Une méthode de conception orientée objets, appliquée aux systèmes de maintenance d'équipements de radiodiffusion

Jean-Christophe Burneau

Université de Nancy I U.F.R. S.T.M.I.A. G.F.D. Informatique

Centre de Recherche en Informatique de Nancy

THESE

soutenue le 4 décembre 1991

pour obtenir le titre de

DOCTEUR DE L'UNIVERSITE EN INFORMATIQUE

UNE METHODE DE CONCEPTION ORIENTEE OBJETS, APPLIQUEE AUX SYSTEMES DE MAINTENANCE D'EQUIPEMENTS DE RADIODIFFUSION

par

Jean-Christophe BURNEAU

Président du jury	M.	Jean-Paul HATON
Rapporteurs	MM.	Gia Toan NGUYEN Pierre MARCHAND
Examinateurs	Mmes	Odile FOUCAUT Odile THIERY
	M.	Hervé CAUDRON

La route se poursuit sans fin,
Descendant de la porte où elle commença.
Maintenant, loin en avant, la route s'étire
Et je la dois suivre, si je le puis,
La parcourant d'un pied avide,
Jusqu'à ce qu'elle rejoigne quelque voie plus grande
Où se joignent maints chemins et maintes courses
Et vers quel lieu, alors ? Je ne saurais le dire.

J.R.R. TOLKIEN, *"Le seigneur des anneaux"*

A Annick et Laura,

Puisse ce chemin à venir
ressembler à celui
que nous avons déjà parcouru

Ce rapport présente un travail qui a été réalisé au Centre d'Etudes et de Recherche de Lorraine de TéléDiffusion de France et dans l'équipe Exprim du Centre de Recherche en Informatique de Nancy.

Je suis très reconnaissant à Monsieur D. Flaender, directeur du CERLOR, de m'avoir accueilli dans ce centre de recherche.

Que Monsieur Hervé Caudron, directeur adjoint du CERLOR et responsable du laboratoire T.M.I., soit remercié pour les importants moyens et les facilités qu'il a mis à ma disposition.

Je souhaite que Madame le Professeur M. Créhange reçoive ma profonde gratitude pour m'avoir intégré à son équipe de recherche et pour la confiance qu'elle m'a accordée.

Je dois des remerciements particuliers à Madame le Professeur O. Thiéry pour l'attention qu'elle a portée à la réalisation de ce travail, pour les nombreux conseils qu'elle m'a prodigués et pour sa patience lors de nombreuses relectures des différents rapports et articles que je lui ai soumis.

J'ai le vif plaisir de remercier Messieurs D. Mafille, J.-C. Ginisty et G. Bourlier, avec lesquels nous avons formé une équipe de choc dans laquelle régnait une cordiale entente.

J'exprime ici mes remerciements à Monsieur P. Woda, ancien responsable de la cellule Application de l'Intelligence Artificielle, pour l'intérêt qu'il a porté aux concepts présentés ici.

Je désire que l'ensemble du personnel du CERLOR ainsi que chaque membre de l'équipe Exprim soient remerciés pour l'excellente ambiance, de part et d'autre, dans laquelle ce travail a été réalisé.

Que tous les autres membres du jury soient également remerciés pour le temps qu'ils ont consacré à la lecture de ce rapport.

Table des Matières

Index des figures

Introduction

L'établissement d'un diagnostic, technique ou médical, fait non seulement appel à une capacité de raisonnement, mais également à une quantité importante de connaissances théoriques, expérimentales, expertes ou de sens commun. Par conséquent, la réalisation d'un outil informatique d'aide au diagnostic nécessite la définition préalable d'un système de représentation des connaissances capable de prendre en compte cet ensemble complexe d'informations : ce problème de la représentation des connaissances constitue l'une des études fondamentales en Informatique Avancée[1].

Le laboratoire «Technologie des Machines Intelligentes» (TMI) du Centre d'Etudes et de Recherches de Lorraine de la société TéléDiffusion de France (TDF-CERLOR) mène des travaux concernant la réalisation de systèmes de Maintenance Assistée par Ordinateur (MAO) destinés à des équipements de radiodiffusion. Ces équipements font intervenir des technologies très différentes (particulièrement de l'électronique, de l'électromécanique, des systèmes de refroidissement hydraulique, des mécanismes de protection et de régulation thermique, ...). De ce fait, leur maintenance impose aux techniciens de maîtriser un volume très important de connaissances, tant au sujet de l'équipement lui-même qu'à propos de la façon dont un diagnostic ou une réparation doivent être réalisés. Pour cette raison, la conception d'un système de maintenance des équipements de radiodiffusion doit débuter avec l'étude d'une représentation de la connaissance adaptée.

L'hypothèse fondamentale de cette étude est que la totalité de la connaissance possédée par un logiciel d'aide à la maintenance est fournie par des intervenants humains, experts dans le domaine concerné. La science apportée par ces experts est transformée en un ensemble de connaissances artificielles (informatiques) aussi proches que possibles des données originales. C'est pourquoi la base de connaissances correspondante contient principalement des informations dites «expertes». Par extension, les systèmes d'aide à la maintenance étudiés au CERLOR sont dits «experts», non du fait d'un raisonnement intelligent, mais parce qu'ils sont fondés sur un ensemble d'énoncés qui proviennent d'un intervenant humain expert. Par conséquent, l'appellation de «Systèmes Experts de Maintenance» (SEM) ne préjuge nullement de la forme du raisonnement adoptée, mais bien des connaissances sur lesquelles est appliqué ce raisonnement.

---

1 *Cette nouvelle acception du sigle IA a été suggérée par J. STERN, P.-D.G. de Bull, lors de sa conférence d'ouverture des 9èmes journées d'Avignon en 1989.*

Notre thèse est alors que les systèmes experts de maintenance constituent un domaine d'application particulier, devant bénéficier de modèles et méthodes spécifiques. Le but de ce travail est de proposer une méthode de conception de systèmes experts de maintenance fondée sur les concepts du ***modèle orienté objets*** et de l'appliquer dans le cadre de la conception d'un système expert de maintenance d'un émetteur de télévision. La caractéristique principale du modèle orienté objets est qu'il permet de représenter aussi fidèlement que possible le domaine étudié, sans contraintes préalables et sans hypothèses sur l'implantation ultérieure. C'est là un avantage qu'il possède sur les méthodes de conception conventionnelles ; en effet, nous émettons l'hypothèse que la difficulté de représentation tient, non dans l'application d'un modèle donné, mais dans l'adéquation de ce modèle au domaine considéré. Les méthodes de conception ont généralement été étudiées dans le cadre de l'informatisation d'organisations : de ce fait, elles incluent habituellement des contraintes de conception telles que la conservation d'historiques, la création d'états imprimés, la gestion de processus en temps réel ... Un SEM n'est pas une organisation dans laquelle circulent des documents qu'il faut traiter ou générer. C'est pourquoi un modèle qui laisse le concepteur libre de s'adapter pleinement au domaine de la maintenance est préférable à un formalisme qui impose la prise en compte d'aspects absents ou inutilisables.

Pour étayer notre thèse, ce travail comporte trois parties. La première partie consiste en une étude globale des méthodes de conception existantes. Cette étude décrit notamment les caractéristiques principales de toute méthode de conception, définies dans [INFORSID 76] par :

- le modèle
- le langage
- la démarche
- les outils.

La totalité de notre discours est fondée sur ce quadruplet de caractéristiques, que nous employons comme critère dans une analyse détaillée de deux méthodes de conception de systèmes d'information. Nous déduisons de cette étude que les méthodes de conception dédiées à la création de systèmes d'informations artificiels sont inadaptées à la conception de systèmes experts. Sur la base de ces constatations, nous exposons en détail le modèle orienté objets, puis étudions les méthodes existantes qui utilisent ce modèle, pour aboutir à la conclusion que ces

méthodes, si elles sont plus proches de notre objectif, n'en permettent pas encore sa réalisation.

A partir de cette étude, nous décrivons dans la seconde partie la méthode SECOOSSE (*Spécification Et Conception Orientées Objet de Systèmes Experts*). Il s'agit d'un modèle conceptuel auquel est associé un modèle de représentation basé sur Information Analysis, qui a été proposé initialement par Nijssen en 1977. SECOOSSE est présentée à l'aide de son propre formalisme. Un environnement de conception, lui-même conçu à l'aide de notre méthode, est ensuite proposé.

La troisième partie de ce document présente l'application de SECOOSSE aux Systèmes Experts de Maintenance des Equipements de Radio-diffusion, dans le cadre du projet STEAMER mené par le laboratoire TMI. Les implantations particulières des fonctionnalités de représentation propres à SECOOSSE, de même que les différents choix de réalisation du système expert, sont détaillés dans cette partie.

Nous concluons cette étude sur un panorama des développements possibles autour d'un système d'aide à la maintenance. Ces réalisations peuvent être générales -des extensions vers une gestion de bases de données multimédia à l'aide d'un système hypertexte, par exemple- ou plus spécifiques au domaine d'application de l'étude réalisée -l'une des propositions étant d'équiper les futurs équipements de radiodiffusion de systèmes de diagnostic intégrés.

Partie I
Vers la proposition d'une nouvelle méthode

Avant toute étude, il est important d'étudier l'adéquation des travaux à réaliser aux besoins des utilisateurs. Or, une enquête menée dans 7000 entreprises utilisant l'informatique montre que les *outils et méthodes de conception* sont jugés prioritaires sur les autres outils (Cf. [MUENIER 90]). De plus, les *outils de spécification* sont importants dans les entreprises techniques tandis que les *outils de modélisation* sont préférés pour l'informatique de gestion. Ces statistiques mettent en lumière un besoin qui s'exprime en trois mots : conception, spécification, modélisation. Nous nous attachons avant tout à définir ces termes en partant d'une définition plus générale, celle d'***analyse*** : l'analyse consiste en un examen ou une décomposition méthodiques de quelque chose qui existe déjà. Au contraire, le terme ***conception*** se réfère au processus de définition de quelque chose qui n'existe pas encore. Une ***spécification***, par contre, est le produit indifféremment de l'analyse ou de la conception. Enfin, la ***modélisation*** est la création d'une abstraction à partir d'un existant, une abstraction consistant en une formalisation dans laquelle certains détails ont été délibérément omis afin que seule subsiste une description utile.

