans ce domaine sont principalement la planification, le développement et le contrôle du produit. L'objet est un système logiciel dans les différentes étapes de son développement. L'assurance de sa qualité est devenue l'objectif le plus

important, et son contrôle ne doit pas être une des dernières étapes du développement, mais inhérent et simultané au processus de construction.

Une première caractéristique de la technologie CASE est le support graphique. La grande majorité des outils intègrent des moyens textuels et graphiques permettant l'entrée et la présentation des données. Ajoutés à une technologie hardware relativement avancée, les graphiques ou diagrammes ne sont plus lents et difficiles à produire, et cette capacité de présentation a largement contribué à l'acceptation et l'utilisation des outils CASE. Par exemple, STP (Software Through Pictures) [Wass87] est un outil CASE qui supporte des éditeurs graphiques multiples.

Ensuite, il est aussi reconnu que dans les développements de systèmes, les utilisateurs ont besoin de moyens de "feed-back" qui leur permettent d'augmenter la confiance en leur travail. Ces moyens vont de la simple vérification et analyse (spécifications, aspects performances, ...) jusqu'à l'animation et la simulation pour l'interprétation des conséquences possibles des activités de développement. Ces outils permettant le feed-back ont donc été intégrés aux environnements de conception, augmentant leur utilité, et étendant leur utilisation sur une plus grande partie du développement. Un exemple d'un tel outil CASE est 'STATEMATE' [Har90] qui fournit un bon support graphique pour ses relations, et 'STATECHARTS' [Har87], simulation intégrée pour l'analyse et les feed-backs, et générateur rapide de prototype.

Enfin, une troisième réalisation, plus récente, dans le domaine des outils CASE, représente une conséquence naturelle de la recherche dans les éditeurs syntaxiques. Ceux-ci, supportant une notation ou un langage unique au départ, ont évolué jusqu'à des éditeurs généralisés, permettant de générer des éditeurs spécifiques en définissant leur grammaire, leurs règles ou un ensemble de symboles graphiques. Ils fournissent ainsi une bonne base pour des "méta-outils" CASE à partir desquels d'autres outils peuvent être générés, permettant ainsi à un utilisateur de définir son propre outil avec sa grammaire et ses règles, ou même de redéfinir le méta-outil lui-même et d'en obtenir une nouvelle version à partir d'une précédente.

Une certaine maturité a été atteinte dans le domaine des outils CASE. Même si ceux-ci sont toujours en cours d'évolution et subissent de nouvelles améliorations, la plupart d'entre eux sont aujourd'hui facilement utilisés. Ceci est dû en grande partie au progrès dans les interfaces graphiques qui a considérablement amélioré leur souplesse et leur manipulation. Une des difficultés majeures pour l'ingénieur est le volume de faits, commandes, et règles qu'il doit pouvoir utiliser et maîtriser pour une utilisation efficace de ces environnements. Leur convivialité fait qu'ils sont utilisables intuitivement, et qu'il n'est plus nécessaire de consacrer des heures à leur apprentissage pour pouvoir les utiliser.

5.1.2 Les outils CASE de conception de BD

5.1.2.1 Généralités

Faire évoluer les différentes applications d'une base de données est une activité relativement compliquée et qui dépend de la complexité de cette application, la qualité de sa conception, et la culture du personnel de développement et de maintenance. Différents problèmes pratiques peuvent surgir concernant la maintenance, la rétro-ingénierie, et la prévision des développements futurs de la BD. Certains aspects de la conception peuvent difficilement être réalisés manuellement, et doivent pouvoir être automatisés. Par exemple, la détection des conflits entre des schémas à intégrer est très difficile à faire de façon manuelle, évaluer les alternatives pour la conception physique peut prendre beaucoup de temps, la transformation globale de schémas est par contre relativement systématique et peut être facilement automatisée.

De plus en plus d'outils d'aide à la conception de bases de données font leur apparition, constituant une variante d'un outil CASE, ou bien un composant-même de celui-ci. Ils ont pour but d'aider le concepteur pour certains aspects du développement d'une BD. Ils concernent aussi bien les aspects conceptuels, logiques et physiques que les phases de collecte des requirements et d'analyse. Pour ne pas nous attarder sur la description de ces outils, nous décrivons simplement quelques caractéristiques qu'ils doivent posséder pour être suffisamment efficaces :

- Une **interface** facile à manipuler qui permet au concepteur de se consacrer sur sa tâche plutôt que sur l'utilisation de l'outil.

- Des **composants analytiques** pour les tâches difficiles à réaliser manuellement, comme l'évaluation d'alternatives ou de contraintes conflictuelles entre des vues.

- Des **composants heuristiques** qui, spécifiés dans l'outil, permettent d'automatiser les aspects qui ne peuvent pas être quantifiés précisément, ou d'évaluer certaines alternatives de conception.

- Des **moyens d'analyse** comparative pour aider les concepteur à choisir parmi plusieurs alternatives.

- L'**affichage** des résultats, aussi bien les schémas que des tables, des listes, et des rapports qui peuvent ainsi être facilement interprétés.

- Des **moyens de vérification** assurant que le résultat de la conception correspond bien aux requirements initiaux.

La fonction principale d'un outil de conception est de gérer la documentation concernant une base de données en projet. Ils ont pour objet de diminuer le travail pénible de certaines tâches, et améliorer les performances des systèmes développés. Cette fonction, qui concerne la spécification et l'analyse des requirements, l'aspect conceptuel, l'intégration, la transformation de schéma suivant différents modèles, les optimisations, les performances, l'ajustement, la réorganisation, la restructuration des systèmes,... peut se décomposer en sous-fonctions de saisie, de consultation et de modification des spécifications, ainsi que de production de documentation, sur support externe.

Elle peut aussi être couplée à des fonctions de transformation et de validation des spécifications, ainsi que de génération de descriptions destinées à un SGBD.

La saisie des spécifications peut se faire via un langage de spécification, ou bien grâce à un éditeur syntaxique ou graphique. Elle peut aussi se faire par analyse de descriptions exécutables (Reverse Engineering). La consultation des spécifications peut se faire sous forme de textes ou graphiques, et grâce à un mécanisme de sélection et de navigation au travers ces spécifications. Les modifications de spécifications se font généralement grâce à un éditeur syntaxique ou graphique. Elles sont ponctuelles ou globales, manuelles ou automatiques, et peuvent être assistées par un contrôle de cohérence en temps réel ou différé, et par des mécanismes de reprise arrière et avant (undo, redo). Les modifications qu'entraînent ces transformations doivent avoir des répercussions au travers des spécifications dérivées.

La validation des spécifications peut être de trois types : conceptuelle (syntaxe, cohérence et complétude, normalisation), logique (conformité à un SGBD) ou physique (conformité à un SGBD, complétude et cohérence des paramètres).

La production de documentation externe se fait sur un support externe (généralement papier) et est sous forme graphique ou textuelle. Elle peut être paramétrable.

La production de descriptions exécutables consiste à générer partiellement ou complètement un texte de description de la BD dans le langage de description de données (LDD) du SGBD cible choisi.

5.1.2.2 Faiblesses des outils CASE

Depuis plusieurs années maintenant, des modèles et méthodes ont été développés pour décrire, analyser et construire des bases de données. Les outils CASE sont peu à peu apparus par la suite. Actuellement, ces modèles, méthodes et outils sont largement répandus et il est devenu nécessaire de les maîtriser. Certains d'entre eux sont devenus des standards et sont disponibles sur tout le marché.

Cependant, d'une manière générale, les méthodes et les outils CASE reposent principalement sur un modèle simplifié du cycle de vie des BDs. [Hai94] Ils sont basés sur des stratégies simples et rigides qui donnent l'illusion que le processus de conception de bases de données est systématique et déterministe une fois que le schéma conceptuel a été élaboré. Certaines tâches, qui ont leur importance lors de la conception d'application moyennes ou plus importantes, sont parfois faiblement supportées ou même complètement ignorées, et certaines étapes de la conception sont sous-estimées. Les résultats en sont des produits qui valident les spécifications de manière correcte, mais qui s'avèrent inadéquats pour une implémentation opérationnelle. Ainsi, la majorité des outils CASE sont principalement concentrés sur l'étape de spécification conceptuelle, et laissent de côté la production d'un schéma physique efficace.

D'un autre côté, il faut savoir que le cycle de vie d'une BD comprend encore un certain nombre d'activités après qu'une première version de l'application ait été produite. La plupart des méthodes et outils semblent ignorer les étapes suivant cette conception, et même ne tiennent pas compte du fait que la plupart des développements actuels ne concernent pas de nouvelles applications, mais consistent à en étendre ou retransformer des existantes.

Il en ressort que la plupart des outils ne contiennent pas les moyens permettant au concepteur d'opérer un développement réellement fructueux. Ils ne possèdent en général pas suffisamment de concepts qui puissent être utilisés pour forcer, ni même aider celui-ci à suivre une méthodologie correcte depuis le début jusqu'à la fin de la conception. Afin de constituer des assistants véritablement efficaces, les outils doivent "comprendre" ce qu'ils font, c'est-à-dire savoir ce qui est réalisé, comment, et pourquoi. Ceci n'est possible que si on leur a inclus les éléments méthodologiques nécessaires à tout développement rigoureux et efficace.

5.1.3 L'aspect méthodologique dans les outils CASE

Les processus de développement sont souvent divisés en différentes étapes, depuis l'interprétation de la spécification des requirements jusqu'à la construction et la maintenance du système. Un grand nombre de méthodes ont été proposées pour améliorer celui-ci, chacune couvrant une étape. Elles consistent généralement en un ou plusieurs diagrammes de représentation avec un ensemble de procédures et heuristiques permettant d'obtenir une représentation complète et cohérente du système, ainsi que d'aider le concepteur à passer d'une étape à l'autre du développement.

Notre propos concerne les cadres de représentation pour l'intégration d'une méthode qui permette de fournir un support pour le cycle de vie complet du système, c'est-à-dire essayer de voir comment un outil CASE peut être utilisé pour supporter l'aspect méthodologique d'un processus de développement.

La plupart des outils CASE ont tendance à être concentrés sur des étapes particulières du processus de développement. Le besoin de s'étendre pour couvrir une plus grande partie de celui-ci et d'en fournir différentes vues nécessite l'intégration de différentes méthodes, en plus de celles de notations et d'outils. Ceci est en grande partie dicté par les échanges d'informations nécessaires réalisés par l'utilisateur de modèles de données et souvent supportés par une BD ou repository centralisée, et a l'avantage de fournir une base uniforme pour la vérification de la consistance. Le traitement des inconsistances est inhérent au besoin plus général de fournir un réel support pour la guidance méthodologique.

Plutôt que de remplacer le niveau d'un utilisateur, une méthode a plus pour objectif d'y ajouter un complément.

Les règles et les contraintes de la méthode utilisée sont faciles à proposer, par contre l'assistance et la guidance qui doivent pouvoir aussi être fournies à des utilisateurs de degrés d'expertise fortement variables sont beaucoup plus complexes à réaliser.

Il est donc important de pouvoir ajouter un support méthodologique au sein d'un environnement qui permette aux concepteurs de travailler efficacement au développement de systèmes. L'intégration d'ensembles prédéfinis de méthodes peut en plus constituer une aide importante pour la conduite des différentes étapes du processus.

5.2 Intégration

Nous présentons les principes et architecture d'un assistant de conception au sein d'un outil qui fait partie d'un projet dans lequel plusieurs écoles et compagnies industrielles sont impliquées.

L'objectif est d'inclure dans cet environnement les concepts méthodologiques pour la conception. Notre approche consistera à présenter tout d'abord l'environnement général et ses outils principaux, ensuite à définir les concepts nécessaires à l'intégration d'un moteur de méthode pour les inclure dans l'architecture actuelle de l'outil.

Les caractéristiques concernant les attributs et les méthodes des objets présentés sont décrites dans l'annexe B.

5.2.1 L'outil DB-MAIN

5.2.1.1 Présentation

DB-MAIN est un outil CASE destiné à l'ingénierie des applications de bases de données, et en particulier la conception, la rétro-ingénierie, la réingénierie et la maintenance [DBMAIN93]. Cet outil est un des principaux produits du programme DB-MAIN lancé par l'Institut d'informatique en septembre 1993, dont l'objectif est d'étudier les problèmes survenant lors de l'évolution des requirements des applications de BDs. Ce projet de recherche et transfert de technologie s'est donné comme but d'étudier les problèmes concernant l'évolution et la maintenance des fichiers et des bases de données dans les applications de gestion, et de proposer des techniques et des méthodes pratiques aidant les développeurs à résoudre ces problèmes. Il cherche aussi à développer un prototype d'outil CASE qui supporte ces concepts et ces techniques.