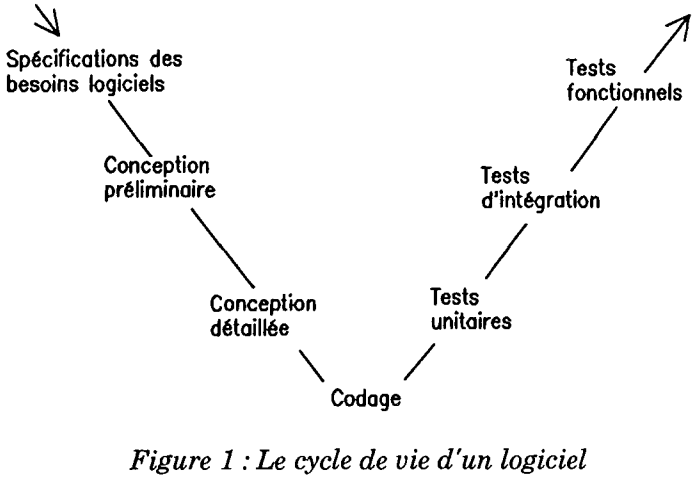


Figure 1 : Le cycle de vie d'un logiciel

Le processus de conception d'un système informatique d'un système est décrit, par exemple, dans [OLLE 90] ou [PERNICI 90]. Il débute par une analyse qui aboutit à une modélisation ; cette modélisation est ensuite exploitée pour spécifier les besoins logiciels (Cf. en particulier [PERNICI 90] et [VISSER 88]). A partir de l'énoncé et de la modélisation, une équipe d'analyse suggère une première solution technique exprimée à l'aide d'un ***modèle conceptuel***. Une deuxième étape de conception, la conception détaillée, aboutit à un schéma d'implantation mettant en évidence les structures de données nécessaires pour obtenir un système efficace quel que soit le caractère critique des transactions. Comme [OLLE 90], nous nommerons produit de la conception indifféremment la spécification des besoins, le schéma conceptuel ou le système informatique obtenu.

Par rapport au cycle de vie d'un logiciel, présenté en figure 1, nous ne nous intéressons qu'aux trois premières étapes, les étapes de codage et de test relevant d'une problématique différente.

Un processus de conception peut sans aucun doute être mené sans que ses différentes étapes soient mieux définies : n'est-ce pas ce que fait généralement un informaticien qui doit écrire un petit programme utilitaire ? Cependant, dès que les produits de la conception deviennent importants (en volume de papier ou de code informatique, voire en communications entre les différents intervenants dans la conception), cette absence de règles de conception devient pénalisante et une approche *méthodologique* du processus de conception s'avère indispensable. Cette constatation amène immanquablement à l'utilisation d'une ***méthode de conception***. Avant tout, une méthode de conception est une méthode[1], définie comme la manière de dire, de faire, d'enseigner une chose suivant certains principes et avec un certain ordre. Une méthode de conception consiste donc en un ensemble de règles régissant le processus de conception. Elle a pour rôle l'assistance à la recherche d'une solution informatique à un problème donné. Elle apporte une aide à la formulation et à la maîtrise des problèmes à résoudre et permet de construire des applications complètes, cohérentes, fiables, flexibles et adaptatives. Une méthode de conception substitue à un développement individuel une concertation entre experts, utilisateurs et concepteurs ; les solutions empiriques sont supprimées, au profit de réalisations mettant en oeuvre des outils et des compétences rigoureux.

L'utilisation de l'informatique peut être, très globalement, de deux natures : l'informatique de gestion consiste en une utilisation des ordinateurs pour la maintenance des informations d'une entreprise ou d'une organisation ; l'informatique technique a pour finalité de remplacer l'homme dans des tâches qu'il n'apprécie pas ou qu'il n'est pas capable d'assumer. Informatique de gestion et informatique technique sont donc complètement différentes ; cependant, dans ces deux domaines, les méthodes de conception sont nécessaires. Dans le domaine de la gestion des entreprises, une définition a été posée très tôt (Cf. [INFORSID 76]). Cette définition indique qu'une méthode de conception est composée de :

- un ***modèle***, ensemble de concepts et de règles pour les utiliser, destiné à expliquer et représenter les éléments qui composent le système organisationnel et leurs relations.
- un ***langage***, ensemble de constructions qui permettent de décrire formellement les spécifications.

---

1 *du grec méta (après) et odos (voie).*

- une ***démarche***, processus opératoire pour le travail de modélisation, de description, d'évaluation et de réalisation.
- un ***outil logiciel*** qui supporte la démarche.

Nous notons que cette définition est propre aux méthodes de conception de systèmes de gestion, dans la mesure où le modèle est destiné à représenter l'entreprise elle-même. Cependant, si l'on excepte ce point, le quadruplet proposé peut définir n'importe quelle méthode de conception. C'est pourquoi nous exploitons cette définition des méthodes de conception, en redéfinissant un modèle :

- un ***modèle*** est un ensemble de concepts et de règles pour les utiliser, destiné à représenter un univers réel, concret ou abstrait, ainsi que les règles qui régissent cet univers.

Suite à ces définitions fondamentales, nous rappelons que l'objet de ce travail est la conception d'un Système Expert de Maintenance de seconde génération (SEM2G). Avant de démarrer le processus de conception, il nous faut choisir une méthode de conception adaptée à ce type de logiciel, donc définir des critères de comparaison et de choix, étudier plusieurs méthodes afin de déterminer celles qui sont parfaitement adaptées à notre domaine. Le chapitre A définit les SEM2G et établit une liste de critères qui doivent être respectés par la méthode qui sera choisie. L'informatique de gestion est une grande pourvoyeuse de méthodes de conception, appliquées à ce que les gestionnaires nomment les Systèmes d'Informations (SI). C'est pourquoi nous étudions, dans le chapitre B, deux méthodes de conception de systèmes d'informations (MCSI) ; nous conclurons de cette étude que cette catégorie de méthodes n'est pas adaptée au domaine des SEM. Nous avons adapté la définition du *modèle* d'une méthode de conception de telle sorte qu'elle soit toujours valable. Or, il existe un modèle de données qui a depuis peu la faveur de la communauté informatique : le modèle orienté objets. Nous définissons ce modèle dans le chapitre C et montrons qu'il respecte la définition ci-dessus. Nous étudions donc, dans le chapitre D, des méthodes fondées sur ce modèle, mais montrons que ces méthodes ne conviennent pas non plus pour la conception d'un SEM2G. Nous concluons dans le chapitre E par un récapitulatif des méthodes de conception étudiées. Ce récapitulatif met en évidence la nécessité de créer une méthode de conception spécifique à l'élaboration de systèmes experts de seconde génération.

Chapitre A
Critères de choix d'une méthode de conception de SEM2G

L'objectif de ce travail est de fournir les spécifications d'un Système Expert de Maintenance de seconde génération, appliqué aux équipements de radiodiffusion. Les particularités du logiciel désiré tiennent intégralement dans sa dénomination :

- *maintenance* : L'AFNOR [AFNOR 88] définit la maintenance comme étant «l'ensemble des actions permettant de maintenir ou de rétablir un bien dans un état spécifié ou en mesure d'assurer un service déterminé». La maintenance d'un matériel consiste donc :
  . à assurer son fonctionnement permanent (tel n'est pas le rôle du logiciel désiré)
  . en cas de panne, à diagnostiquer l'origine de la défaillance du matériel et à la réparer.

- *système expert* : [GALLAIRE 85] définit un système expert comme un programme informatique dans lequel la programmation n'est pas impérative, mais déclarative. Les connaissances sont exprimées «en vrac» dans une base de connaissance, exploitée par un moteur d'inférence.
- *seconde génération* : contrairement aux systèmes experts de première génération, dans lesquels les connaissances sont déclarées formellement sous forme de règles, les systèmes experts de seconde génération se basent sur une représentation de la réalité pour raisonner.
- *équipements de radiodiffusion* : le SEM2G qui doit être créé a pour but la maintenance des émetteurs de télévision qui transmettent la deuxième chaîne. Ces émetteurs, très puissants (2*25 KW), sont très complexes et font intervenir à la fois des composants électroniques, électro-mécaniques, mécaniques et hydrauliques.

A partir de cette présentation très succincte des applications que nous devons créer, nous pouvons définir les critères auxquels doit répondre la méthode de conception utilisée :

(1) il est fondamental que la méthode de conception utilisée permette de modéliser exactement la structure et le fonctionnement de l'équipement cible.

(2) dans la réalité, plusieurs intervenants humains s'occupent de la maintenance d'un même équipement : par exemple, un diagnostic sur tube d'amplification de signal image est réalisé par les techniciens du site d'émission, mais les réparations et les changements sont opérés par des techniciens de la société qui fabrique ce tube. Dès lors, la prise d'expertise fait intervenir non un expert, mais un ensemble d'experts interrogés par plusieurs cogniticiens chargés de recueillir la connaissance. La méthode de conception doit tenir compte du fait que les sources d'informations sont multiples.

(3) par extension du critère précédent, nous prenons en compte le fait que plusieurs concepteurs doivent oeuvrer en même temps. En effet, la connaissance recueillie est de deux ordres :

- connaissance de l'équipement lui-même.
- connaissance de la démarche de diagnostic.

De plus, certains aspects propres au systèmes sont réalisés par un concepteur particulier : c'est le cas de l'interface du SEM, qui fait intervenir un ergonome.

La méthode de conception doit par conséquent permettre à plusieurs concepteurs de concevoir, simultanément et indépendamment, des aspects différents d'une même application.

(4) le but de l'application n'est pas de *reproduire* le fonctionnement de l'équipement. Le fonctionnement, lorsqu'il est modélisé, est utilisé comme *support* du diagnostic. Le raisonnement de diagnostic n'est pas fondé sur une suite d'événements dans l'équipement, mais sur une inférence globale à la modélisation.

(5) le point précédent induit que données et traitements sont très peu dissociés puisque le raisonnement de diagnostic est fondé sur la modélisation de l'univers réel.

(6) le raisonnement de diagnostic est différé par rapport au fonctionnement de l'équipement (il n'intervient qu'après que la panne ait eu lieu, donc que

l'équipement ait été arrêté). Il n'est par conséquent pas utile que la méthode de conception inclue la gestion de processus en temps réel.[1]

Ces critères doivent être vérifiés pour la méthode de conception qui sera retenue.

1 *Cet aspect de diagnostic différé n'est pas une généralité. En particulier, [COURTIN 88] présente un système expert de diagnostic embarqué qui fonctionne en temps réel sur la base de mesures télémétriques.*

Chapitre B
Méthodes de Conception de Systèmes d'Informations

Historiquement, les matériels informatiques n'ont tout d'abord été accessibles qu'à des entreprises importantes, désireuses d'automatiser leur fonctionnement. Après l'apparition du micro-ordinateur, les entreprises qui ont acquis de tels matériels ont été en nombre croissant ; l'informatisation d'une société est désormais un fait habituel, sinon indispensable. A toute époque, les entreprises ont attendu de l'informatique qu'elle reproduise au mieux leur fonctionnement afin de le simplifier, de le rendre plus fiable, voire de l'optimiser. Cet impératif de *reproduction* des organisations a amené le développement d'une branche particulière de l'informatique, appelée ***informatique de gestion*** ou ***informatique des organisations***. C'est dans cette branche, qui inclut moins des techniciens de l'informatique que des sociologues et des gestionnaires, qu'ont été développés des outils particuliers dédiés à l'analyse, à la formalisation et à la représentation des organisations. Ces outils ont pour rôle la création d'un ***système d'informations*** (SI), c'est à dire un objet automatisé destiné à recueillir, mémoriser et traiter des informations afin d'en alimenter les centres de décision de l'organisation ou de l'entreprise. Profitant à tout moment des derniers développements des bases de données et des modèles de données, ces outils sont devenus de véritables méthodes de conception, incluant à la fois un modèle de description de l'organisation, un modèle de représentation des données et souvent une démarche de conception aboutissant à une implantation finale.

[ROLLAND 87] décompose les MCSI en deux grandes classes : les méthodes *structurées* et les méthodes *systémiques*. Les méthodes structurées mettent l'accent sur la démarche de conception tandis que les méthodes systémiques mettent en valeur la compréhension du SI et sa décomposition rigoureuse. Après un bref rappel sur l'évolution des MCSI en B.I, nous étudions dans les paragraphes B.II et B.III une méthode de chaque classe : SADT (méthode structurée présentée en particulier par [IGL 88]) et REMORA (démarche systémique, Cf. [ROLLAND 87]).

Nous concluons ce chapitre sur l'inadéquation des MCSI à la conception de systèmes experts de seconde génération.

B.I. Historique des MCSI

Pendant longtemps, la seule solution pratique efficace pour créer des systèmes informatiques important a été la Base de Données. L'évolution des méthodes de conception suit donc nécessairement l'évolution de ces Bases de Données et particulièrement celle des modèles selon lesquels elles sont organisées.

B.I.1. L'évolution des modèles

Dans les années 1950, l'unité de traitement était le fichier. Les modèles utilisés étaient essentiellement basés sur la description textuelle dont la syntaxe variait d'une école à l'autre. Comme le précise [THIERY 85], la préoccupation principale était alors la recherche de langages à syntaxe «propre», adaptés aux différents domaines d'application de l'informatique. [ROCHFELD 89b] rapproche ces modes de programmation de recettes, plutôt que de méthodes.
Dans les années 1960 sont apparues les bases de données, qui intégraient des fichiers multiples. Cette décennie est caractérisée par un effort pour s'abstraire des notations propres aux langages de programmation, effort dont la *programmation structurée* est l'aboutissement. Parallèlement de nouveaux modèles, fondés sur les réseaux ou les hiérarchies, sont utilisés dans des méthodes d'analyse, couramment basées sur l'étude des organisations à informatiser.
C'est au début des années 1970 que le ***modèle relationnel*** des données est apparu (Cf. [CODD 70]). Ce modèle consiste à organiser les informations en tableaux indépendants de l'implantation finale. Les structures physiques des bases de données peuvent évoluer, la description de l'organisation de l'information reste identique ; le modèle relationnel a été normalisé avec un rapport ANSI-SPARC en 1975. Après ce rapport, grâce au concept unique de *relation*, le modèle relationnel fut le premier à apporter un schéma homogène créé par une approche méthodologique, une indépendance entre l'organisation logique et physique des données et un langage non procédural de manipulation. Le modèle relationnel a rapidement donné naissance à des modèles incluant plus de sémantique (citons en particulier le modèle Entité-Relation proposé dans [CHEN 76] ou le modèle RMT proposé par Codd dans [CODD 79]).
Cependant, le modèle relationnel est resté relativement figé, si on excepte les travaux sur les formes normales (Cf. [BRODIE 91]), malgré des progrès techniques énormes, dans le cadre des matériels comme des logiciels. Les années

1980 ont vu l'apparition des données complexes et des données multimédia (images -voire images animées-, sons, ...), dont la manipulation entraîne une baisse relative des performances, sinon une impossibilité de traitement, de la part des systèmes relationnels.
C'est pourquoi, durant les années 80, de nouveaux modèles sont apparus. Ces nouveaux modèles utilisent, de plus en plus souvent, la théorie des réseaux sémantiques. Nous citerons notamment IA (Cf. [NIJSSEN 77], [VERHEIJEN 82], [VERMEIR 82], [HABRIAS 88]), FDM (Functionnal Data Model) ou SDM (Semantic Data Model) (Cf. [HULL 87]). Ces modèles garantissent une représentation stable et cohérente et assurent une indépendance totale (ou censée l'être) entre l'expression des concepts et leur implantation physique.
Les années 1990 amènent de nombreux nouveaux développements sur plusieurs axes de recherche : bases de données déductives (Cf. [BAZEX 89], [DESICY 89]), bases de données orientées objets, machines bases de données (...) dont l'influence sur les modèles des méthodes de conception ne saurait être des moindres (Cf. [ROCHFELD 89a-89b]).

B.I.2. L'évolution des méthodes de conception

La notion de méthode de conception est apparue avec l'utilisation de moyens informatiques importants dans les organisations et notamment avec l'apparition des bases de données. Le rapport ANSI-SPARC de 1975 a servi de point de départ pour une nouvelle approche de la conception de bases de données, notamment grâce aux trois niveaux (conceptuel, logique et physique) qu'il propose. L'évolution des méthodes est décrite dans [OLLE 90], [ROCHFELD 89b] et [THIERY 85] : les premières méthodes portaient sur la représentation des données à l'aide du modèle relationnel, puis elles se sont enrichies dans plusieurs directions : à mesure que les modèles devenaient plus précis, la prise en charge méthodologique des différentes étapes de conception et la répartition des tâches au sein d'une équipe de concepteurs leur ont été intégrées.
Les modèles à base de réseaux sémantiques pouvant être eux-même formalisés, des outils d'aide à la conception sont ensuite apparus. Ces outils ont d'abord pris en charge la mémorisation et la restitution des concepts. Ils ont ensuite permis la construction incrémentale des spécifications (Cf. [COMYN-WATTIAU 89]).
Profitant des progrès de l'IA, ces outils savent de mieux en mieux gérer la cohérence de contraintes exprimées par les différents concepteurs au moyen d'outils experts (citons les prototypes et applications SEIVE de

[BOULANGER 87], SECSI de [BOUZEGHOUB 86], LACSI de [ROCHE 88-89], OICSI de [CAUVET 87-90], ...).

B.II. Une méthode structurée : SADT

La méthode SADT (*Structured Analysis and Design Technique*), présentée dans [IGL 88], est essentiellement fondée sur un modèle en deux parties, l'une s'appliquant aux données et l'autre aux traitements. Un langage graphique adapté à ces modèles ainsi qu'une démarche bien définie en font une méthode appréciée dans le domaine de l'informatique technique.

B.II.1. Modèle

Le modèle de SADT est fondé sur les notions de module et de sous-module. Un module est une description du système modélisé dans un niveau d'abstraction ; ce module peut être décomposé en sous-modules, chacun représentant alors une composante du module. Les relations de chaque module avec les autres modules sont précisées par des arcs labellés.
Dans le modèle des traitements, chaque module représente une fonction ; les arcs indiquent alors les données nécessaires à cette fonction et les informations qu'elle génère. Eventuellement, des arcs particuliers définissent des données de «contrôle» de la fonction, qui précisent comment se déroule l'activité modélisée par le module.
Dans le modèle des données, le rôle des modules et des arcs est inversé : chaque information est modélisée par un module, les traitements qui génèrent ou utilisent cette information étant modélisés par des arcs.

Le modèle de SADT présente l'avantage d'allier une simplicité extrême à une modélisation simultanée des données et des traitements. Ce modèle peut ou non, au gré des concepteurs, laisser transparaître les solutions techniques. Sa simplicité ne permet pas de faire apparaître et de regrouper les objets identiques. La minimalité n'est donc pas nécessairement respectée. Par contre la complétude est assurée par la démarche qui est associée au modèle.

B.II.2. Langage

Le langage de SADT est aussi simple que son modèle, puisque chaque module est représenté par une boîte et chaque arc par une flèche. Des conventions graphiques concernant les points de départ et d'arrivée d'une flèche déterminent sa nature (contrôle d'une activité, entrée, sortie, ...). Les schémas SADT sont réalisés sur des feuilles dont les entêtes fournissent des informations sur leur

contenu. De même que les modèles des données et des traitements sont identiques, les schémas d'analyse des traitements (nommés *actigrammes*) et des données (*datagrammes*) sont réalisés à l'aide d'un langage unique.
La lisibilité d'un schéma SADT est excellente ; cependant les différents niveaux d'abstraction, qui sont matérialisés par des schémas différents, alourdit le processus de lecture en imposant de nombreux changements de schémas.

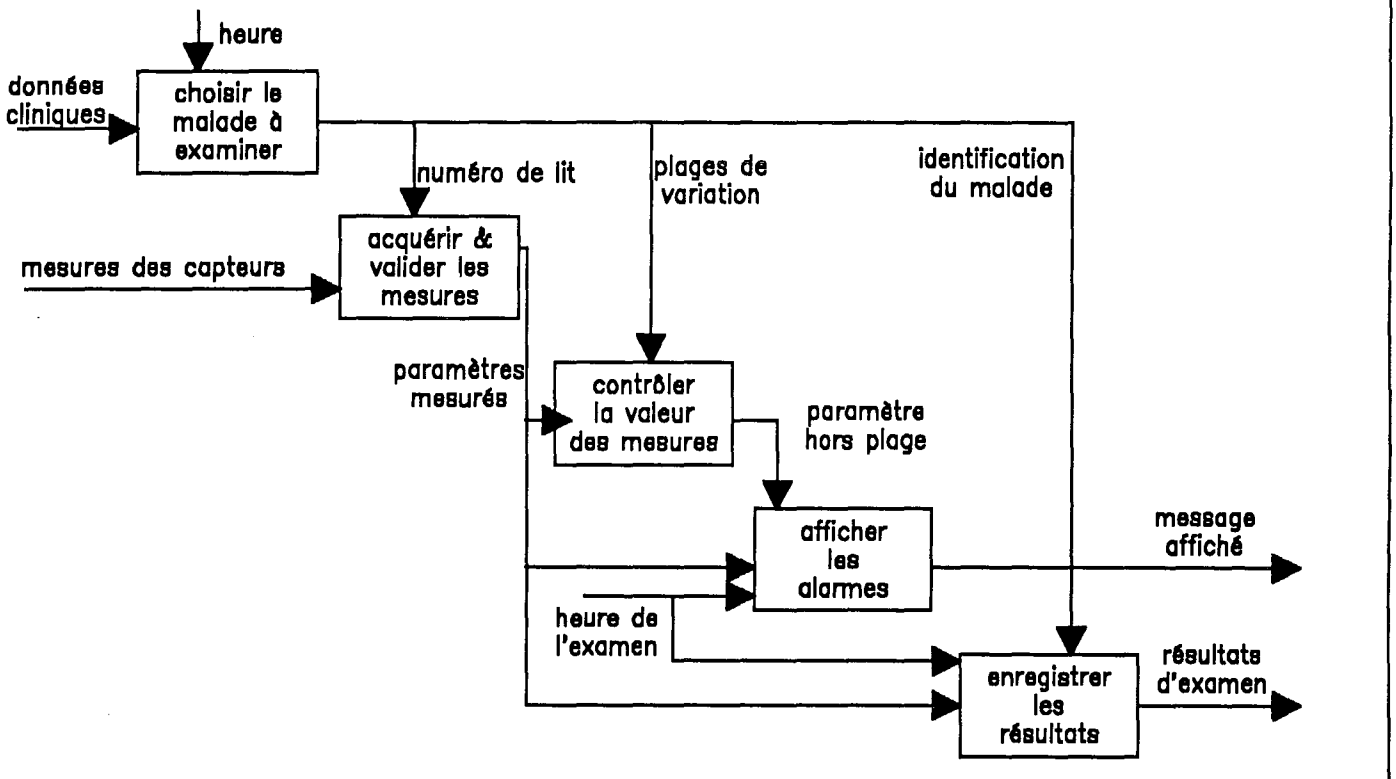


Figure 2 : Un actigramme SADT de suivi d'un malade

L'actigramme présenté figure 2 formalise un exemple de suivi des malades d'un hôpital, couramment retrouvé dans la littérature sur SADT. Le module de choix du malade à examiner nécessite, comme informations, l'ensemble des données cliniques des malades ; il est contrôlé par l'heure à laquelle l'examen a lieu, et génère trois informations :

- le numéro de lit du malade
- les plages de variations admises pour les mesures à effectuer
- l'identification du malade.