Comme la plupart des autres outils, DB-MAIN propose les fonctions standards pour l'entrée, la consultation, la validation et le stockage des spécifications. Il offre aussi des générateurs pour certains SGBD standards. Il possède en plus certaines caractéristiques plus originales, et qui ont pour but de satisfaire certains requirements évoqués précédemment, et supporter les activités de maintenance en plus de celles de développement. Nous présentons ici ces caractéristiques de manière succincte, on peut en trouver une description plus précise dans [Hai94].

- Le **repository** contient la représentation de l'historique d'un développement, avec la version de la méthodologie utilisée pour le conduire. Cela inclut la description précise des schémas, spécifications et processus qui ont été utilisés.

- Des **interfaces graphiques et textuelles** permettent de présenter les spécifications et de naviguer parmi celles-ci à travers différentes vues, et à différents niveaux de granularité.

- Une **boîte à outils de transformation** permet la manipulation des spécifications grâce à des opérateurs préservant la sémantique. Leur réversibilité est essentielle pour les activités de rétro-ingénierie et de maintenance.

- Un **moteur méthodologique**, conduit par la méthodologie de référence stockée dans le repository, permet l'adaptation de l'outil selon une méthodologie particulière.

- Un **assistant** offre des fonctions de haut niveau qui conduisent les opérations complexes (transformations globales de schémas, génération de code,...) et des heuristiques de support (normalisation, optimisation, analyse, ...).

- L'extraction des structures de données de la rétro-ingénierie est supportée par un "**parser**" de code source qui charge dans le repository l'expression abstraite des structures décodées dans les programmes d'application et les textes DDL.

Une première version de DB-MAIN est disponible, elle comprend les fonctions essentielles pour la conception et la rétro-ingénierie de bases de données, une boîte à outils de transformations de base, des parsers pour les codes IDS-2, SQL et COBOL, un assistant pour la manipulation des schémas, et un processeur pour les activités du journal et de replay.

Il est encore loin d'inclure toutes les caractéristiques que nous venons de présenter, mais il est déjà utilisé de manière intensive dans le domaine de l'enseignement et pour des applications de bases de données de grande taille.

5.2.1.2 Architecture

Pour ne pas alourdir le travail et nous attarder sur une présentation trop poussée de l'outil, ce qui n'est pas notre propos, nous ne présenterons pas son architecture. Celle-ci s'avérant cependant nécessaire pour ce qui suit, nous conseillons vivement la lecture de sa description dans [DBMAIN94].

5.2.2 Définition d'un sous-ensemble utile

Notre propos est de définir les concepts nécessaires à l'intégration d'un moteur méthodologique au sein de l'outil, ceux-ci venant s'ajouter à l'architecture actuelle du repository. Le rôle du moteur de méthode est double. D'une part, il concerne le contrôle du processus et l'aide à apporter au concepteur pour son développement et le suivi d'une méthode particulière. D'autre part, il a aussi pour mission de tenir à jour l'historique du processus.

5.2.2.1 Le contrôle de processus

L'intégration au sein d'un atelier de génie logiciel (AGL) des concepts de méthode tels que nous les avons présentés nous conduit tout d'abord à définir la notion de contrôle de processus. Ceci nous permet de décrire les mécanismes constituant les éléments principaux nécessaires pour forcer et aider le concepteur à suivre une méthode.

Ces mécanismes ont principalement pour objectif de pouvoir définir les fonctions qui pourront être activées à chaque moment dans le système, et en fonction d'un état particulier du

développement. Nous référant à ce que nous avons vu dans le chapitre 3 concernant le modèle de représentation de méthode, ce contrôle dépend de deux critères bien précis :

- d'une part, il est influencé par l'état des différents produits utilisés dans le développement. Un processus ne peut être utilisé que si les produits courants correspondent à ce dont il a besoin comme argument. Le contrôle de la validité des produits par rapport à la spécification des processus est appelé le **contrôle d'exécution**.

- d'autre part, il est évidemment issu de la méthodologie définie par le spécialiste et qui contient la spécification des processus mis à la disposition de l'utilisateur, ainsi que l'ordre dans lequel il doivent être appliqués. Nous appellerons ce contrôle le **contrôle de la séquence des opérations**.

a. Le contrôle d'exécution (par les produits)

Le contrôle d'exécution consiste à comparer la spécification des produits définis dans le développement courant avec les conditions d'application des processus spécifiés dans la méthode. Chaque processus pour lequel ces spécifications correspondent pourra être rendu activable par l'utilisateur, les autres ne pourront être exécutés.

Comme nous l'avons vu, les propriétés des produits définis dans la méthode constituent les préconditions à l'exécution des processus. Il est donc logique que l'activabilité de ces derniers soit aussi guidée par l'évaluation des caractéristiques des produits en cours de développement. Le rôle de l'évaluation des propriétés est double :

1. Tout d'abord, comme nous venons de le voir, elle permet de définir quels sont les processus que le concepteur peut utiliser, et donc limite fortement son choix parmi l'ensemble des processus fournis par la méthode.

2. Ensuite, il est possible de voir quelles sont les propriétés qui ne sont pas observées et de renseigner le concepteur sur celles-ci. Il peut alors plus facilement réfléchir aux transformations à appliquer pour qu'ils satisfassent aux conditions requises.

L'évaluation des propriétés des produits courants constitue donc une aide importante au développeur et permet en quelque sorte de fournir certaines lignes de conduite durant les phases du développement.

b. Le contrôle de la séquence des opérations (par la spécification des processus)

Comme nous l'avons expliqué au troisième chapitre, chaque processus est défini comme une suite de sous-processus dont l'exécution permet de réaliser l'objectif. La définition de la séquence d'exécution des sous-processus est précisée dans la partie procédurale de la spécification du processus grâce notamment aux mécanismes de contrôle du langage.

Le contrôle de la séquence des opérations a pour but de forcer l'utilisateur à suivre la méthode pour développer ses produits, et est donc réalisé grâce à l'interprétation de la partie procédurale, dans laquelle les différentes alternatives qui sont susceptibles d'être exécutées par l'utilisateur sont définies indépendamment de toute situation spécifique à un processus en cours d'exécution, mais bien en fonction de la méthode utilisée.

Pour pouvoir guider correctement l'utilisateur, il faut donc que le système connaisse à tout instant l'état de développement et les différentes options qu'il peut lui proposer pour continuer. Il doit donc élaborer et maintenir un contexte de développement, et un état méthodologique de celui-ci, qui représente un chemin et une étape courante dans le modèle de processus que constitue la méthode suivie. Chaque action du concepteur dans l'outil doit être passée au moteur afin que celui-ci puisse mettre à jour le contexte et l'état méthodologique du processus, et analyser son comportement avec un de ceux stockés dans la base des méthodes. Pour tout changement d'étape détecté, il vérifie que celui-ci peut avoir lieu étant données les spécifications de la méthode, et le valide. Il compare alors l'état courant du développement et les spécifications pour fournir des conseils méthodologiques au concepteur, et définir les processus suivants qui peuvent être utilisés.

5.2.2.2 L'historique

L'autre rôle du moteur de méthode concerne la mise à jour de l'historique des processus. Nous avons déjà parlé dans le troisième chapitre de l'importance de pouvoir garder une trace des éléments intervenant dans une exécution réelle. Nous ne décrirons pas à nouveau en quoi ils consistent, mais rappelons qu'ils concernent notamment la planification du processus, son contrôle et son amélioration.

Le moteur de méthode doit gérer l'enregistrement des différents événements et caractéristiques mis en oeuvre au cours d'un développement.

5.2.2.3 Les concepts et leur rôle

En ce qui concerne le premier objectif du moteur de méthode, et pour pouvoir faire face à la complexité des processus, il est nécessaire d'avoir des formalismes de représentation assez puissants, et nous croyons qu'il est important, en plus du contrôle tel que nous l'avons décrit, de pouvoir donner une description d'un processus courant, des prescriptions pour des processus "futurs", et des restrictions de tout type qui peuvent être imposées. Les concepts à définir pour l'intégration d'une méthode dans un atelier logiciel doivent absolument prendre en compte ces caractéristiques, et la spécification d'une méthode telle que nous l'avons proposée permet en partie d'y parvenir.

Nous allons présenter les concepts dont l'objet est de permettre la définition des éléments relatifs à une méthode particulière et qui puissent être utilisés pour aider le concepteur à réaliser la transformation des spécifications formelles ou informelles en un produit correspondant suivant une conception détaillée.

L'intégration a pour but de :

1. permettre le contrôle des actions du concepteur, voir si elles sont permises, l'obliger à suivre la méthodologie prescrite.

2. guider le concepteur pendant sa session, lui donner des conseils à propos des étapes sur lesquelles il travaille (quoi faire et comment), lui montrer les étapes suivantes qu'il peut entreprendre, ...

A n'importe quel moment, de nouveaux processus types doivent pouvoir être ajoutés à la base de connaissance par les spécialistes.

Au niveau de l'historique, les concepts proposés doivent permettre l'enregistrement de tous les éléments intervenant dans un développement. Ceux-ci concernent non seulement les processus exécutés et les produits construits, mais aussi les raisonnements utilisés pour leur réalisation. Ces concepts doivent pouvoir être utilisés pour retrouver toute l'information concernant le processus, et qui s'avérerait nécessaire pour satisfaire à l'un des objectifs que nous avons présentés à propos de l'historique.

5.2.3 Description des éléments d'architecture

Les objets que nous allons introduire sont évidemment issus du modèle de représentation que nous avons proposé dans le chapitre 3. Ils forment une architecture qui s'intégrera dans celle du méta-schéma actuel décrit dans [DBMAIN94], et y sont présentés suivant les mêmes formalismes et conventions.

La première partie concerne les éléments permettant de spécifier une méthode particulière, et qui doivent être utilisés par le moteur de méthode pour le contrôle du processus et l'aide au concepteur. La seconde partie traite des éléments relatifs à l'historique dont il gère l'instanciation au fur et à mesure du déroulement du processus.

5.2.3.1 Les objets de définition de méthode

a. Method

L'élément central de la spécification d'une méthode au sein de l'atelier est l'objet **Method**. C'est grâce à celui-ci que sont décrites les différentes caractéristiques relatives à une méthode particulière.

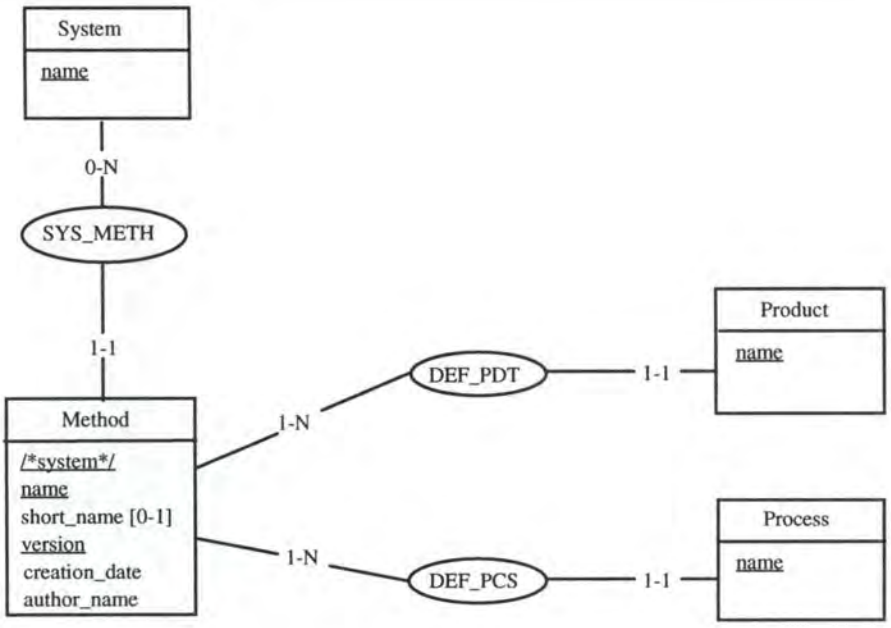


Fig. 5.1 L'environnement de l'objet **Method**.

L'objet **Method** est lié à l'élément principal du repository (**System**) via l'association **SYS_METH**.

Une méthode est caractérisée par un nom (*name*), un nom court (*short_name*), un numéro de version (*version*), une date de création (*creation_date*), ainsi que le nom de son auteur (*author_name*). Elle est identifiée par son nom, son numéro de version, et le rôle vers l'association **SYS_METH**.

Faisant référence à ce que nous avons vu au chapitre 3, une méthode définit un certain nombre de produits (via **DEF_PDT**) et de processus (via **DEF_PCS**) qui pourront être utilisés par le concepteur. Lors de la définition d'une nouvelle méthode au sein de l'atelier, l'ingénieur donne une valeur aux différents attributs, et l'objet créé est relié à l'objet système. Il doit ensuite procéder à la spécification des différents produits et processus définis par la méthode.

b. Product

L'objet **Product** sert à définir les caractéristiques des produits utilisés dans la méthode.