L'acquisition des mesures fait intervenir des informations provenant des capteurs (en fonction du numéro de lit du malade) et génère la liste des mesures.

Cette liste est validée par un module de contrôle, chargé d'afficher les alarmes en fonction des plages de variation fournies par le premier module.
Enfin, tous les résultats sont enregistrés en même temps que l'identification du malade.

B.II.3. Démarche

La démarche de SADT est descendante, modulaire et hiérarchique :

- descendante en ce sens que la conception va d'une description la plus générale et abstraite possible vers une description la plus détaillée possible.
- modulaire par son modèle.
- hiérarchique en ceci que les informations sont structurées sous une forme parent/enfant.

En fait, on peut considérer que c'est la démarche qui fait la méthode, bien plus que le modèle. En effet, SADT donne un grand nombre de directives, de contraintes et de suggestions (par exemple, tout choix effectué durant l'analyse doit être consigné par écrit) qui permettent aisément à plusieurs concepteurs de partager leurs spécifications.

B.II.4. Application aux SEM2G

B.II.4.a) Avantages

SADT est fondamentalement une méthode d'analyse et de modélisation de l'existant, beaucoup plus qu'une méthode de conception. Elle est donc adaptée à la modélisation de la structure et du fonctionnement de l'équipement.
SADT est conçue pour être utilisée par plusieurs concepteurs simultanément, ce qui correspond à nos critères.

B.II.4.b) Inconvénients

Le premier avantage cité se retourne contre SADT lorsqu'il est question de concevoir la partie "raisonnement", c'est-à-dire de créer des traitements qui n'appartiennent pas à une réalité physique, mais qui formalisent des connaissances abstraites. En particulier, la démarche SADT n'inclut aucune règle d'assistance à l'introduction de concepts abstraits dans un modèle de la réalité ; au contraire, le modèle de la réalité forme une spécification homogène et cohérente, sur laquelle il est difficile et délicat de «greffer» des spécifications

abstraites. Dès lors, il est difficile de créer des datagrammes et -l'expérience le montre- seuls les actigrammes sont réellement conçus et réalisés.
Dans le même ordre d'idées, il faut bien noter que la démarche descendante, telle qu'elle est pratiquée dans SADT, s'avère peu pratique lors des mises à jour et corrections. En effet, la suppression ou la modification d'un module qui a été décomposé entraîne la correction d'un grand nombre de schémas qui y faisaient référence.

B.III. Une méthode systémique : REMORA

La méthode REMORA est présentée dans [FOUCAUT 82], [ROLLAND 82-83-85-87], [THIERY 85-86]. Cette méthode est basée sur le modèle relationnel. Elle intègre les approches orientée donnée, orientée traitements et orientée dynamique avec cependant une certaine focalisation sur cette dernière.
REMORA est une MCSI complète, qui comprend une étape conceptuelle et une étape logique distinctes, clairement définies. Nous n'étudions ici que l'aspect conceptuel proposé par la méthode.

B.III.1. Modèle

Le modèle de REMORA est de type relationnel : tout phénomène réel, quelle que soit sa catégorie, est représenté par tout ou partie d'une ou plusieurs relations. L'approche dynamique est visible au niveau du modèle par la normalisation des phénomènes réels qu'il impose : une structuration de ces phénomènes en classes permet une expression simple de la dynamique. Le temps est intégré dans le modèle relationnel, permettant une représentation historique, complète et minimale des classes de phénomènes. Des types de relations et de constituants permettent la prise en compte des différentes sémantiques existant entre les classes de phénomènes et leurs propriétés.

B.III.2. Langage

REMORA offre une convention de représentation graphique et un langage textuel pour la représentation des schémas conceptuels.

B.III.2.a) Langage graphique

Le langage graphique de REMORA comporte deux parties qui mettent respectivement en évidence la composante statique et la composante dynamique du futur SI.
Le sous-schéma statique définit la structure conceptuelle des données à l'aide de conventions qui permettent de décrire l'organisation des informations, leur

regroupement en classes et les relations entre classes. Ainsi, la figure 3, tirée de [ROLLAND 87], met en évidence la formalisation d'une ligne de commande : à un numéro de ligne correspondent toujours un numéro de commande, un numéro d'article, une semaine de livraison et une quantité commandée. Par contre, l'état de la ligne varie en fonction de la date ; à un couple <numéro de ligne,date> correspond toujours un état unique (par exemple, "Attente de fabrication", "Préparation de livraison", ...). Enfin, la quantité du produit commandé doit être affectée à la ligne de commande en magasin ; en fonction de la date d'affectation, la quantité affectée peut varier. C'est pourquoi on associe au couple <numéro de ligne,date) la quantité affectée par le magasin.
Ce sous-schéma conceptuel statique met directement en évidence les entités qui devront être manipulées à l'aide de «boîtes» (Ligne-cde, Etat-lig, Affectation-mag) qui matérialisent l'ensemble des caractéristiques indépendantes du temps.

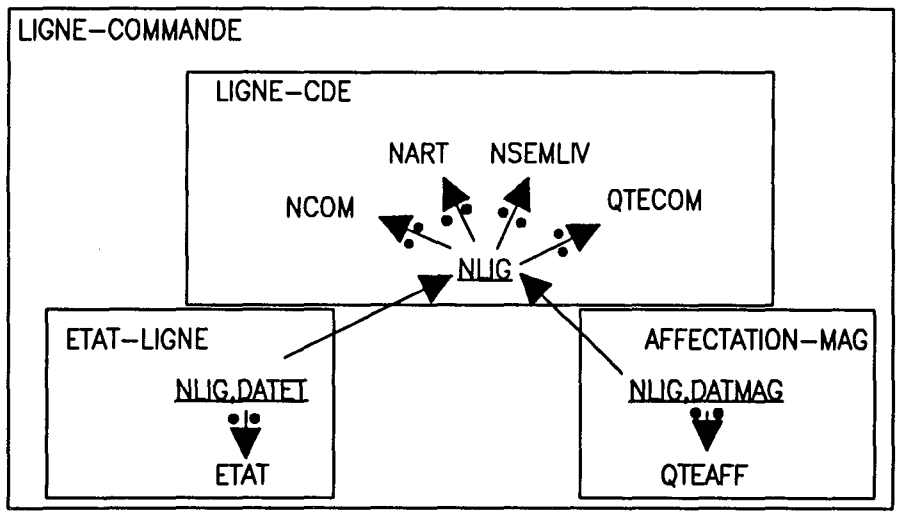


Figure 3 : Sous-schéma conceptuel statique REMORA représentant la classe des lignes de commande

Le sous-schéma dynamique de REMORA définit les situations dans lesquelles les données du SI sont modifiées, la nature des modifications et leurs conséquences sur les données qui peuvent à leur tour générer des événements. Ainsi, la figure 4 montre que l'arrivée d'une ligne de commande (événement EV1, portant sur un objet de type Ligne-Cde) entraîne des traitements différents selon que le stock de l'article commandé est suffisant (condition C1) ou non :

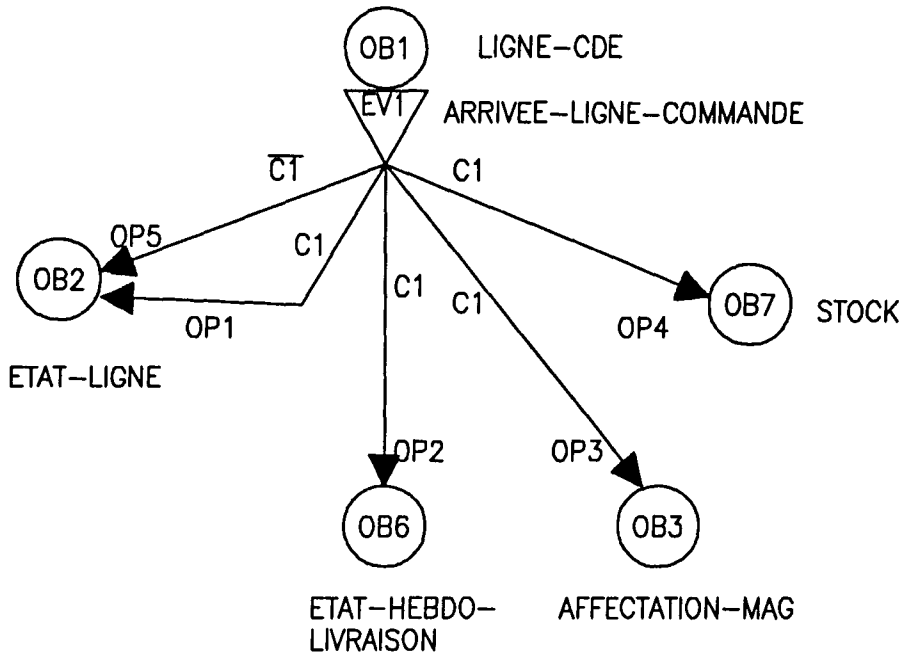


Figure 4 : Sous-schéma conceptuel dynamique REMORA représentant la classe des lignes de commande

- si la condition C1 est réalisée, on affecte les quantités disponibles en stock à la ligne de commande (opération OP3), on effectue la mise à jour correspondante dans le stock (opération OP4), on prépare un état récapitulatif des lignes à livrer dans la même semaine (opération OP2) et on met à jour l'état de la ligne, qui passe à "en cours de préparation de livraison" (opération OP1).
- si par contre C1 n'est pas réalisée, la ligne de commande est mise dans l'état "Attente de fabrication (opération OP5).

B.III.2.b) Le langage textuel

La description textuelle décrit tous les détails des éléments du schéma conceptuel (Cf. [THIERY 85]). Elle reprend et complète la description graphique et décrit les contraintes d'intégrité, qui ne sont pas exprimés graphiquement. Ce langage, nommé dans [ROLLAND 87] REMO-LANGUE, inclut à la fois les constructions relationnelles et des constructions plus algorithmiques. Sa syntaxe est empruntée au langage relationnel SQL.

B.III.3. La démarche

La démarche de REMORA débute par une modélisation, les spécifications obtenues étant ensuite conceptualisées. Le schéma conceptuel correspondant

peut alors être normalisé puis validé avant d'être transformé en une spécification conceptuelle finale.
La majeure partie des règles impératives de la démarche de REMORA est destinée à la normalisation des schémas conceptuels.

B.III.4. Application aux SEM2G

Le modèle de la dynamique de REMORA fonde toute l'activité du système informatique sur la notion d'événement, c'est-à-dire de changement d'état d'un objet. Or, nous l'avons vu avec le critère (4), nous ne désirons pas reproduire le fonctionnement d'une entité réelle[1] ; un raisonnement de diagnostic ne peut être intégralement exprimé sous forme d'événements qui déclenchent des actions. Par conséquent, le modèle de la dynamique est peu adapté à la conception d'un SEM2G.
De plus, le modèle de la dynamique ainsi que le langage graphique associé imposent la formalisation d'un grand nombre de concepts dès que le système formalisé est important. Nous regrettons que plusieurs schémas soient nécessaires pour l'expression de ces concepts, cette multiplicité rendant les produits de la conception difficilement maintenables.
La représentation statique du fonctionnement d'un équipement de radiodiffusion nécessite probablement la formalisation de concepts mathématiques (transformation d'un signal X en un signal Y, ...). Cet aspect «informatique scientifique» et «informatique de calcul» est très difficilement modélisable à l'aide de REMORA.
Enfin, REMORA ne propose pas de solution pour la création de schémas conceptuels par plusieurs intervenants.

B.IV. Conclusion

Les MCSI sont principalement destinées à reproduire le fonctionnement d'une organisation et non à créer des solutions informatiques à des problèmes quelconques. Même SADT, qui est pourtant indifféremment utilisée en informatique de gestion ou en informatique technique, n'offre pas de support méthodologique à la *conception* elle-même. Par contre, naturellement, sur la base des deux méthodes étudiées, nous constatons que les MCSI sont parfaitement adaptées à la *modélisation* d'univers simples, contenant principalement des objets simples. Or, le critère (1) défini dans le chapitre A indique qu'il est nécessaire de formaliser les équipements à maintenir, donc des éléments complexes, qui interagissent fortement entre eux. De plus, le critère (4) rappelle

1 *Dans ce cas, le modèle dynamique serait relativement adapté.*

que nous ne désirons pas reproduire l'équipement cible, ce qui va à l'encontre du principe des Systèmes d'Informations, dont le but est de simuler les organisations cibles ; au contraire, les critères (4) et (5) signalent que nous désirons formaliser un raisonnement complexe, qui n'est pas défini comme étant déterministe, fondé autant sur les données que sur les traitements, ce qui semble dépasser quelque peu le cadre de la conception de Systèmes d'Informations.

Pour ces différentes raisons, fondées sur l'étude des méthodes SADT et REMORA, nous nous proposons d'utiliser un modèle et une méthode de conception qui ne sont pas issus de l'univers des Systèmes d'Information : nous analysons dans les chapitre suivants un modèle de données et les méthodes associées : le *modèle orienté objets* et les *méthodes de conception orientées objets*.

Chapitre C
Le modèle orienté objets

Ayant montré que les modèles utilisés habituellement dans les méthodes de conception de systèmes d'informations ne nous permettent pas de réaliser nos objectifs, nous nous proposons d'étudier un modèle de données récent : le ***modèle orienté objet***s. L'expression *Orienté Objets* (OO) est de plus en plus utilisée, dans des contextes de plus en plus variés, désignant indifféremment des langages de programmation, des bases de données, des démarches conceptuelles, ... L'apparition d'outils de plus en plus nombreux, parés de ce qualificatif, conduit à une certaine confusion -pour ne pas dire une confusion certaine- quant à la définition de ce que d'aucuns nomment le *paradigme objet*[1]. Les termes les plus communément entendus sont ceux de *langages à objets*, de *langages orientés objets*, de *bases d'objets* ou *à base d'objets*. Ces formulations ne sont pas équivalentes, loin de là. Par contre, elles peuvent être regroupées en une seule définition générique que nous reprenons à [BARBIER 89] : «le terme ***système à objets*** désigne indifféremment les langages ou systèmes d'acteurs, centrés objets ou orientés objets». Notons tout d'abord que cette définition intéresse non seulement les *langages*, mais tout *système* à objets. Nous considérons le terme «système» dans son sens le plus large, sans connotation industrielle, logicielle ...
Dans ce contexte, [BLAIR 89] définit, à partir des propositions de [WEGNER 87], les ***systèmes à base de classes*** comme des systèmes centrés objets auxquels a été ajouté le concept de *classe*. Les ***systèmes orientés objets*** sont alors des systèmes à base de classes auxquels a été ajouté le concept *d'héritage*.

Nous reprenons ces trois définitions de systèmes centré objet, à base de classes et orienté objets pour en décrire le modèle de façon détaillée. Cette étude est réalisée sur la base de définitions proposées par [AUBERT 89], [AULD 88-89], [BARTHES 90], [BLAIR 89], [BARBIER 89], [BOIS 89], [CARRE 89], [COX 86], [LEONARD 89], [MASINI 89], [ROSEN 90], [WEGNER 87-88]. Elle nous conduit à définir ce que nous qualifierons par la suite de *modèle orienté objets standard.* Dans les paragraphe C.I et C.II, nous définissons les systèmes centrés objets et

---

1 *Paradigme : par extension du latin des grammairiens, exemple ou modèle général.*

décrivons comment des objets différents peuvent être mis en relation pour former une application. Nous mettons en évidence dans le paragraphe C.III la nécessité de la classification et en donnons la définition, avant d'expliquer le mécanisme d'héritage. Enfin, nous montrons dans le paragraphe C.IV l'intérêt que présente le modèle orienté objets dans le contexte des Systèmes Experts de Maintenance avant de proposer notre conclusion sur le modèle étudié en C.V.

C.I. Systèmes Centrés Objets

C.I.1. Le modèle centré objets statique

Le modèle des systèmes centrés objets est fondé sur un principe de structuration permettant de représenter chaque entité de l'univers du discours sous une forme intuitive et fidèle : ce principe de structuration est nommé ***objet***.
Un système réel est représenté à l'aide d'un ensemble d'objets différents ; chacun des objets qui interviennent dans cette représentation possède une *identité* qui lui est propre. Nous formalisons cette identité en attribuant à chaque objet un ***identifiant*** de forme quelconque (nombre, atome, ...) mais *unique*. L'identité de l'objet est alors assimilée à cet identifiant.
Un objet qui intervient dans le système réel peut être caractérisé par des grandeurs, des positions, des étapes d'évolution, et de nombreuses autres quantifications ou qualifications. Pour représenter ces différentes caractéristiques d'une entité, nous définissons un objet comme ***un ensemble de valeurs***. L'ensemble des valeurs d'un objet est habituellement appelé ***état*** de cet objet. Chaque valeur formalise une caractéristique de l'entité représentée (par exemple une taille de *13 centimètres*, une position *ouverte* ou *fermée*, ...).
Certaines caractéristiques sont «simples», mais certaines valeurs peuvent consister en une *référence* à d'autres objets. Par exemple, un relais est composé de sa bobine et de son contact. Chacun de ces objets, le relais, la bobine et le contact possèdent par définition un identifiant ; il existe alors dans l'état du relais, une valeur qui est l'identifiant de la bobine et une valeur qui est l'identifiant du contact. Ces valeurs de liaison entre objets permettent de définir toutes sortes de relations qui existent dans l'univers du discours. En particulier, nous notons que cette notion de référence ne préjuge pas de la nature de la relation. Par exemple, [BRODIE 84] définit une relation d'association dans laquelle un ensemble d'objets similaires est considéré comme un unique objet-ensemble de plus haut niveau ; les caractéristiques d'un objet membre sont alors supprimées tandis que celles de l'objet-ensemble sont mises en évidence (l'exemple donné étant l'objet DOGS qui consiste en une association d'objets DOG).

Dans le modèle centré objets, l'objet DOGS possèderait une caractéristique qui est l'ensemble des identifiants des objets DOG qui le constituent.
De la même façon, [SMITH 77a] définit l'abstraction d'agrégation, selon laquelle plusieurs objets peuvent être considérés comme un seul objet avec un plus haut niveau d'abstraction. L'exemple donné concerne les objets HOTEL, CHAMBRE, DATE et PERSONNE, qui peuvent être considérés comme un seul objet RESERVATION. Dans le modèle centré objets, l'objet RESERVATION possèderait quatre caractéristiques, l'une référençant l'objet HOTEL, la seconde l'objet DATE, la troisième l'objet CHAMBRE et la quatrième l'objet PERSONNE.

Naturellement, dans le système réel, les caractéristiques d'un objet évoluent. Cette évolution des caractéristiques est représentée par des changements de valeurs, donc par des modifications de l'état des objets. L'une des hypothèse du modèle centré objets est que la modification de l'état d'un objet est réalisée sous la responsabilité exclusive de cet objet. Une autre hypothèse est que chaque entité sait comment se comporter, indépendamment des autres objets. Nous unifions ces deux hypothèses en énonçant que chaque entité possède un comportement qui lui est propre, ce comportement pouvant entre autres entraîner des modifications des caractéristiques. Le comportement global d'une entité est représenté par un ensemble ***d'opérations***. Chaque opération représente une part du comportement de l'entité. Chaque opération est caractérisée par un nom, ou ***sélecteur***.
Pour respecter l'hypothèse précédente, le modèle centré objets spécifie qu'une opération peut seulement modifier l'état de l'objet qui la possède. Si un objet X désire que l'état d'un objet Y soit modifié, alors la seule possibilité dont dispose X est de faire appel à une opération de Y. C'est pourquoi l'ensemble des opérations d'un objet constitue ***l'interface*** de cet objet ; c'est alors par le biais de leurs interfaces respectives que les objets échangent des informations.

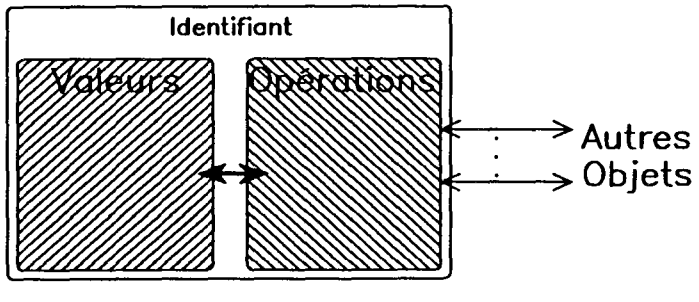


Figure 5 : Un objet

Il est couramment fait mention, dans le modèle centré objet, du terme ***encapsulation***. Un objet parfaitement encapsulé ne pourrait modifier/accéder à une de ses valeurs que par l'intermédiaire d'une opération spécifique. En fait, dans la plupart des modèles, la contrainte d'encapsulation est légèrement relâchée, une valeur d'un objet pouvant être

manipulée par n'importe quelle opération de cet objet. Dans certains modèles, même, des valeurs peuvent appartenir à l'interface de l'objet.

C.I.2. Le modèle centré objets dynamique

Il est courant de trouver, dans un modèle centré objets, la notion de ***message*** : il s'agit alors d'une requête, envoyée par un objet à un autre objet ; à chaque message est associé un sélecteur ; l'objet récepteur du message réalise alors l'opération, définie dans son interface, dont le sélecteur correspond à celui du message. De la sorte, un même message, envoyé à des objets différents, peut avoir des conséquences différentes : les comportements, décrits dans les opérations de mêmes sélecteurs, ne sont pas nécessairement identiques.
Cependant, il est exceptionnel qu'un modèle centré objets définisse plus avant la ***dynamique*** de l'ensemble des objets : en particulier, les cas d'indétermination du comportement réalisé par un objet récepteur ne sont jamais explicitement élucidés par les modèles centrés objets.