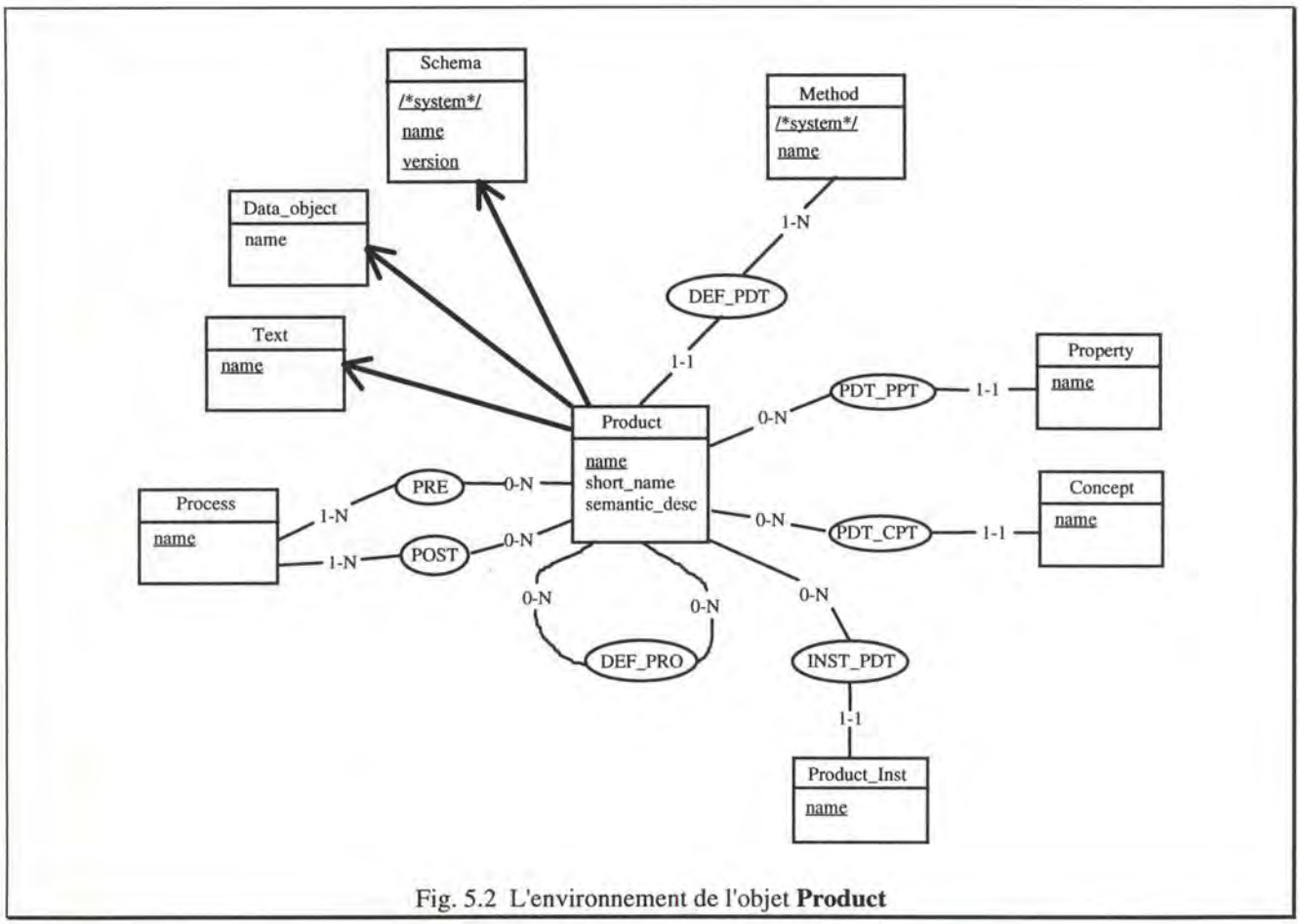


Fig. 5.2 L'environnement de l'objet **Product**

Un produit peut appartenir à trois types différents. Premièrement, il peut constituer un "Data_object" tel qu'il a déjà été spécifié dans l'architecture actuelle. Il s'agit par exemple d'un type d'entité, type d'association, ... Il peut aussi s'agir d'un "Schéma" dont la description a également été faite. Enfin, il peut encore être un "Text" qui permet de spécifier un document de spécifications, du code, ... Cet objet est décrit ci-après.

Un produit possède un nom (*name*), un nom court (*short_name*), et une description sémantique (*semantic_desc*).

Tout produit doit être défini au sein d'une méthode (via **DEF_PDT**). Il utilise un certain nombre de propriétés (via **PDT_PPT**) et de concepts (via **PDT_CPT**), et peut être considéré comme précondition (via **PRE**) et postcondition (via **POST**) pour l'exécution de processus.

L'association **DEF_PRO** permet de lier le produit à celui à partir duquel il est défini, ceci permettra au moteur de méthode de retrouver les caractéristiques héritées.

L'association **INST_PDT** permet de relier le produit type défini dans la méthode à toutes les instances dont il fait l'objet lors d'une exécution réelle, et qui font partie de l'historique.

La définition d'un produit au sein d'une méthode consiste tout d'abord à spécifier ses caractéristiques concernant son nom et sa description informelle. Il faut ensuite préciser, selon le cas, de quel autre produit il tire sa spécification, et apporter les modifications nécessaires en définissant d'autres propriétés ou concepts de structure.

c. Text

L'objet **Text** sert à représenter un produit correspondant à un texte comme par exemple des spécifications, du code, ...

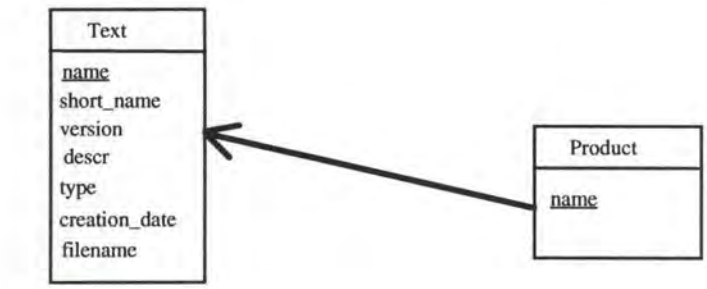


Fig. 5.3 L'environnement de l'objet **Text**

Un objet texte possède un nom (*name*), un nom court (*short_name*), un numéro de version (*version*), une date de création (*creation_date*) et une description sémantique (*descr*). Il est aussi caractérisé par un type qui permet de spécifier à quoi il correspond, comme par exemple 'texte Cobol 71', 'Code SQL', ... Enfin, un nom de fichier (*filename*) permet de spécifier le nom externe et le chemin d'accès du fichier dans lequel il est stocké.

d. Property

L'objet **Property** sert à définir les propriétés qui caractérisent les produits.

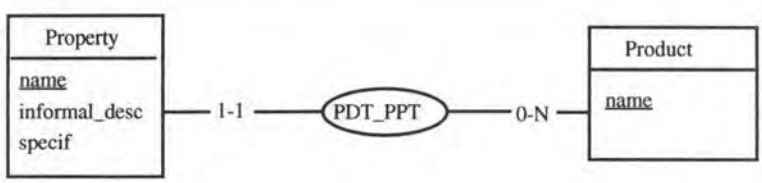


Fig. 5.4 L'environnement de l'objet **Property**

Une propriété est définie pour un et un seul produit. Elle possède un nom (*name*), une description informelle (*informal_desc*), et une spécification mathématique (*specif*) qui pourra être interprétée par le moteur méthodologique.

Rem. La cardinalité minimale du rôle joué par "Product" dans l'association **PDT_PPT** est 0. Si aucune propriété n'est définie pour un produit, cela signifie que celui-ci utilise telles quelles toutes celles du produit à partir duquel il est défini.

La définition de propriétés pour un produit peut servir soit à en ajouter des nouvelles, soit apporter des modifications à celles qu'il hérite du produit à partir duquel il est défini. Le moteur de méthode doit donc analyser les propriétés qui sont propres à ce produit pour soit compléter, soit modifier les propriétés héritées d'autres produits.

e. Concept

Un objet **Concept** permet de préciser les éléments utilisés par un produit pour définir sa structure.

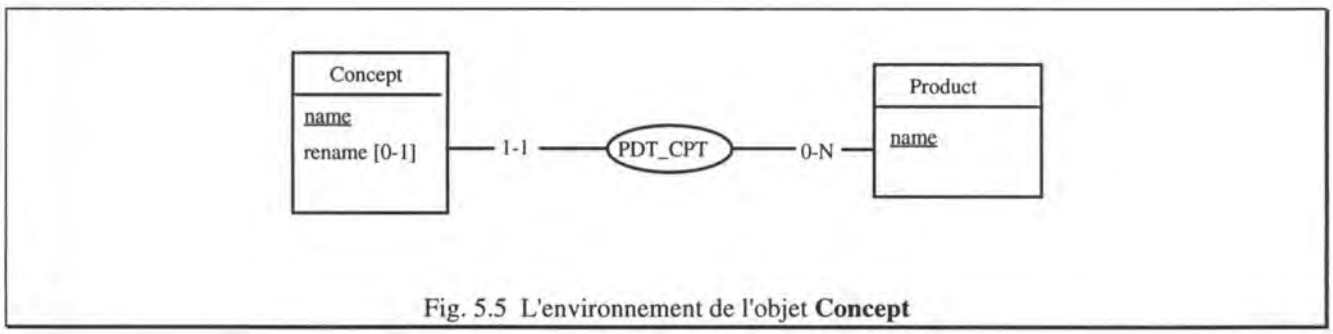


Fig. 5.5 L'environnement de l'objet **Concept**

Tout objet concept est défini pour un et un seul produit (via **PDT_CPT**). Il possède un nom (*name*), et un attribut facultatif (*rename*) lui permettant d'être renommé quand il est hérité d'un autre produit.

Rem. La cardinalité minimale du rôle joué par **Concept** dans l'association **PDT_CPT** est 0. Si aucun concept n'est défini pour un produit, cela signifie que celui-ci utilise tels quels ceux du produit à partir duquel il est défini.

De même que pour les propriétés, la définition d'un concept permet d'ajouter ou d'apporter des modifications à l'ensemble de ceux que le produit pourrait récupérer d'un autre. Le moteur utilise donc les nouveaux objets créés avec le produit pour mettre à jour l'ensemble des concepts qu'il utilise et qu'il a hérités des autres produits.

f. Process

L'objet **Process** sert à définir les différents processus mis à la disposition du concepteur par la méthode.

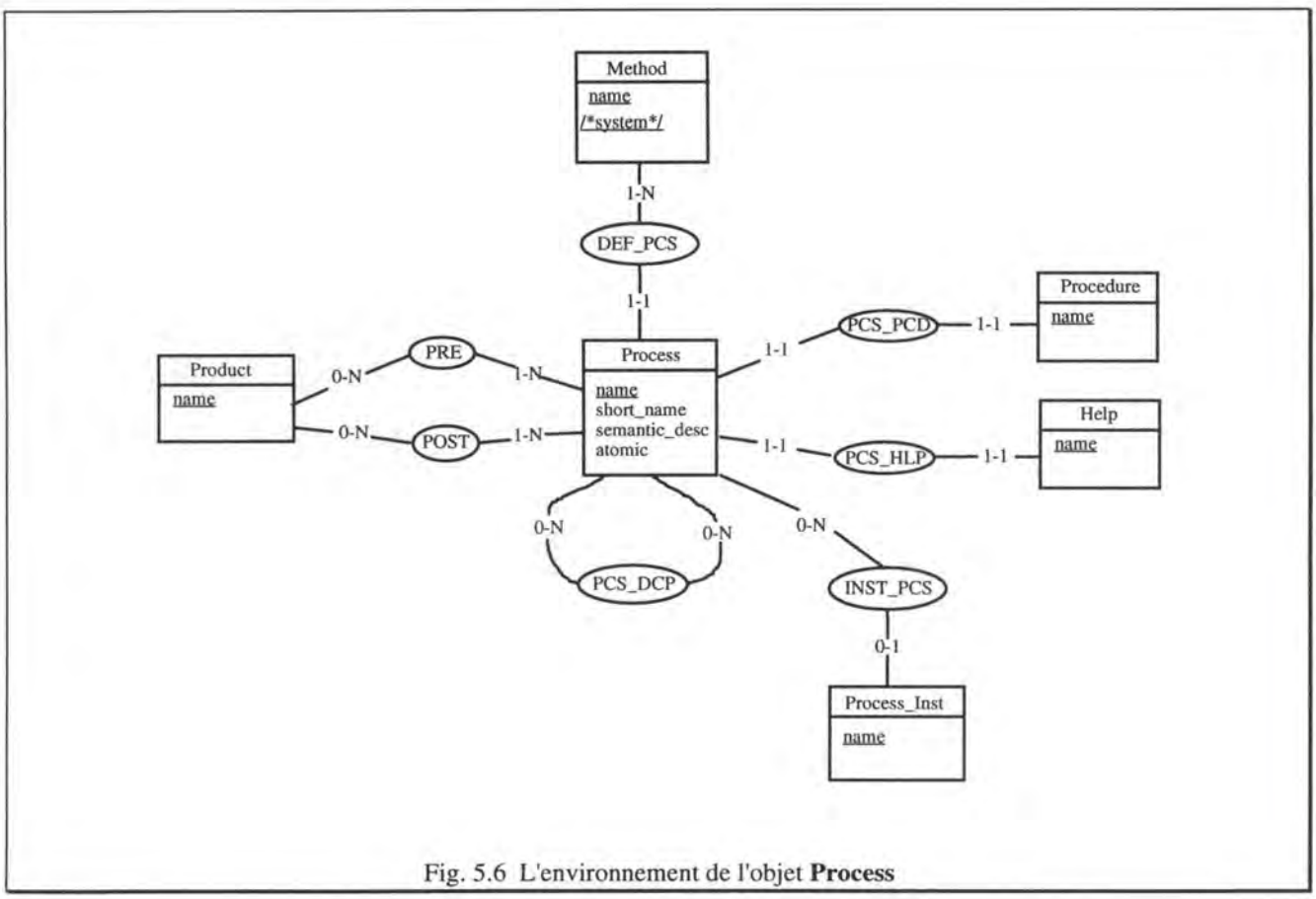


Fig. 5.6 L'environnement de l'objet **Process**

Un processus possède un nom (*name*), un nom court (*short_name*), une description sémantique (*semantic_desc*), et un type booléen (*atomic*) pour indiquer s'il est atomique ou décomposé.