C.I.3. Un exemple de base d'objets

Cet exemple, ainsi que tous ceux qui suivront dans cet ouvrage, est extrait du contexte de ce travail, c'est-à-dire les équipements de radiodiffusion. En particulier, ces équipements électro-mécaniques sont largement composés de modules particuliers, nommés *relais*. La description de ces entités, leurs particularités et leur utilité sont donnés en annexe.

Considérons ici l'exemple d'un relais particulier. Ce relais est connu des techniciens par le nom que lui a donné le constructeur, c'est-à-dire «K37». Il s'agit donc d'un relais électro-mécanique, dont la bobine, alimentée par une tension de 220V, actionne deux contacts. Si la tension est présente aux bornes de la bobine, K37 est dit en position «Travail», sinon sa position est appelée «Repos». Lorsque K37 est en «Travail», ses contacts sont dits «fermés» sinon ils sont dits «ouverts».
Un bref inventaire fournit donc quatre entités :

- le relais lui-même
- la bobine
- le premier contact
- le second contact

Représentées dans un système centré objets, ces entités sont décrites par les objets représentés sur la figure 6.

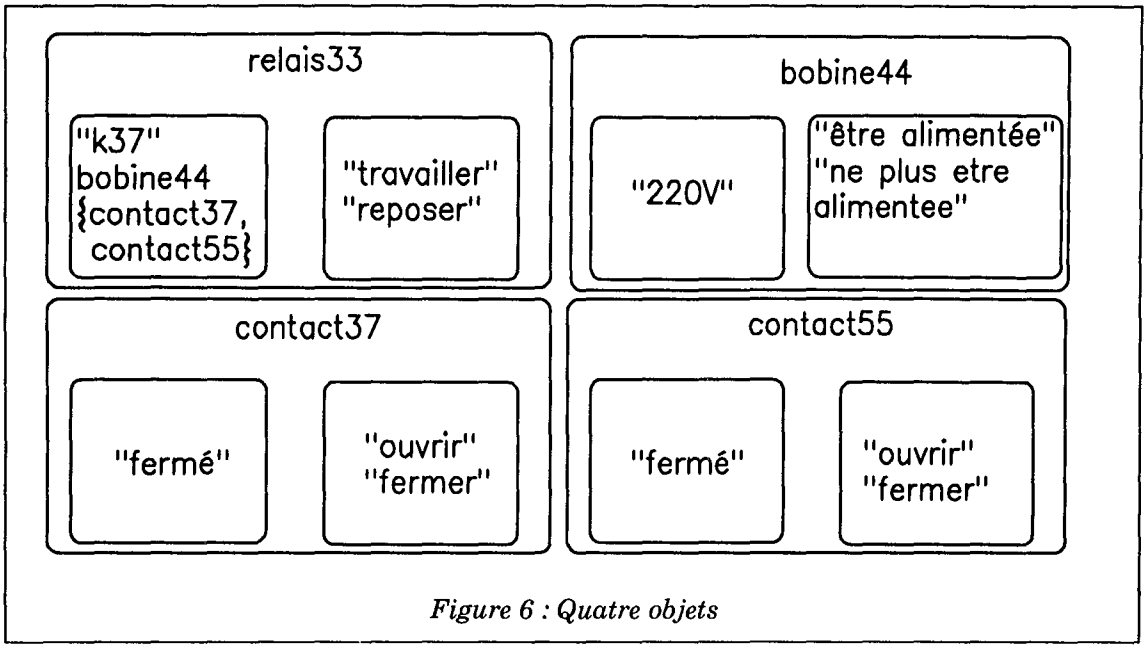


Figure 6 : Quatre objets

Les termes *relais33*, *bobine44*, *contact37* et *contact55* sont absolument arbitraires : il s'agit des identifiants des objets. L'état de l'objet *relais33* contient trois valeurs : *"K37"* est le nom du relais ; *bobine44* est une valeur qui permet de connaître quel est l'objet-bobine qui est associé au relais ; *{contact37,contact55}* est une liste de valeurs représentant les objets-contacts associés au relais : il s'agit d'une caractéristique ***multivaluée*** de l'objet *relais33*. L'interface de l'objet *relais33* comporte deux opérations, nommées *"travailler"* et *"reposer"*.
La dynamique de cette base d'objets, pour laquelle n'existe aucune convention de représentation, montrerait que l'opération *"reposer"* du relais demande à *bobine44* de *"ne plus être alimentée"* et aux objets *contact37* et *contact55* de s'*"ouvrir"*.

C.I.2. Notre avis sur le modèle centré objets

Nous pouvons noter qu'une opération formalise une partie du comportement possible de l'objet ; bien que l'objet puisse, d'une façon ou d'une autre, «exécuter» ou «réaliser» l'opération, cette description peut être considérée comme une *valeur réalisable* et par là-même être rapprochée de la notion statique de *valeur*. C'est pourquoi nous appellerons désormais ***propriété*** d'un objet indifféremment une valeur ou une opération de cet objet. Nous appellerons alors ***sémantique*** d'un objet son comportement intrinsèque, défini par l'ensemble de ses propriétés.

Nous nous accordons avec [WEGNER] sur le fait que la notion de message est loin d'être fondamentale à la définition des systèmes orientés objets. Notamment, la notion de message implique une gestion particulière par des files d'attente, des

priorités, des mécanismes de synchronisation, ... Or les modèles centrés objets incluent rarement de tels mécanismes.
C'est pourquoi nous considérons la dynamique d'un système centré objets comme un ensemble de ***déclenchements d'opérations***. La dualité message/sélecteur est alors supprimée, seul le sélecteur étant utilisé pour spécifier la propriété déclenchée. Cette dynamique est alors fondée sur une exécution séquentielle et synchrone des opérations.
Notre point de vue possède l'avantage d'unifier l'accès aux attributs et l'accès aux opérations puisque dans les deux cas, l'accès est réalisé sur le nom de la propriété.

C.II. Systèmes à base de Classes

La modélisation d'un univers réel à l'aide d'objets amène deux remarques importantes :

- il est courant que plusieurs objets possèdent des ensembles de valeurs dont la structure est identique. Ainsi, nous avons présenté dans figure 6 l'objet *relais33*, dont les valeurs sont <"K37",*bobine44*,{*contact37*,*contact55*}>. Soit un deuxième objet, *relais17*, possédant les valeurs suivantes : <"K18",*bobine13*,{*contact12*,*contact14*}>. Les valeurs qui composent *relais33* et *relais17* sont différentes, mais elles sont organisées selon un même schéma : <chaîne_de_caractères, bobine, ensemble_de_contacts>.
- deuxième remarque, il peut arriver que l'identité des entités de l'univers du discours, de même que leurs valeurs intrinsèques, présentent moins d'importance que leur existence dans la modélisation. Par exemple une modélisation exprimera le rôle et la description *des relais*, sans expliciter que *relais33* et *relais17* sont de tels objets.

C.II.1. Classes et objets

Ces deux remarques nous amènent au concept de ***classe***. Dans son acceptation la plus courante, une classe constitue une description *générique* d'un ensemble d'objets. Cette description consiste en deux parties :

- d'une part, la structure commune à toutes les valeurs de tous les objets concernés est décrite dans la classe à l'aide ***d'attributs***
- d'autre part les opérations communes à tous les objets sont définies, non plus sur ces objets, mais sur leur classe.

Dans les systèmes à base de classe, contrairement à ce que voudrait l'intuition, les mécanismes de classification ne sont pas exactement au coeur du problème. En effet, la classification consiste à considérer un ensemble d'objets et à les catégoriser en fonction de critères donnés (Cf. [NGUYEN 91], [RIEU 91]). Cette démarche peut bel et bien être appliquée afin de trouver la liste et la description des classes ; mais cette *démarche de conception* sort en réalité du cadre exact d'un modèle de données, tel qu'est le modèle à base de classes.

En fait, dans ce modèle, on considère que les classes ont déjà été définies. Alors, le modèle à base de classe définit un objet comme la *valorisation* de chacun des attributs de la classe. L'ensemble de valeurs résultant est référencé par un identifiant : jusque là, la définition de l'objet n'est pas modifiée, seul le mécanisme d'obtention des objets a été précisé. En fait, une légère modification est apportée à la définition elle-même des objets : les opérations que possède un objet sont désormais définies sur la classe, et communes à tous les objets correspondant à cette classe. Le vocabulaire couramment admis nomme ***instance*** d'une classe un objet créé à partir de cette classe. Le mécanisme de création est alors appelé ***instanciation***.

Considérons par exemple une classe qui s'appellerait RELAIS, définie de la façon suivante :

- un attribut, NOM_DU_RELAIS, doit être valorisé avec une chaîne de caractères.
- un attribut, nommé BOBINE_DU_RELAIS, doit être valorisé avec l'identifiant d'une instance d'une classe que nous nommerons BOBINE.
- un attribut, nommé CONTACTS_DU_RELAIS, doit être valorisé avec un ensemble d'identifiants d'instances d'une classe que nous nommerons CONTACT.
- Deux opérations, nommées TRAVAILLER et REPOSER, sont définies sur la classe.

Dès lors, la classe RELAIS peut être instanciée avec deux objets :

- un objet d'identifiant *relais33*, tel que :
  - . à l'attribut NOM_DU_RELAIS corresponde la valeur "K37"
  - . à l'attribut BOBINE_DU_RELAIS corresponde la valeur *bobine44*
  - . à l'attribut CONTACTS_DU_RELAIS corresponde l'ensemble {*contact37*, *contact55*}

- un objet d'identifiant *relais17*, tel que :
  . à l'attribut NOM_DU_RELAIS corresponde la valeur "K18"
  . à l'attribut BOBINE_DU_RELAIS corresponde la valeur *bobine13*
  . à l'attribut CONTACTS_DU_RELAIS corresponde l'ensemble {*contact12*, *contact14*}.

Ces deux objets, *relais33* et *relais17*, possèdent les opérations définies sur leur classe, c'est-à-dire les opérations TRAVAILLER et REPOSER.

C.II.2. Attributs

Nous avons appelé attribut d'une classe la description générique d'une valeur pour les instances de cette classe. D'autres auteurs choisissent le nom de *variable d'instance*, de *caractéristique*, de *slot*, ... Le terme d'attribut est cependant adapté à notre contexte. En effet, les attributs d'une classe possèdent souvent, dans le modèle à base de classes usuel, des ***facettes*** qui sont en fait autant «d'attributs des attributs». Or cette terminologie est relativement standard dans le contexte des langages d'acteurs. C'est pourquoi nous choisissons de la conserver dans notre modèle "standard".

Pour ce qui est des facettes des attributs, chaque auteur propose des concepts plus ou moins évolués, faisant intervenir des mécanismes de contrôle plus ou moins déterministes, ... Voici, à titre d'exemple, quelques facettes définies simplement :

- le fait que l'attribut soit monovalué ou multivalué ;
- le type des valeurs admises pour l'attribut. En particulier cette facette permet de différencier les attributs simples des attributs de relation ;
- la valeur initiale de l'attribut, à la création d'une instance.