Tout processus doit être défini au sein d'une méthode (via **DEF_PCS**). Il utilise des descriptions de produits comme préconditions (via **PRE**) et comme postconditions (via **POST**), et est réalisé suivant une procédure (via **PCS_PCD**). Un processus peut aussi être décomposé en d'autres processus qu'il utilise dans la spécification de sa procédure. Il est relié aux différents processus qui le composent ainsi qu'à ceux qu'il compose lui-même par la relation "**PCS_DCP**".

Enfin, à chaque processus est associée une aide (via **PCS_HLP**), qui peut être utilisée par le système pour l'assistance à apporter au concepteur dans son développement.

L'association **INST_PCS** permet de relier le processus défini dans la méthode à toutes les instances dont il fait l'objet lors d'une exécution réelle, et qui font partie de l'historique.

La définition d'un processus au sein d'une méthode consiste tout d'abord à spécifier les caractéristiques concernant son nom et sa description informelle. Il faut ensuite faire un lien avec les différents produits qu'il met en oeuvre à la fois comme arguments et résultats, faire éventuellement le lien avec d'autres processus qui pourraient le composer, et spécifier sa partie procédurale. Enfin, on peut encore ajouter l'aide, qui concerne principalement une description plus précise du processus que la description informelle, et sa "participation" dans la méthode.

g. Procedure

L'objet **Procedure** sert à spécifier l'algorithme de réalisation du processus.

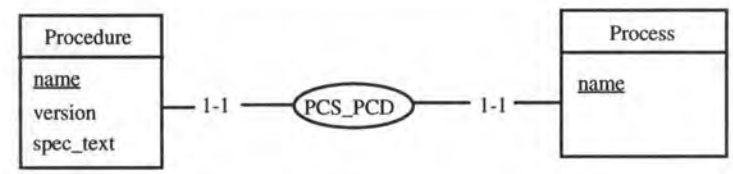


Fig. 5.7 L'environnement de l'objet **Procedure**

Une procédure est définie par un nom (*name*), une version (*version*), et un texte de spécification (*spec_text*) qui doit être interprété par le moteur méthodologique. Ce texte est écrit dans le langage de description que nous avons présenté dans le chapitre 3.

Chaque objet **Procedure** est défini pour un et un seul objet **Process** (via **PCS_PCD**), et inversement tout processus n'est réalisé que par une et une seule procédure.

h. Help

L'objet **Help** sert à spécifier des renseignements qui peuvent être utilisés par l'outil pour proposer à l'utilisateur une aide concernant les processus qu'il utilise.

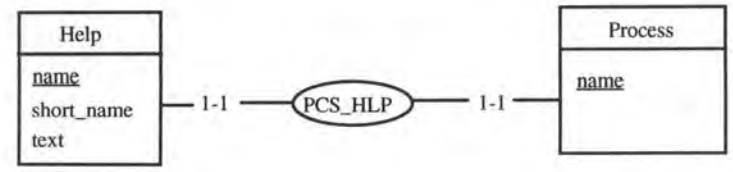


Fig. 5.8 L'environnement de l'objet **Help**

Un objet help est spécifié par un nom (*name*), un nom court (*short_name*), et le texte contenant les renseignements (*text*). Tout objet help n'est défini que pour un et un seul processus, (via **PCS_HLP**), et inversement, à chaque processus ne correspond qu'un et un seul objet help. Cette façon de spécifier l'aide peut paraître assez simple. En fait, à chaque fois que l'utilisateur le demandera, le texte entier apparaîtra et il devra lui-même retrouver les éléments qui l'intéressent. Nous nous limitons à cette représentation pour l'instant, mais rien n'empêche par la suite de définir une structure plus complexe qui, couplée à des fonctionnalités de l'outil, permettra de définir une aide indexée, avec des mots-clés, sujets, ...

5.2.3.2 Les objets de l'historique

a. Process_Inst

L'objet **Process_Inst** permet de spécifier un processus exécuté au cours d'un développement.

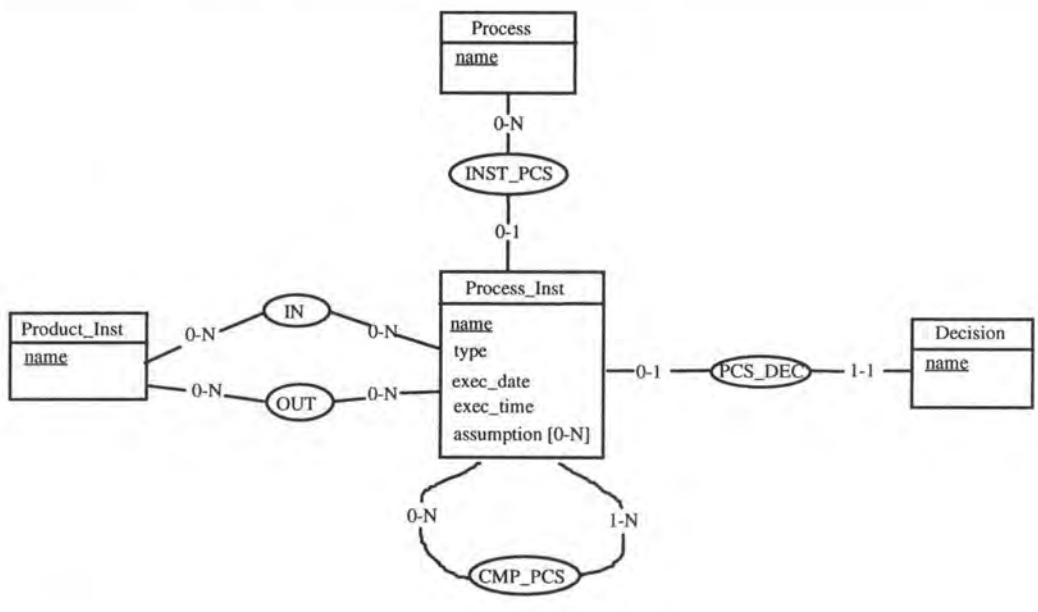


Fig. 5.9 L'environnement de l'objet **Process_Inst**

L'instance d'un processus est caractérisée par un nom (*name*), un type (*type*), une date d'exécution (*exec_date*), une heure d'exécution (*exec_time*), et des hypothèses (*assumption*) dont l'exécution du processus découle. Nous avons choisi de représenter les différentes hypothèses proposées par le concepteur comme des annotations liées à l'exécution des processus. Si un processus est réalisé suivant une hypothèse, tous les processus qui le composent en découlent aussi de manière implicite.

L'instance d'un processus peut représenter l'exécution d'un processus défini par une méthode, mais aussi le fait que le concepteur prend une décision. L'attribut "type" permet de spécifier à quelle catégorie il appartient en prenant pour valeur soit le nom du processus méthodologique, soit la chaîne de caractères "Decision". Si elle est du premier type, elle est reliée via l'association **INST_PCS** à la spécification du processus défini dans la méthode. Dans le deuxième cas, la relation **PCS_DEC** permet de retrouver un objet "decision" dans lequel peuvent être spécifiées des informations complémentaires concernant la décision prise.

Elle peut aussi être reliée à des instances de produits via deux relations différentes **IN** et **OUT**, selon que ces produits sont utilisés comme arguments ou fournis comme résultats par le processus.

Enfin, une instance de processus peut être mise en relation avec plusieurs autres grâce à l'association **CMP_PCS**, qui spécifie que ce processus est réalisé grâce à l'exécution de ces derniers.

b. Product_Inst

L'objet **Product_Inst** permet de spécifier un produit utilisé ou construit au cours d'un développement.

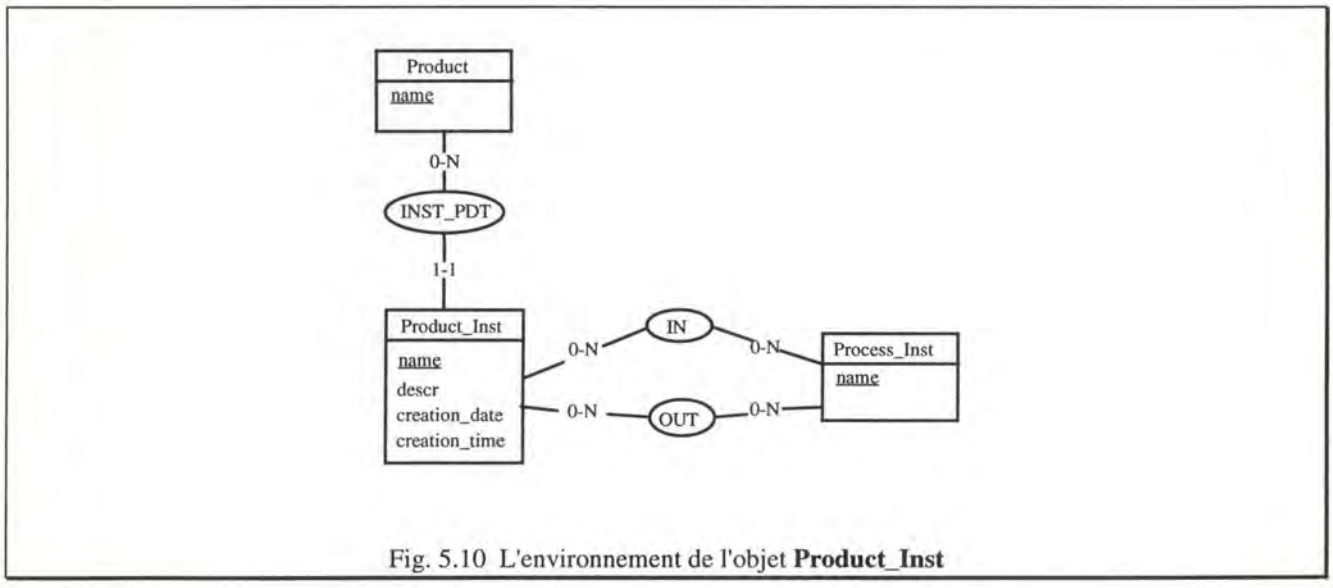


Fig. 5.10 L'environnement de l'objet **Product_Inst**

L'instance d'un produit est caractérisée par un nom (name), une description (descr), une date de création (*creation_date*) et une heure de création (*creation_time*).

Elle est reliée à la spécification d'un produit type défini dans la méthode et dont elle constitue l'instance via la relation **INST_PDT**.

Elle est aussi reliée à l'instance des processus qui l'utilisent comme argument via l'association **IN**, et à l'instance des processus qui l'ont fourni comme résultat via l'association **OUT**.

c. Decision

L'objet **Decision** permet de spécifier les informations concernant une décision prise au cours d'un développement.

Fig. 5.11 L'environnement de l'objet **Decision**

Une décision est caractérisée par un nom (*name*), un objet (*object*) sur lequel elle porte, une description (*descr*) de la décision en elle-même, et une justification (*justif*) qui consiste en des arguments la supportant.

Toute description d'une décision est reliée à l'instance du processus qui constitue la prise de cette décision via l'association **PCS_DEC**. La spécification d'un objet "Decision" est donc faite quand ce type de processus est activé par le concepteur au sein de l'outil.

Conclusion

Nous avons commencé par donner une description générale des outils CASE et des outils de conception de bases de données, et mis en évidence les faiblesses de ceux-ci en ce qui concerne l'aspect méthodologique.

Cette analyse nous a permis, dans la seconde partie de ce chapitre, de définir les concepts nécessaires à l'intégration d'un moteur de méthode au sein de l'atelier DB-MAIN. Après avoir précisé le rôle de celui-ci, à savoir principalement le contrôle de processus, l'aide au concepteur, et la mise à jour d'un historique, nous avons présenté la spécification des différents objets nécessaires à sa réalisation. Dérivés du modèle de représentation proposé précédemment, ces objets sont décrits selon les mêmes conventions que ceux de l'architecture actuelle du repository de l'outil auquel ils viennent s'intégrer. On peut trouver une spécification de leurs attributs et méthodes en annexe.

Conclusion générale

1. Conclusion du travail

L'objectif de ce travail était de rechercher des moyens permettant d'aider à la compréhension et l'amélioration des activités de développement de systèmes.

Après avoir tout d'abord essayé de montrer l'importance de la modélisation des processus, une discipline relativement nouvelle, à la fois d'un point de vue général, et de manière plus spécifique, nous avons présenté les principaux modèles rencontrés dans la littérature dont nous avons extrait les concepts nécessaires et mis à jour les défauts pour la représentation d'un processus.

Nous en avons déduit que le modèle devait prendre en compte principalement deux aspects pour la description d'un processus, à savoir :

1. La description des produits et activités qui le constituent.
2. L'expression des raisonnements suivis pour le conduire.

Nous avons aussi pensé qu'il était important de pouvoir procurer au concepteur des moyens d'assistance, surtout en ce qui concerne l'aspect méthodologique d'un développement. L'amélioration de la conduite d'un processus passe aussi par l'observation de méthodes qui, même si elles sont connues par le développeur ne sont pas toujours respectées de manière rigoureuse. Nous avons donc essayé de définir les éléments permettant de concrétiser cette aide.

Dans le but de réaliser ces objectifs, nous avons décidé de nous concentrer sur le domaine particulier du génie logiciel que constitue celui des bases de données. Etant donné que cette discipline est relativement bien cernée, et possède ses propres concepts, nous avons cru plus intéressant de nous en servir comme base, en espérant que ce que nous y avons développé puisse être adapté au domaine entier du génie logiciel.

Une présentation des méthodes et modèles de représentation utilisés dans ce domaine nous a permis d'en faire une description générale, utile à la compréhension de certains points dans la suite du travail. Nous avons aussi redéfini plus précisément les objectifs en fonction de ce domaine.

L'étude de plusieurs modèles fréquemment rencontrés, et l'analyse des différents concepts mis en exergue dans ceux-ci nous a donné la possibilité de définir un modèle reprenant les caractéristiques jugées importantes pour une approche convenable de modélisation.

En se concentrant à la fois sur les concepts d'activités et de produits, associés à celui de raisonnement, et étant donnée la relative généricité des concepts repris, nous pensons que le modèle peut servir de base à la représentation d'une grande variété de processus et de méthodes. L'avantage de ces caractéristiques est qu'elles autorisent la comparaison de ces processus et méthodes de manière bénéfique, et qu'elles soulignent aussi bien leurs similitudes que leurs différences, et donc leurs avantages et inconvénients respectifs.

Développé dans le contexte des bases de données, le modèle a été facilement utilisé, dans une étude de cas, pour la description d'une méthode qui, bien que particulière et propre à ce contexte, pourrait représenter le profil général d'un certain nombre de méthodes de développement. Nous pensons que le modèle est suffisamment générique et qu'il doit être possible, à quelques modifications près, d'adapter certaines de ses caractéristiques pour qu'il puisse être appliqué de manière uniforme à des développements appartenant à d'autres domaines du génie logiciel.

Le modèle peut aussi être supporté par un environnement automatisé pour certaines fonctionnalités comme par exemple :

- l'enregistrement des occurrences d'activités comme elles sont réalisées,
- la vérification qu'une activité peut bien être réalisée à un moment du développement,
- la proposition de conseils au concepteur,
- l'appel à des outils pour réaliser des activités automatiques,
- etc...

Une approche vers la concrétisation de ces fonctions a consisté à intégrer des concepts de méthodes dans l'architecture courante d'un atelier de génie logiciel de conception et maintenance de bases de données. Bien que cette réalisation ne constitue qu'une faible partie du travail à faire pour inclure tout l'aspect méthodologique au sein d'un tel outil, elle représente une base et une étape importante vers la réalisation de l'assistance à apporter au concepteur.

2. Perspectives futures

Plusieurs efforts fournis dans ce travail pourront peut-être s'avérer bénéfiques pour l'évolution de la discipline. Par exemple, en ce qui concerne l'ensemble des différentes méthodes et outils qui pourraient être utilisés et appliqués à la modélisation-même.

Nous espérons qu'il pourra servir et être vu comme une étape permettant d'aboutir au développement de méthodes plus basées sur la réflexion, à de nouveaux moyens de représentation, ou encore au développement d'outils plus efficaces pour aider à solutionner les problèmes de conception.

Enfin, un autre avantage pourrait encore être gagné en essayant de modéliser d'autres aspects du cycle de vie d'un système. Par exemple, nous avons choisi de ne pas tenir compte dans notre approche du concept d'agent, qui est pourtant repris dans certains modèles cités dans l'état de l'art. Nous ne l'avons pas jugé nécessaire dans le domaine qui nous concernait, mais peut-être ce besoin se fera-t-il plus sentir pour d'autres applications.

Il pourrait aussi être instructif de modéliser différents types de processus, et partir d'une synthèse générale des caractéristiques obtenues pour en déduire un nouveau modèle complet et compréhensible, unique et efficace.

Nous estimons que le modèle développé ici peut déjà être utilisé de manière efficace pour la modélisation des processus de développement, du moins dans une certaine mesure. Nous espérons que ce travail aura permis de faire avancer les connaissances dans le domaine, pourra être utilisé pour les travaux futurs, et constituer une aide dans la maturation du "process modeling" comme une technique d'aide à l'évolution des processus de développement de systèmes logiciels.

Annexes

Annexe A : Diagrammes de représentation des différents modèles de données

A.1 Le modèle relationnel

Ce modèle repose sur la notion de relation construite à partir d'un ensemble de domaines de valeurs. Il représente une base de données comme une collection de tables.

Un schéma relationnel S est un ensemble de relations $S=\{R_1,R_2, \ldots ,R_n\}$. Une instance DB de ce schéma est un ensemble d'instances de relations $DB=\{r_1,r_2, \ldots ,r_n\}$, tel que chaque r_i est une instance de R_i.

La figure ci-dessous montre un exemple de représentation d'une base de données correspondant au modèle relationnel. Cet exemple reprend trois relations, il représente le concept de département regroupant des employés et ayant en charge un certain nombre de projets.

EMPLOYE	Num-emp	Nom-emp	Tel-emp	Dep-Num
	e04578	Smith	2432	4
	d25987	Brown	3318	6
	d45986	Harvey	2469	4
	h45691	Oliva	1847	3

DEPARTEMENT	Num-dep	Nom-dep	Localisation
	4	Recherche	B2
	3	Administration	A1
	6	Maintenance	A2

PROJET	Num-pro	Nom-pro	Localisation	Dep-Num
	20	Reorganisation	A1	3
	3	ProduitX	B2	4
	16	ProduitZ	B1	4
	42	Reseau12	A2	6

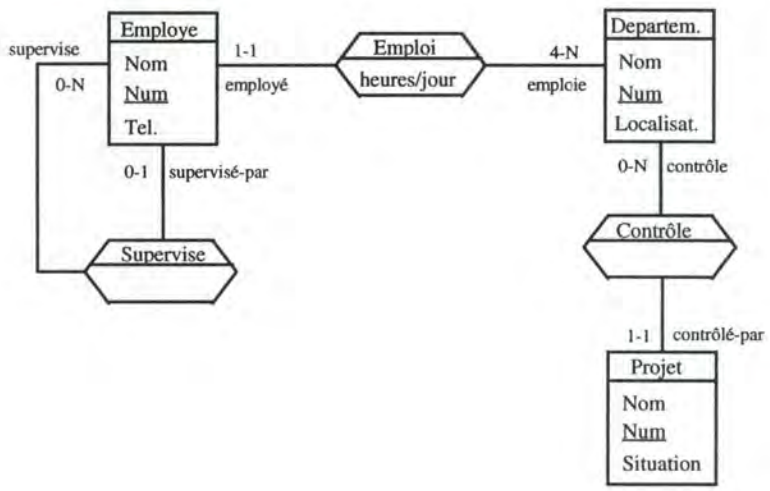
Le nom de la relation est spécifié en majuscules dans la première colonne. Les attributs sont les en-têtes des colonnes qui suivent, et chaque ligne reprenant un élément des différentes colonnes forme un tuple.

Dans certaines versions précédentes du modèle relationnel, les attributs représentant les mêmes concepts du monde réel devaient avoir le même nom dans toutes les relations. Un schéma relationnel peut encore contenir des contraintes d'intégrité, qui sont des propriétés devant être vérifiées par toutes les instances de ce schéma. Elles définissent par exemple les clés candidates de chaque relation, l'interdiction pour certains attributs d'avoir la valeur 'NULL', ...

A.2 Le modèle ERA

Le modèle ERA peut être représenté de manière graphique grâce au diagramme Entités/Associations (ou ERD : Entity Relationship Diagram). Il représente les concepts de type d'entités et type d'associations plutôt que leur instances. Ceci est beaucoup plus utile puisqu'un schéma d'une BD change rarement tandis que l'extension est modifiée fréquemment. De plus, le schéma est plus facile à représenter que l'extension car il est plus petit.

L'exemple de la figure ci-dessous est assez général et montre les différents concepts que nous avons décrits.

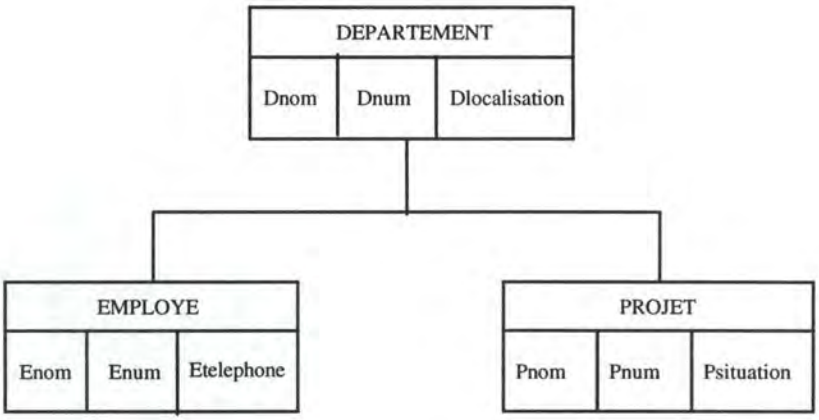


Il reprend l'exemple précédent avec certains éléments supplémentaires. Chaque employé travaille pour un département à raison d'un certain nombre d'heures par jour, et peut superviser d'autres employés. Un département, qui contient au moins quatre personnes, peut contrôler plusieurs projets. Chaque entité (employé, département, projet) est identifiée par son numéro.

Le nom d'un rôle n'est pas obligatoire pour les TA où tous les TE sont distincts. Cependant, le même TE peut participer plus d'une fois dans le même TA avec des rôles différents, il est donc essentiel de nommer ceux-ci pour faire la distinction entre les participations.

A.3 Le modèle hiérarchique

Le modèle hiérarchique représente les données sous forme d'une structure en arbre. Un schéma hiérarchique consiste en un ensemble de schémas hiérarchiques constitués d'un nombre de types de records et de types de RPE (relations parent-enfants).



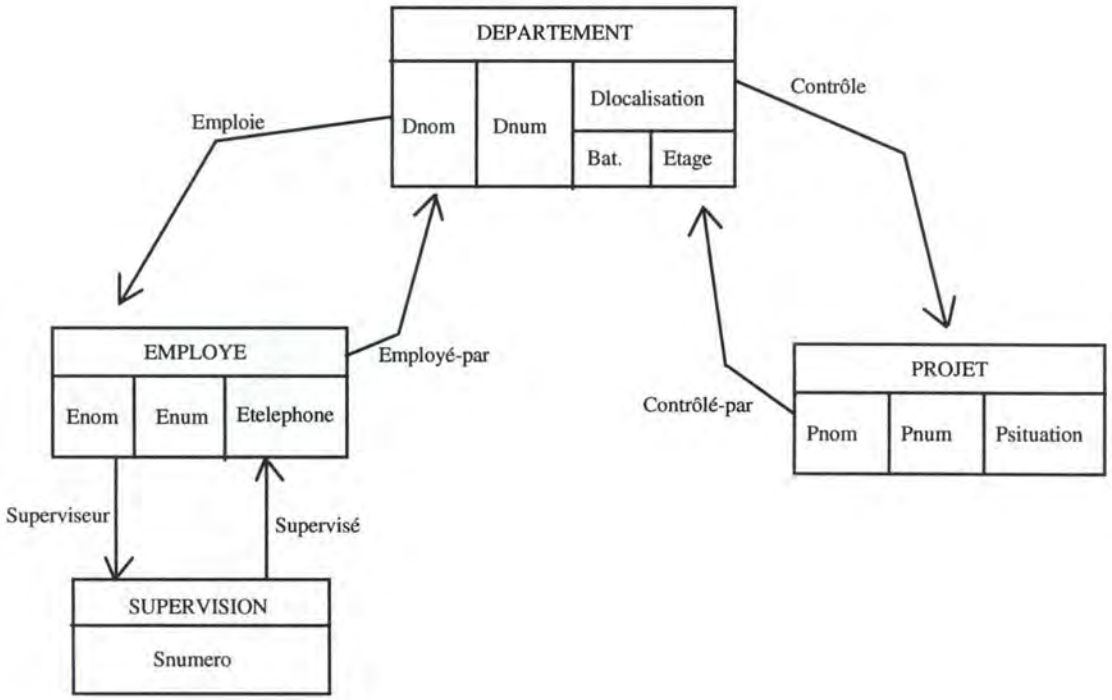
Le diagramme correspondant au modèle hiérarchique utilise des boîtes rectangulaires avec le nom du type de record et des champs, et des arcs pour les RPE.