Voici par contre des facettes qui ont été proposées, dont la manipulation est beaucoup moins aisée :

- si aucune valeur n'a été explicitement associée à un attribut, la description d'un mécanisme qui permet de calculer la valeur à retourner ;
- les conséquences de la création, de la modification ou de la suppression d'une valeur pour un attribut. Ces facettes sont usuellement nommées ***réflexes*** ou ***démons***.

C.II.3. Opérations et dynamique

Si la description des attributs est très peu normalisée, la description des opérations quant à elle ne l'est pas du tout. Il n'existe aucun consensus sur le contenu, la description ou la représentation des opérations et des interactions entre objets.

C.II.4. Modèles réflexifs

La réflexivité n'est pas spécifique du modèle à base de classes. D'ailleurs, tous les modèles proposés n'exploitent pas nécessairement cette caractéristique. Nous la présentons cependant car elle peut s'avérer importante.

Un système réflexif entretient une relation de cause à effet avec lui-même : sa structure décrit la connaissance que possède le système de lui-même. Dans le cas particulier du modèle à base de classes, la réflexivité peut être exploitée pour considérer les classes, elles-mêmes comme des instances. Les "classes des classes" sont conventionnellement appelées ***métaclasses***. Une autre application de la réflexivité consiste à considérer les attributs comme des objets ; les facettes des attributs sont alors les valeurs de ces objets et il existe une classe des attributs, possédant elle-même des attributs !

Il est évident -ne serait-ce qu'au vu des exemples précédents- que la construction d'un modèle réflexif est fort peu aisée. Cependant la réflexivité garantit ensuite que le modèle est homogène et peut être aisément étendu.

C.II.5. Notre avis sur le modèle à base de classes

Le concept de classe est au modèle des systèmes à objets ce que la notion de relation est au modèle relationnel : incontournable. La notion d'attribut ne bouleverse pas non plus le paysage de la représentation de données ; les facettes apportent cependant une aisance de représentation de l'information que des modèles antérieurs n'offraient pas. En fait, selon nous, la grande amélioration offerte par le modèle des systèmes à base de classes ne réside pas uniquement dans une capacité accrue de représentation de l'information -en particulier de l'information dynamique. L'apparition de modèles réflexifs, extrêmement maniables, autorise la manipulation -voire la déformation- des concepts du modèle tout en garantissant la cohérence du modèle.
C'est pourquoi nous nous plaçons désormais uniquement dans le contexte de modèles réflexifs.

C.III. Systèmes Orientés Objets

En introduisant le modèle des systèmes à base de classes, nous avons en fait décalé l'échelle de représentation pour passer de l'objet, modélisation fidèle d'une entité de l'univers du discours, à la classe, modélisation fidèle d'un ensemble d'entités de cet univers. Ce changement d'échelle est particulièrement sensible lorsque le modèle considéré est réflexif, puisque l'objet n'apparaît plus que comme une information secondaire, déduite (issue) du modèle des classes, des métaclasses, ...

C'est pourquoi des concepts particuliers sont nécessaires, pour manipuler non plus l'information de base -les instances- mais les informations constructives -les classes. De nombreuses relations entre classes ont été proposées, de telle sorte que l'univers réel puisse être modélisé au mieux. Parmi ces relations, nous ne pouvons manquer de citer la relation d'agrégation, qui définit l'instance d'une classe comme l'agrégation d'instances d'autres classes (l'agrégation est décrite dans [SMITH 77a]). La relation nommée *part_of*, qui indique qu'une instance d'une classe est composée d'instances d'autres classes, est également souvent proposée (notons qu'elle est très proche de la relation d'agrégation). Nous constatons cependant que ces deux relations entre classes sont en fait des descriptions génériques de relations entre objets, et affirmons qu'elles peuvent être spécifiées à l'aide d'attributs correctement typés. Par contre il existe une relation entre classes qui ne s'applique pas aux objets, mais bel et bien aux classes : cette relation est couramment nommée *is_a* ; elle permet de définir *une description* d'objets comme un *cas particulier* d'une description plus générale. Elle est bien différente des deux relations précédentes, en ceci qu'elle ne décrit aucunement une relation entre deux objets

Cette relation entre classes est nommée ***héritage***. De très nombreuses définitions ont été proposées, mais nous en dégageons deux : l'une considère l'héritage dans un contexte ensembliste tandis que l'autre définit l'héritage en fonction des propriétés attachées aux classes. Nous proposons ensuite une définition intermédiaire, tirée de [BRODIE 84], avant de discuter d'un cas particulier, *l'héritage multiple*.

C.III.1. L'héritage ensembliste

Il est possible de considérer qu'une classe est associée à l'ensemble de ses instances. Les propriétés que définit la classe existent alors pour chacune de ces

instances. L'héritage permet alors de manipuler un objet en le considérant dans des ensembles différents.

Par exemple, considérons l'ensemble des relais qui sont alimentés en 220 V, modélisés à l'aide de la classe RELAIS_220V ; considérons également les relais alimentés en 12 V, modélisés par la classe RELAIS_12V. Dans le cas où nous voudrions gérer un relais indifféremment 12 ou 220 V, nous pouvons créer une classe des RELAIS. Nous pouvons objectivement énoncer que «un relais alimenté en 220V est un relais» et que «un relais alimenté en 12V est un relais». L'héritage nous permet donc de mettre en relation la classe RELAIS avec, d'une part, la classe RELAIS_220V et, d'autre part, la classe RELAIS_12V. Cette relation nous permet de considérer les objets de l'une de ces deux classes comme s'ils étaient instances de la classe RELAIS. Par conséquent, l'héritage peut être considéré comme une réunion ensembliste. Naturellement, puisque un RELAIS_12V est un relais, toutes les propriétés des relais peuvent lui être appliquées.

C.III.2. L'héritage sémantique

L'héritage sémantique relève d'une démarche identique à celle de l'héritage ensembliste. Cependant, la relation entre classes est fondée, non plus sur l'appartenance des objets à des ensembles plus ou moins vastes, mais sur les propriétés définies sur chaque classe. En effet, l'héritage sémantique définit une relation entre deux classes A et B si l'ensemble des propriétés attachées à la classe A est inclus dans celui de la classe B : la classe B hérite alors de la classe A, les propriétés communes n'étant plus définies que sur A. On le voit, dans ce cas, l'héritage permet de définir incrémentalement une classe en fonction de classes déjà existantes : il s'agit d'un mécanisme de ***spécialisation***.

L'application de cette démarche fournit une hiérarchie de classes, de la plus générale à la plus spécifique. Or, il peut survenir que des propriétés définies à un niveau général doivent être décrites de façon plus précise à un niveau inférieur de la hiérarchie : ce raffinement de la définition d'une propriété dans la hiérarchie est appelé ***surcharge*** ou ***exception***, selon que la propriété est une opération (redéfinition de l'activité correspondante) ou un attribut (redéfinition de certaines facettes). Nous ne désirons pas prendre en compte un cas particulier de surcharge, nommé ***masquage***, qui consiste en une disparition de propriété dans la hiérarchie : le masquage met en effet en péril la cohérence de la définition de l'héritage.

C.III.3. L'héritage de façon générale

Avant de proposer une définition, précisons le vocabulaire usuellement employé. Lorsque deux classes sont mises en relation à l'aide de l'héritage, la classe la plus générale, c'est-à-dire la plus haute dans la hiérarchie, est nommée ***super-classe*** de la classe la plus spécifique. Réciproquement, la classe spécifique est nommée ***sous-classe*** de la classe générale. Il est courant, par extension, de nommer super-classe d'une classe, non ses ascendants directs, mais toutes les classes qui sont plus hautes dans la hiérarchie (réciproquement on appelle sous-classe d'une classe toute classe située plus bas dans la hiérarchie).

[BRODIE 84] propose une définition de l'héritage qui ne préjuge pas de la signification qui lui est associée, ensembliste ou sémantique. Nous retenons donc la proposition suivante, qui a le mérite d'être simple et de rester valide dans les deux cas précédents :

Une instance d'une classe est également instance de ses super-classes

Considérée selon l'héritage ensembliste, cette définition permet de considérer comme instances d'une classe la réunion de l'ensemble des objets qui instancient cette classe et des ensembles des instances de ses sous-classes.
Considérée selon l'héritage sémantique, la définition proposée associe à un objet à la fois les propriétés de sa classe, mais également toutes les propriétés héritées de ses super-classes.

C.III.4. L'héritage multiple

[CARRE 89] met en exergue l'avantage de l'aspect déclaratif de l'héritage et de la classification : un expert regroupe les connaissances de façon naturelle et les représente directement dans le Système Orienté Objets.
Cependant, dans le cas d'une approche multi-expertise, chaque expert réalise une classification en fonction de sa propre perspective : un graphe d'héritage simple peut devenir rapidement complexe lorsqu'il contient plusieurs points de vue concernant un même concept.
L'héritage multiple apporte une simplification de ce graphe d'héritage. Il permet d'associer à une classe plusieurs super-classes directes, mettant ainsi l'accent sur chaque super-classe en tant que ***point de vue***, ou ***aspect***, de la classe considérée. L'héritage multiple permet donc de créer une classe par combinaison de plusieurs autres ou, inversement, d'effectuer des factorisations multiples de classes.

Cette forme d'héritage permet une plus grande liberté d'expression mais peut créer de nouveaux problèmes. En effet, comme le montrent de nombreux travaux -plus ou moins divergents- sur la question ([DUCOURNAU 89], [DUGERDIL 88], [CARDELLI 84]), ni la sémantique, ni les mécanismes de l'héritage multiple ne sont clairement et unanimement définis.

C.IV. Notre point de vue sur le modèle orienté objets

Comme nous l'avons montré dans le chapitre précédent, les méthodes de conception destinées à la spécification de systèmes d'informations possèdent des modèles dédiés à cette activité. De ce fait, ces modèles sont souvent mal adaptés à la formalisation d'une activité indéterministe. En particulier, lorsqu'ils sont comparés au modèle orienté objets, les modèles des MCSI semblent relativement pauvres, en particulier du fait de l'absence de la notion d'héritage et de l'hypothèse de déterminisme qui est appliquée lors de la formalisation de la dynamique.

A l'opposé, le modèle orienté objets permet de spécifier n'importe quelle forme d'univers réel, sans hypothèse sur la nature ou l'utilisation des entités manipulées. L'expression de la connaissance sur l'univers réel possède alors plusieurs avantages : d'une part, le principe d'encapsulation, en imposant une très forte modularité, permet aux concepteurs de maintenir aisément les spécifications réalisées. D'autre part, la possibilité de réutiliser une spécification, intégralement ou partiellement, allège la tâche du concepteur. Enfin, le modèle lui-même permet aux objets de s'auto-documenter, rendant ainsi plus aisé le travail de conception mettant en oeuvre plusieurs personnes. Nous verrons cependant que le modèle orienté objets ne permet pas de valider une spécification.