Notons que les types de RPE n'ont pas de nom dans le modèle, cependant le concepteur leur associe une signification.

Le modèle hiérarchique pose problème pour la représentation de certains types de relations comme les relations M:N, celles où un type de record participe à plus d'une RPE comme enfant, et les relations n-aires avec plus de deux types de records participants. Le problème des relations M:N peut être résolu en autorisant la duplication des instances de types de records enfants. Le deuxième cas peut aussi être représenté de façon similaire. Le troisième pose problème car les RPE sont binaires. Pour ne pas entrer dans les détails, nous dirons que pour le résoudre, l'idée est d'inclure plusieurs schémas hiérarchiques dans le schéma de la BD, et d'avoir des pointeurs d'un schéma vers un autre pour représenter les relations.

A.4 Le modèle en réseau

Le modèle en réseau représente les données sous forme de types de records, ainsi que les relations, limitées au type 1:N, appelées types de set. La figure ci-dessous montre un schéma correspondant à ce modèle pour notre base de données "Département".



Les types de record sont représentés par des rectangles, et les types de set par des arcs dirigés. Les relations de type M:N sont représentées par deux arcs entre les types de records, les types de relations récursives sont représentées par deux arcs et l'ajout d'un type de record.

A.5 Le modèle structurel

La représentation d'un schéma selon le modèle structurel est assez complexe, et nous n'en avons pas trouvé d'exemple dans la littérature. Pour ne pas commettre d'erreurs en improvisant un schéma selon les définitions trouvées, nous nous contenterons de montrer de manière générale comment sont représentés les différents types de relations que nous avons présentées dans le deuxième chapitre.

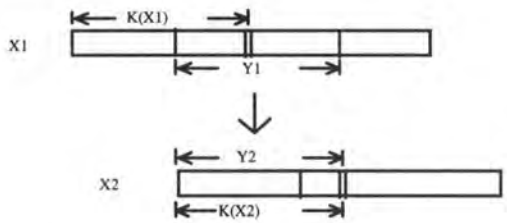
Une connexion définie entre deux schémas de relations X1 et X2, et dont les instances existent entre deux tuples, est établie par deux ensembles d'attributs Y1 et Y2 tels que :

- $Y1 \subseteq X1$
- $Y2 \subseteq X2$
- $Dom(Y1) = Dom(Y2)$

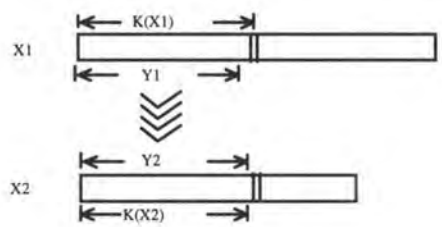
On dit alors que X1 est connecté à X2 au travers de (Y1,Y2). Deux tuples sont connectés si les valeurs des attributs de connexion sont les mêmes dans les deux. Le modèle structurel utilise trois types de connexions :

a) les références propriétaires
b) les références identités
c) les références directes

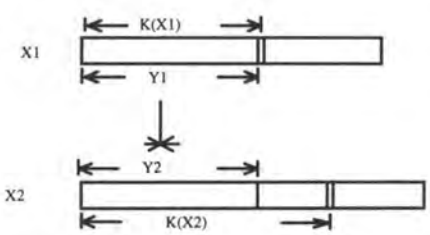
La figure suivante montre comment elles sont représentées graphiquement. K représente l'ensemble des attributs identifiants de la relation, ils sont séparés des attributs dépendants par "ll".



(a) Référence directe <X1,X2>

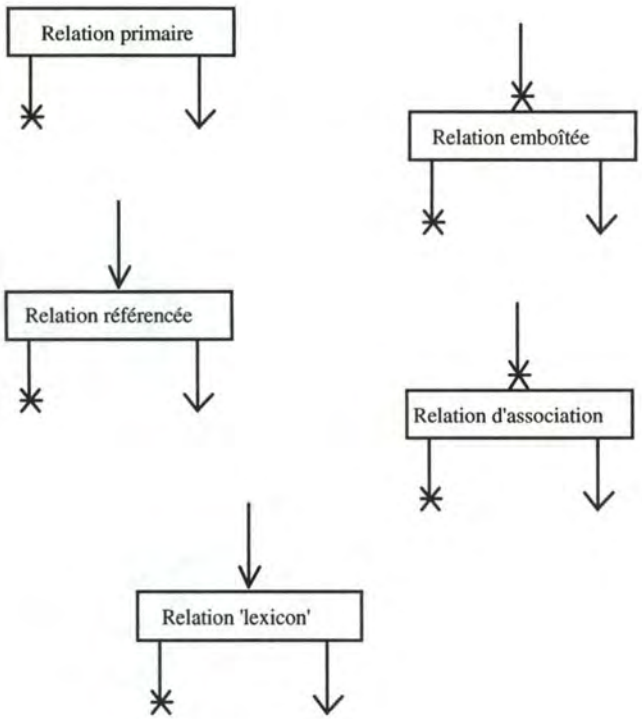


(b) Référence identité <X1,X2>



(c) Référence propriétaire <X1,X2>

Les connexions sont définies entre les schémas de différents types de relations. Ceux-ci utilisent les constructions présentées dans la figure suivante.

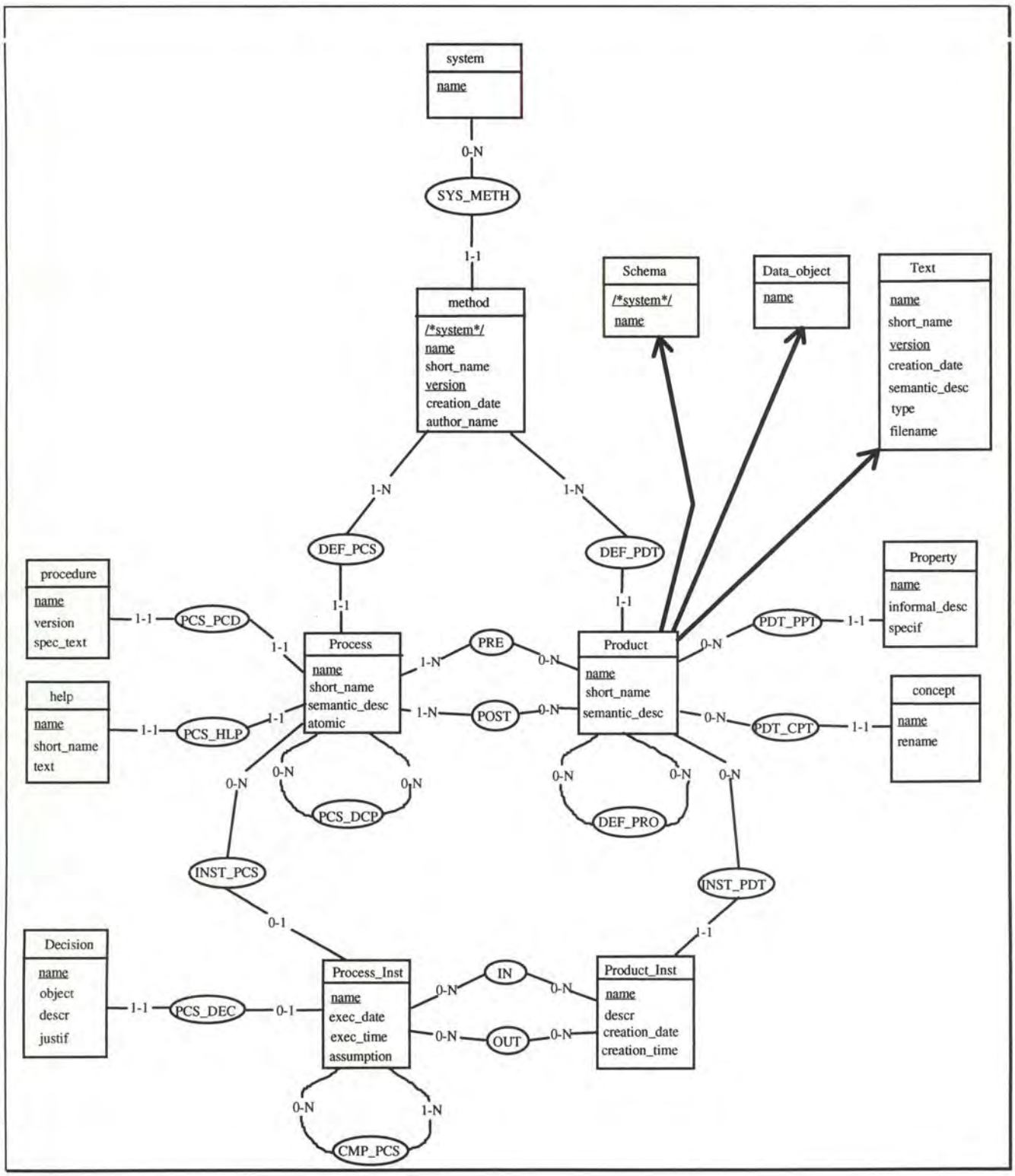


On peut voir qu'il est alors possible de définir des relations M:N entre A et B de quatre manière différentes :

1. Une association propriétaire A, B;
2. Une relation qui référence deux relations référencées A et B;
3. Une relation emboîtée dont la relation A est propriétaire, et qui référence une relation B;
4. Une relation emboîtée dont la relation B est propriétaire, et qui référence une relation A.

Annexe B : Spécification de l'architecture du moteur méthodologique

B.1 Architecture générale du moteur de méthode



B.2 Spécification des objets

1. L'objet Method

Attributs

Nom de l'attribut	Type	Longueur	Répétitivité
name	char	L_NAME	1-1
short_name	char	L_SNAME	0-1
version	char	L_VERSION	1-1
creation_date	char	L_DATE	1-1
author_name	char	L_NAME	0-1

Méthodes

get_name : renvoie le pointeur vers le nom (n) de la méthode (this);
Arg : --;
Res : char *n;

get_short_name : renvoie le pointeur vers le nom court (sn) de la méthode (this);
Arg : --;
Res : char *sn;

get_version : renvoie le pointeur vers la version (v) de la méthode (this);
Arg : --;
Res : char *v;

get_creation_date : renvoie le pointeur vers la date de création (cd) de la méthode (this);
Arg : --;
Res : char *cd;

get_author_name : renvoie le pointeur vers le nom de l'auteur (an) de la méthode (this);
Arg : --;
Res : char *an;

update_name : met à jour le nom (n) de la méthode (this);
Arg : char *n;
Res : --;

update_short_name : met à jour le nom court (sn) de la méthode (this);
Arg : char *sn;
Res : --;

update_version : met à jour la version (v) de la méthode (this);
Arg : char *v;
Res : --;

update_creation_date : met à jour la date de création (cd) de la méthode (this);
Arg : char *cd;
Res : --;

update_autor_name : met à jour le nom de l'auteur (an) de la méthode (this);
Arg : char *an;
Res : --;

get_first_product : renvoie le premier produit (pdt) de la méthode (this);
Arg : --;
Res : product *pdt;

get_next_product : renvoie le produit (n_pdt) suivant un autre produit (pdt) de la méthode (this);
Arg : product *pdt;
Res : product *n_pdt

add_first_product : ajoute un produit (pdt) en première position de la liste des produits de la méthode (this). Le produit créé sera associé à la méthode, ainsi qu'à un autre produit à partir duquel il pourrait être défini;
Arg : product *pdt;
Res : --;

get_first_process : renvoie le premier processus (pcs) de la méthode (this)
Arg : --;
Res : process *pcs;

get_next_process : renvoie le processus (n_pcs) suivant un autre processus (pcs) de la méthode (this);
Arg : process *pcs;
Res : process *n_pcs

add_first_process : ajoute un processus (pcs) en première position de la liste des produits de la méthode (this). Le processus créé sera associé à la méthode, ainsi qu'à un autre processus dont il pourrait constituer un composant.
Arg : process *pcs
Res : --;

2. L'objet Product

Attributs

Nom de l'attribut	Type	Longueur	Répétitivité
name	char	L_NAME	1-1
short_name	char	L_SNAME	0-1
semantic_desc	text		0-1

Méthodes

get_name : renvoie le pointeur vers le nom (n) du produit (this);
Arg : --;
Res : char *n;

get_short_name : renvoie le pointeur vers le nom court (sn) du produit (this);
Arg : --;
Res : char *sn;

get_semantic_desc : renvoie le pointeur vers la description (sd) du produit (this);
Arg : --;
Res : char *sd;

update_name : met à jour le nom (n) du produit (this);
Arg : char *n;
Res : --;

update_short_name : met à jour le nom court (sn) du produit (this);
Arg : char *sn;
Res : --;

update_semantic_desc : met à jour la description (sd) du produit (this);
Arg : char *sd;
Res : --;

get_first_concept : renvoie le premier concept (cpt) du produit (this);
Arg : --;
Res : concept *cpt;

get_next_concept : renvoie le concept (n_cpt) suivant un autre concept (cpt) du produit (this);
Arg : concept *cpt;
Res : concept *n_cpt

add_first_concept : ajoute un concept (cpt) en première position de la liste des concepts du produit (this). Le concept créé sera associé au produit;
Arg : concept *cpt;
Res : --;

get_first_property : renvoie la première propriété (ppt) du produit (this);
Arg : --;
Res : property *pcs;

get_next_property : renvoie la propriété (n_ppt) suivant une autre propriété (ppt) du produit (this);
Arg : property *ppt;
Res : property *n_pcs;

add_first_property : ajoute une propriété (ppt) en première position de la liste des propriétés du produit (this). La propriété créée sera associée au produit;
Arg : property *ppt;
Res : --;

3. L'objet Text

Attributs

Nom de l'attribut	Type	Longueur	Répétitivité
name	char	L_NAME	1-1
short_name	char	L_SNAME	0-1
version	char	L_VERSION	1-1
creation_date	char	L_DATE	1-1
semantic_desc	text		
type	char	L_NAME	0-1
filename	char	L_FILENAME	1-1

Méthodes

get_name : renvoie le pointeur vers le nom (n) du texte (this);
Arg : --;
Res : char *n;

get_short_name : renvoie le pointeur vers le nom court (sn) du texte (this);
Arg : --;
Res : char *sn;

get_version : renvoie le pointeur vers la version (v) du texte (this);
Arg : --;
Res : char *v;

get_creation_date : renvoie le pointeur vers la date de création (cd) du texte (this);
Arg : --;
Res : char *cd;

get_semantic_desc : renvoie le pointeur vers la description (d) du texte (this);
Arg : --;
Res : char *d;

get_type : renvoie le pointeur vers le type (t) du texte (this);
Arg : --;
Res : char *t;

get_filename : renvoie le pointeur vers le nom du fichier (nf) contenant le texte (this);
Arg : --;
Res : char *nf;

update_name : met à jour le nom (n) du texte (this);
Arg : char *n;
Res : --;

update_short_name : met à jour le nom court (sn) du texte (this);
Arg : char *sn;
Res : --;

update_version : met à jour la version (v) du texte (this);
Arg : char *v;
Res : --;

update_semantic_desc : met à jour la description (d) du texte (this);
Arg : char *d;
Res : --;

update_type : met à jour le type (t) du texte (this);
Arg : char *t;
Res : --;

update_filename : met à jour le nom de fichier (nf) du texte (this);
Arg : char *nf;
Res : --;

4. L'objet Property

Attributs

Nom de l'attribut	Type	Longueur	Répétitivité
name	char	L_NAME	1-1
informal_desc	text		0-1
specif	text		1-1

Méthodes

get_name : renvoie le pointeur vers le nom (n) de la propriété (this);
Arg : --;
Res : char *n;

get_informal_desc : renvoie le pointeur vers la description (id) de la propriété (this);
Arg : --;
Res : char *id;

get_specif : renvoie le pointeur vers la spécification (s) de la propriété (this);
Arg : --;
Res : char *s;

update_name : met à jour le nom (n) de la propriété (this);
Arg : char *n;
Res : --;

update_informal_desc : met à jour la description (id) de la propriété (this);
Arg : char *id;
Res : --;

update_specif : met à jour la spécification (s) de la propriété (this);
Arg : char *s;
Res : --;

5. L'objet Concept

Attributs

Nom de l'attribut	Type	Longueur	Répétitivité
name	char	L_NAME	1-1
rename	char	L_NAME	0-1

Méthodes

get_name : renvoie le pointeur vers le nom (n) du concept (this);
Arg : --;
Res : char *n;

get_new_name : renvoie le pointeur vers le nouveau nom (nn) du concept (this);
Arg : --;
Res : char *nn;

update_name : met à jour le nom (n) du concept (this);
Arg : char *n;
Res : --;

update_new_name : met à jour le nouveau nom (nn) du concept (this);
Arg : char *nn;
Res : --;

6. L'objet Process

Attributs

Nom de l'attribut	Type	Longueur	Répétitivité
name	char	L_NAME	1-1
short_name	char	L_SNAME	0-1
semantic_desc	text		0-1
atomic	bool		1-1

Méthodes

get_name : renvoie le pointeur vers le nom (n) du processus (this);
Arg : --;
Res : char *n;

get_short_name : renvoie le pointeur vers le nom court (sn) du processus (this);
Arg : --;
Res : char *sn;

get_semantic_desc : renvoie le pointeur vers la description (sd) du processus (this);
Arg : --;
Res : char *sd;

get_atomic : renvoie le pointeur vers le type (t) du processus (this);
Arg : --;
Res : char *t;

update_name : met à jour le nom (n) du processus (this);
Arg : char *n;
Res : --;

update_short_name : met à jour le nom court (sn) du processus (this);
Arg : char *sn;
Res : --;

update_semantic_desc : met à jour la description (sd) du processus (this);
Arg : char *sd;
Res : --;

update_atomic : met à jour le type (t) du processus (this);
Arg : char *t;
Res : --;

get_procedure : renvoie la procedure (pcd) du processus (this);
Arg : --;
Res : procedure *pcd;

get_help : renvoie le texte d'aide (hlp) du processus (this);
Arg : --;
Res : help *hlp;

get_first_pre_product : renvoie le premier produit (pdt) input du processus (this);
Arg : --;
Res : product *pdt;

get_next_pre_product : renvoie le produit (n_pdt) suivant un autre produit (pdt) input du processus (this);
Arg : product *pdt;
Res : product *n_pdt;

add_first_pre_product : ajoute un produit (pdt) en première position de la liste des produits input du processus (this). Le produit sera associé au processus;
Arg : product *pdt;
Res : --;

get_first_post_product : renvoie le premier produit (pdt) output du processus (this);
Arg : --;
Res : product *pdt;

get_next_post_product : renvoie le produit (n_pdt) suivant un autre produit (pdt) output du processus (this);
Arg : product *pdt;
Res : product *n_pdt;

add_first_post_product : ajoute un produit (pdt) en première position de la liste des produits output du processus (this). Le produit sera associé au processus;
Arg : product *pdt;
Res : --;

7. L'objet Procedure

Attributs

Nom de l'attribut	Type	Longueur	Répétitivité
name	char	L_NAME	1-1
version	char	L_VERSION	1-1
spec_text	text		1-1

Méthodes

get_name : renvoie le pointeur vers le nom (n) de la procedure (this);
Arg : --;
Res : char *n;

get_version : renvoie le pointeur vers la version (v) de la procédure (this);
Arg : --;
Res : char *v;

get_specif : renvoie le pointeur vers le texte de spécification (st) de la procédure (this);
Arg : --;
Res : char *st;

update_name : met à jour le nom (n) de la procédure (this);
Arg : char *n;
Res : --;

update_version : met à jour la version (v) de la procédure (this);
Arg : char *v;
Res : --;

update_specif : met à jour le texte de spécification (st) de la procédure (this);
Arg : char *st;
Res : --;

8. L'objet Help

Attributs

Nom de l'attribut	Type	Longueur	Répétitivité
name	char	L_NAME	1-1
short_name	char	L_SNAME	0-1
text	text		1-1

Méthodes

get_name : renvoie le pointeur vers le nom (n) de l'aide (this);
Arg : --;
Res : char *n;

get_short_name : renvoie le pointeur vers le nom court (sn) de l'aide (this);
Arg : --;
Res : char *sn;

get_text : renvoie le pointeur vers le texte (t) de l'aide (this);
Arg : --;
Res : char *t;

update_name : met à jour le nom (n) de l'aide (this);
Arg : char *n;
Res : --;

update_short_name : met à jour le nom court (sn) de l'aide (this);
Arg : char *sn;
Res : --;

update_text : met à jour le texte (t) de l'aide (this);
Arg : char *t;
Res : --;

9. L'objet Process_Inst

Attributs

Nom de l'attribut	Type	Longueur	Répétitivité
name	char	L_NAME	1-1
exec_date	char	L_DATE	1-1
exec_time	char	L_TIME	1-1
assumption	text		0-1

Méthodes

get_name : renvoie le pointeur vers le nom (n) du processus (this);
Arg : --;
Res : char *n;

get_exec_date : renvoie le pointeur vers la date d'exécution (d) du processus (this);
Arg : --;
Res : char *d;

get_exec_time : renvoie le pointeur vers l'heure d'exécution (h) du processus (this);
Arg : --;
Res : char *h;

get_assumption : renvoie le pointeur vers la description de l'hypothèse (hy) impliquant le processus (this);
Arg : --;
Res : char *hy;

update_name : met à jour le nom (n) du processus (this);
Arg : char *n;
Res : --;

update_assumption : met à jour la description de l'hypothèse (hy) impliquant le processus (this);
Arg : char *hy;
Res : --;

get_decision : renvoie la décision (d) constituée par le processus (this);
Arg : --;
Res : decision *d;

get_first_in_product : renvoie le premier produit (pdt) input du processus (this);
Arg : --;
Res : product_inst *pdt;

get_next_in_product : renvoie le produit (n_pdt) suivant un autre produit (pdt) input du processus (this);
Arg : product_inst *pdt;
Res : product_inst *n_pdt;

add_first_in_product : ajoute un produit (pdt) en première position de la liste des produits input du processus (this). Le produit sera associé au processus;
Arg : product_inst *pdt;
Res : --;

get_first_out_product : renvoie le premier produit (pdt) output du processus (this);
Arg : --;
Res : product_inst *pdt;

get_next_out_product : renvoie le produit (n_pdt) suivant un autre produit (pdt) output du processus (this);
Arg : product_inst *pdt;
Res : product_inst *n_pdt;

add_first_out_product : ajoute un produit (pdt) en première position de la liste des produits output du processus (this). Le produit sera associé au processus;
Arg : product_inst *pdt;
Res : --;

get_first_comp_process : renvoie le premier processus (pcs) composant le processus (this);
Arg : --;
Res : process_inst *pcs;

get_next_comp_process : renvoie le processus (n_pcs) suivant un autre processus (pcs) composant celui-ci (this);
Arg : product_inst *pcs;
Res : product_inst *n_pcs;

add_first_comp_process : ajoute un nouveau processus (pcs) en première position de la liste des processus composant celui-ci (this). Le nouveau sera associé au processus;
Arg : process_inst *pcs;
Res : --;

10. L'objet Decision

Attributs

Nom de l'attribut	Type	Longueur	Répétitivité
name	char	L_NAME	1-1
object	text		1-1
descr	text		1-1
justif	text		1-1

Méthodes

get_name : renvoie le pointeur vers le nom (n) de la décision (this);
Arg : --;
Res : char *n;

get_object : renvoie le pointeur vers l'objet (o) de la décision (this);
Arg : --;
Res : char *o;

get_descr : renvoie le pointeur vers la description (d) de la décision (this);
Arg : --;
Res : char *d;

get_justif : renvoie le pointeur vers la justification (j) de la décision (this);
Arg : --;
Res : char *j;

update_name : met à jour le nom (n) de la décision (this);
Arg : char *n;
Res : --;

update_object : met à jour la description de l'objet (do) de la décision (this);
Arg : char *do;
Res : --;

update_descr : met à jour la description (d) de la décision (this);
Arg : char *d;
Res : --;

update_justif : met à jour le texte de justification (jt) de la décision (this);
Arg : char *jt;
Res : --;

11. L'objet Product_Inst

Attributs

Nom de l'attribut	Type	Longueur	Répétitivité
name	char	L_NAME	1-1
descr	text		0-1
creation_date	char	L_DATE	1-1
creation_time	char	L_TIME	1-1

Méthodes

get_name : renvoie le pointeur vers le nom (n) du produit (this);
Arg : --;
Res : char *n;

get_descr : renvoie le pointeur vers la description (d) du produit (this);
Arg : --;
Res : char *d;

get_creation_date : renvoie le pointeur vers la date de création (cd) du produit (this);
Arg : --;
Res : char *cd;

get_creation_time : renvoie le pointeur vers l'heure de création (h) du produit (this);
Arg : --;
Res : char *h;

update_name : met à jour le nom (n) du produit (this);
Arg : char *n;
Res : --;

update_descr : met à jour la description (d) du produit (this);
Arg : char *d;
Res : --;

get_first_process_in : renvoie le premier processus (pcs) utilisant le produit (this) en input;
Arg : --;
Res : process_inst *pcs;

get_next_process_in : renvoie le processus (n_pcs) suivant un autre processus (pcs) utilisant le produit (this) en input;
Arg : process_inst *pcs;
Res : product_inst *n_pcs;

get_first_process_out : renvoie le premier processus (pcs) utilisant le produit (this) en output;
Arg : --;
Res : process_inst *pcs;

get_next_process_out : renvoie le processus (n_pcs) suivant un autre processus (pcs) utilisant le produit (this) en output;
Arg : process_inst *pcs;
Res : product_inst *n_pcs

Bibliographie

[**Bach64**] Bachman C.W., Williams S. "A general purpose programming system for random access memories" Proc. of the Fall Joint Computer Conference, AFIPS, 26 1964.

[**Bach77**] Bachman, C.W., Daya M. "The role concept in data models" Proc. of the international conference on very large databases, 1977.

[**Basi75**] Basili V.R., Turner A.J. "Iterative Enhancement : a practical technique for software development" IEEE Transactions on software engineering, vol. SE-1, n°4, p.390-396, 1975.

[**Bat80**] Batini C. "Top-down design in the Entity-Relationship schema" International Conference on ER Approach, 1980.

[**Bern76**] P.A. Bernstein "Synthesizing third normal form relations from functional dependencies" Transactions on database systems 1,4 December 1976.

[**Bjo82**] Bjorner D., Lovengren H. "Formalization of database systems and formal definition of IMS" Proc. of the international conference on very large databases, 1982.

[**Boehm86**] Boehm B.W. "A spiral model of software development and enhancement" Proc. of an international workshop on the software process and software environments, Coto do Caza, California, March 1985.

[**Bra83**] R.J. Brachman "What IS-A is and isn't : an analysis of taxonomic links in semantic networks" COMPUTER, 16(10) : 67-73, 1983.

[**Chen76**] P.P. Chen "The Entity-Relationship Model : Toward a unified view of data" ACM TODS 1, March 1976.

[**Chen77**] P.P. Chen "The Entity-Relationship Approach to logical database design" Q.E.D. Information Sciences, Wellesley, 1977.

[**Chu78**] Chu Y. "Introducing a software design language" Structured analysis and design, Infotech state of the art report, Maidenhead, UK, 1978.

[**Codd70**] E. Codd "A relational model for large shared data banks" Communications of the ACM, June 1970.

[**Codd71**] E. Codd "Further normalization of the database relational model" Courant Computer Science Symposia 6, Data Base System, Prentice-Hall, New York, May 1971.

[**Cur86**] Currit P.A., Dyer M., Mills H.D. "Certifying the reliability of software" IEEE Transactions on Software Engineering, 12(1), 3-11, 1986.

[**Curt78**] Curtice P.M., Jones P.E. "Key steps in the logical design of data bases" Proceedings NYU Symposium on database design, 51-66, 1978.

[**Curt92**] Curtis B., Kellner M.I., Over J. "Process Modeling" Com. of the ACM, September 1992, Vol. 35 n°9.

[**DBMAIN93**] "Le projet DBMAIN, maintenance et évolution des fichiers et bases de données" Facultés Universitaires N-D de la Paix, Namur, 6 juin 1993.

[**DBMAIN94**] V. Englebert, J. Henrard, J-M. Hick, D. Roland "Description du méta-schéma dans une perspective orienté-objet de l'atelier logiciel DB-MAIN" Rapport technique 8 avril 1994.

[**DBTG71**] Report of the CODASYL Data Base Task Group ACM, April 1971.

[**Dodd69**] Dodd G. "Elements of data management systems" ACM Computing Surveys, 1:2, June 1969.

[**Dows87**] Dowson M. "ISTAR - An integrated project support environment" Proc. of the ACM Software Engineering Symposium on Practical Software Development Environments, Palo Alto, California, December 1986.

[**Elm89**] Elmasri, Navathe "Fundamentals of database systems" The Benjamin / Cummings Publishnig Company, 1989.

[**Fagi77**] R. Fagin "Multivalued dependencies and new normal form for relational databases" Transactions on database systems 2,3, September 1977.

[**Fin90**] Finkelstein A., Kramer J., Goedicke M. "ViewPoint Oriented Software development" Proc. conference "Le génie logiciel et ses applications", Toulouse, p.337-351, 1990.

[**Flo80**] Flory A. "An approach to designing an Entity-Relationship schema" International Conference on ER Approach, 1980.

[**Fou88**] Fourth International Software Process Workshop : Representing and Enacting the Software process, Moretonhampstead, UK, May 11-13, 1988.

[**Hai92a**] J-L Hainaut, M. Cadelli, B. Decuyper, O. Marchand "Database case tool architecture : principles for flexible design strategies" CAiSE-92, Manchester, May 1992.

[**Hai92b**] J-L Hainaut, M. Cadelli, B. Decuyper, O. Marchand "TRAMIS : a transformation-based database CASE tool" Proceedings of the 5th ICSE and its applications, Toulouse, December 1992.

[**Hai92c**] J-L Hainaut "Design Process Modeling - Triggering Structure", 24 Septembre 1992.

[**Hai94**] J-L. Hainaut, V. Englebert, J. Henrard, J-M. Hick, D. Roland "Database evolution : the DBMAIN approach", 29 avril 1994.

[**Ham81**] Hammer M., McLeod D. "Database description with SDM : A semantic data model" ACM Transactions on Database Systems, 6:3, September 1983.

[**Har87**] D. Harel "Statecharts : a visual formalism for complex systems" Sciences of the programming, 8, 1987, 231-274.

[**Har90**] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman "STATEMATE : a working environment for the development of complex reactive systems" IEEE Trans. on Software Engineering, Vol. 16, 1990, 403-414.

[**Hum88**] Humphrey W.J., Kellner M.I. "Software Process Modeling : Principles of Entity Process Models" Proceedings of the 11th International Conference on Software Engineering, IEEE, May 1989.

[**Jar92**] Jarke M., Mylopoulos J., Schmidt J.W., Vassiliov Y. "DAIDA - an environment for evolving information systems" ACM Transactions on Information Systems, vol. 10, n° 1, 1992.

[**Kahn76**] Kahn B.K. "A method for describing information required by the database design process" Proc. of Sigmod International Conference on management of data, June 1976, ACM, New York.

[**Leer76**] Van Leer P. "Top down development using a program design language" IBM systems journal, 15(2), 155-170, 1976.

[**Lehm85**] Lehman M.M. "Approach to a disciplined development process : the ISTAR integrated project support development" Proc. of an International workshop on the Software Process and Software Environments, Coto do Caza, California, March 1985.

[**Lehm87**] Lehman M.M. "Process models, process programs, programming support" In Proceedings of the 9th International Conference on Software Engineering, IEEE, 1987.

[**Linger79**] Linger R.C., Mills H.D., Witt B.I. "Structured programming - Theory and practice" Reading, Mass: Addison-Wesley, 1979.

[**McGee77**] McGee W. "The Information management system IMS/VS, Part I: general structure and operation" IBM Systems Journal, 16:2, June 1977.

[**Mylo78**] Mylopoulos J., Bernstein P.A., Wong H.K.T. "A language facility for designing interactive database intensive applications" Proc. ACM SIGMOD, p.17, 1978.

[**NBS85**] National Bureau of Standards "Guide on logical database design" Special publication 500-122, 1985.

[**Ost87**] Osterweil L. "Software processes are software too" In proceedings of the 9th International Conference on Software Engineering, IEEE, 1987.

[**Pal78**] Palmer J. "Practicalities in applying a formal methodology to data analysis" Proc. NYU Symposium on database design, 67-84, 1978.

[**Peu91**] Peugeot G., Franckson M. "Specification of the object and process modelling language" ESF Report N° D122-OPML-1-0, 1991.

[**Phil89**] Phillips R.W. "State Change Architecture : A protocol for executable process models" In proceedings of the 22nd Annual Hawaii International Conference on System Sciences, Vol. II, IEEE, 1989.

[**PoBr88**] Colin Potts, Glenn Bruns "Recording the reasons for design decisions" MCC Software Technology Program, ICSE, 1988.

[**Pot89**] Potts C. "A generic model for representing design methods" Proc of the 11th ICSE, 1989.

[**Ridd78**] Riddle W.E., Wileden J.C., Sayler J.M., Setgal A.R., Stavely A.M. "Behavioral modeling during software design" IEEE Transactions on Software Engineering, Vol. SE-4, n°4, 1978.

[**Rol93**] Colette Rolland "Modeling the requirements engineering process" 3rd European - Japanese seminars on information modelling and knowledge bases, May 1993, Budapest.

[**Rom89**] Rombach H.D., Leo M. "Software Process & Product Specifications : A basis for generating customized Software Engineering Information Bases" In Proceedings of the 22nd Annual Hawaii International Conference on System Sciences, Vol. II, IEEE, 1989.

[**Royce70**] Royce W.W. "Managing the development of large software systems" Proc. IEEE, WESCON, 1970.

[**Sch79**] Scheuermann P., Schiffner G., Weber H. "Abstraction capabilities and invariant properties modeling with the Entity-Relationship approach" Proc. of the International Conference on ER approach, 1979.

[**Schk78**] Schkolnick M. "A survey of physical database design methodology and techniques" Proc. of the International Conference on Very Large Databases, 1978.

[**Schm75**] Schmidt H.A., Swenson J.R. "On the semantics of the relational data model" Proc. ACM Sigmod Conference, p.9-36, May 1976.

[**Siau92**] K L Siau, H C Chan, K P Tan "A CASE tool for conceptual database design" Information and Software Technology, Vol.34, n°12, 12 December 1992

[**Smith77**] Smith J., Smith D. "Database abstraction : aggregation and generalization" ACM Transactions on Database Design, 2:2, June 1977.

[**Smith78**] Smith J.M., Smith D.C.P. "Principles of database conceptual design" Proceedings NYU Symposium on database design, 35-49, 1978.

[**Souq93**] Souquières J. "Aides au développement de spécifications" Thèse d'état, janvier 1993, Nancy, France.

[**Tard80**] Tardieu H., Pascot D., Nanci D., Heckenroth H. "A method, a formalism and tools for data base design (three years of experimental practice)" International Conference on ER Approach, 1980.

[**Tard83**] Tardieu H. "La méthode Merise, principes et outils" Editions d'organisation, 1983.

[**Tard85**] Tardieu H. "La méthode Merise, démarche et pratiques" Editions d'organisation, 1985.

[**Teo86**] Teorey T., Yang D., Fry J. "A logical design methodology for relational databases using the Extended Entity-Relationship model" ACM Computing Surveys, 18:2, June 1986.

[**Tsao80**] Tsao J.H. "Enterprise schema - An approach to IMS logical database design" International Conference on ER Approach, 1980.

[**Wass87**] Wasserman A.I., Pircher P.A. "A graphical, extensible integrated environnement software development" Proc. of 2nd Symposium on practical software development environments, SIGPlan Notices, Vol. 22, 1, Jan. 1987, 131-142.

[**Web85**] "Webster's Ninth New Collegiate Dictionary", Merriam-Webster, Inc., Springfield, MA, 1985.

[**Whang82**] Whang K., Wiederhold G., Sagalowicz D. "Physical design of network model databases using the property of separability" Proc. of the International Conference on Very Large Databases, 1982.

[**Wie79**] Wiederhold G., Elmasri R. "A structural model for database systems" TR-CS-79-722, Computer Science Dept., Stanford University, February 1979.

[**Wile86**] Wileden J.C. "This is IT : a metamodel of the software process" Proceedings of an international workshop on the software process and software environments, March 1985, California.

[**Will88**] Williams L.G. "Software Process Modeling : A behavorial approach" Proc. of the 10th International Conference on Software Engineering, April 11-15, Singapore 1988.

[**Wong80**] Wong E., Katz R.H. "Logical design and schema conversion for relational and DBTG databases" International Conference on ER Approach, 1980.

[**Yao78**] Yao S.B., Navathe S.B., Weldon J.L. "An integrated approach to logical database design" NYU Symposium on database design, May 1978.

[**Yone93**] Yonezaki N., Saeki M., Ljungberg J., Kinnula T. "Software process modelling with the TAP approach Tasks-Agents-Products" 3rd European - Japanese seminars on information modelling and knowledge bases, May 1993, Budapest.