C.IV.1. Modularité et maintenabilité

Seules les opérations d'un objet ont la possibilité, dans le modèle orienté objets, d'accéder aux valeurs internes de cet objet, ou d'en modifier l'état. Si la structure de ces valeurs est modifiée, seules ces opérations d'accès doivent être mises à jour : les autres objets n'ont pas à être modifiés, puisque l'interface de l'objet mis à jour reste extérieurement identique.
De la même façon, la modification de la structure d'une classe n'entraîne que la mise à jour des opérations d'instance de cette classe. En effet, dans le modèle

conceptuel, la structure des instances est automatiquement modifiée (Cf. [PENNEY 87], [NGUYEN 89]).
Par conséquent, le principe d'encapsulation, qui associe à chaque objet toutes les informations et règles de comportement le concernant, permet d'actualiser simplement et rapidement une spécification.

La maintenabilité est aussi accrue par l'utilisation de l'abstraction de généralisation-spécialisation dans l'héritage. En effet, en permettant d'adapter les structures de données à l'univers réel, le modèle orienté objets évite le recours à plusieurs types proches mais différents, chaque type imposant la spécification de nouvelles propriétés.

C.IV.2. Réutilisabilité

Un objet ou une classe possèdent en eux-mêmes toute leur sémantique : leurs caractéristiques, leur comportement, ... Il est donc possible d'utiliser un objet ou une classe dans un contexte différent de celui dans lequel ils ont été développés.
Nous rejoignons cependant les auteurs de [BYTE 89], qui précisent que la réutilisabilité est souvent sacrifiée en faveur d'une augmentation de la maintenabilité, lors de l'utilisation de l'abstraction de généralisation-spécialisation dans l'héritage.

C.IV.3. Lisibilité

Les détails de l'implantation étant ignorés lorsque le modèle orienté objets est utilisé, seules les interfaces des objets sont considérées. La lecture d'une description d'un objet est donc extrêmement simplifiée par le fait que cet objet décrit toutes ses propriétés : l'interface de l'objet constitue à la fois sa spécification et son mode d'emploi. Dès lors, les spécifications réalisées par plusieurs concepteurs ou plusieurs experts, comme c'est le cas lors de la conception de systèmes experts, sont lisibles par tous les membres de l'équipe de conception.

C.IV.4. Validation

A l'opposé des types abstraits qui, comme l'indique [THIERY 85], «permettent la vérification de la cohérence et de la complétude d'une spécification», le modèle orienté objets n'apporte aucun modèle de validation d'une spécification donnée. Cette absence de validation provient de l'absence de contraintes sur les objets eux-mêmes : le modèle orienté objets permet donc de spécifier n'importe quelle

réalité mais il ne permet pas de vérifier si la spécification obtenue correspond bien à cette réalité.

C.V. Conclusion

Le modèle des Systèmes Orientés Objets offre les mêmes possibilités que les modèles de SI comme celui utilisé par REMORA : décomposition d'un univers en objets, classification des objets, association d'opérations aux objets. Les différences majeures sont les suivantes :

- il n'existe pas dans le paradigme objet d'hypothèses sur l'univers modélisé, telles que le cycle Objet - Evénement - Opération de REMORA ; au contraire, le modèle des opérations est plutôt faible. Ceci permet notamment de créer des opérations qui seront indifféremment de gestion, de calcul, intermédiaires, ...
- le paradigme objet offre un mécanisme d'héritage simple et multiple qui permet une importante économie dans l'énoncé des propriétés des objets.
- le fait que toute propriété soit associée à un objet -et non indépendante, comme c'est le cas par exemple pour les opérations dans REMORA- permet la construction d'une abstraction très proche de la réalité en associant un comportement intrinsèque à chaque objet.

Considérons désormais le modèle orienté objets par rapport aux critères définis dans le chapitre A et opposons ce modèle aux méthodes de conception de systèmes d'informations :

- Les concepts d'attribut et d'opération attachés aux objets permettent de formaliser un univers complexe, composé d'entités dont la structure n'est pas toujours simple, sans poser d'hypothèses sur la nature, la fonction ou la structure de ces entités. Le modèle orienté objets s'oppose en cela aux MCSI qui, étant dédiées à des systèmes organisationnels clairement définis, sont fondées sur de telles hypothèses. Ainsi, le critère (1) peut être respecté, c'est-à-dire que l'équipement à maintenir peut être exactement modélisé.
- Corollairement, ces concepts permettent de formaliser des raisonnements non déterministes, appliqués à des objets arbitraires, qu'ils existent concrètement ou qu'ils soient abstraits, répondant ainsi au critère (4) : le modèle orienté objets peut là encore être opposé aux modèles de MCSI, dans lesquels la dynamique organisationnelle ne laisse pas de place à des raisonnements indéterministes complexes.

- Le principe d'attachement des opérations aux objets permet de «mélanger» les données (les objets et leurs attributs) aux traitements : le critère (5) est également respecté.
- Enfin, le concept d'héritage, et en particulier d'héritage multiple, permet de supposer qu'il existe des solutions pour autoriser plusieurs personnes à posséder des points de vue différents sur une même entité : le critère (2), qui n'était pas respecté dans les MCSI, peut l'être à l'aide de l'utilisation du modèle orienté objets.
- Notons que le critère (3), qui concerne l'intervention simultanée de plusieurs concepteurs, dépasse le cadre d'un modèle de connaissances pour entrer dans celui d'une démarche de conception.

Il est donc clair que le modèle orienté objets permet de respecter un plus grand nombre des contraintes imposées que les modèles des méthodes de conception de systèmes d'informations étudiés. Par conséquent, sur la thèse que le paradigme objet est mieux adapté à la formalisation des Systèmes Experts de Maintenance de Seconde Génération, nous avons décidé d'utiliser ce modèle pour créer une abstraction des mécanismes de la maintenance des équipements de radio-diffusion. Le modèle proposé est certes suffisamment riche pour permettre la construction de cette abstraction, mais il est nécessaire de lui adjoindre une démarche et éventuellement un langage de description pour autoriser la conception de logiciels de maintenance. Le chapitre suivant est une étude des méthodes de conception orientées objet existantes.

Chapitre D
Méthodes de Conception Orientées Objets

Nous avons vu que le modèle orienté objets permet de représenter la connaissance sous une forme proche de la réalité. Cependant, les méthodes de conception ne sont pas encore adaptées à cette richesse, et la plupart des MCSI qui ont été adaptées «brident» la puissance disponible en restreignant les représentations en fonction de critères issus de leurs modèles initiaux. Les méthodes purement orientées objets sont encore rares, quoique depuis 1990 il en ait été présenté un grand nombre.

Le paragraphe D.I met en évidence les différences entre les méthodes de conception orientées objets (MCOO) et les méthodes de conception traditionnelles parmi lesquelles nous considérons les MCSI. Le paragraphe D.II décrit les MCOO et montre notamment l'aspect essentiel du modèle orienté objets dans ces méthodes. Les paragraphes D.III à D.VI mettent en évidence les caractéristiques de MCOO de deux familles : d'une par, O* et J.S.D. sont des méthodes de conception traditionnelles qui ont été adaptées au modèle orienté objets ; d'autre part, les méthodes HOOD et ROME sont des méthodes qui exploitent spécifiquement le modèle orienté objets.

D.I. Comparaison des MCOO aux MCSI

D.I.1. Modèles

Nous avons vu que le but d'une méthode de conception est d'apporter une démarche et des outils pour appliquer un modèle à la construction d'une abstraction de la réalité. Dans le cas des MCSI, cette abstraction est explicitement formée d'aspects pertinents de la réalité. Au contraire, pour [MASINI 89], comme d'après [BOIS 89], la conception orientée objets consiste principalement en une ***description***, *a priori* objective, de l'univers à informatiser. C'est ensuite dans le cadre de cette description que l'expression de la solution est effectuée, en tenant compte des propriétés particulières de chaque objet. Un système organisationnel, simple instance des univers informatisables, peut donc être décrit à l'aide d'une MCOO ; les MCOO sont donc, à première vue,

plus générales que les MCSI, dont le domaine d'application est généralement limité aux organisations.
Les modèles des MCSI incluent généralement des techniques de gestion propres à ce domaine d'application : techniques de conservation automatique d'historiques, notions de différenciation des utilisateurs finals, confidentialité et sécurité des données, ... Ces techniques sont totalement absentes du modèle orienté objets. Cette absence peut être fort regrettable dans le cas d'applications orientée objets destinées à la gestion d'organisations, mais elle ne pénalise que peu les applications techniques ou scientifiques.

D.I.2. Démarches

Globalement, la démarche de conception d'un SI débute avec l'analyse des besoins des utilisateurs finals ; cette analyse aboutit à un schéma conceptuel qui résume toutes les informations apportées. A partir de ce schéma conceptuel, les problèmes d'implantation peuvent être envisagés.
Dans le cas de la Conception Orientée Objets, la démarche consiste à identifier les objets pertinents et les relations entre eux et à associer des comportements à ces objets. On aboutit, là encore, à un schéma que l'on peut appeler de même schéma conceptuel ; c'est à partir de ce schéma que l'analyse de la solution est ensuite élaborée.

Cette similitude entre les démarches des MCOO et des MCSI nous amène à considérer le problème de la COO à partir des outils existants dans le domaine des SI. En effet, ces méthodes possèdent déjà une «longue» vie et ont servi de support à de nombreux développements.

Nous notons cependant, comme [PERNICI 90], qu'une dimension supplémentaire est apportée par l'utilisation des concepts orientés objets : il s'agit de la réutilisabilité, c'est-à-dire que la spécification des besoins doit prendre en compte, dans les MCOO, le fait que des problèmes similaires sont peut-être déjà résolus dans des applications existantes.

D.II. Définition des MCOO

Nous avons défini une méthode de conception comme un quadruplet composé d'un modèle, d'un langage, d'une démarche et d'outils. Les MCOO possèdent ces quatre propriétés. Cependant, le modèle d'une MCOO en est usuellement la caractéristique fondamentale.

D.II.1. Importance du modèle

Une notion souvent associée au modèle orienté objets est l'expression «Tout est objet». Cette phrase ne possède pourtant aucune sémantique, car elle ne donne ni contexte, ni définition du «Tout» et de l'«Objet».
Cependant, la puissance de représentation du modèle orienté objets permet de décrire ses constituants à l'aide d'eux-mêmes, comme nous l'avons indiqué en C.II.4 (page 31) : c'est là le principe des ***schémas méta-circulaires*** qui définissent les systèmes orientés objets selon leurs propres concepts ; dans ces schémas, le modèle peut être exprimé à l'aide d'un ensemble de classes, chaque classe apparaissant dans le schéma étant instance d'une autre classe de ce schéma -voire d'elle même. Dès lors, une Méthode de Conception Orientée Objets peut être définie en utilisant sa propre démarche et son propre modèle. Notons que cette validation devrait être autorisée par n'importe quelle méthode de conception.

D.II.2. Langage

Les MCOO utilisent habituellement un langage graphique pour exprimer une abstraction de la réalité. La notion d'objet encapsulé, les relations entre objets, l'héritage gagnent en effet à être représentés graphiquement car toutes les propriétés d'un objet donné peuvent ainsi être perçues simplement. L'utilisation systématique des réseaux sémantiques apporte un avantage majeur (lisibilité et maintenabilité) aux schémas conceptuels qui intègrent le paradigme objet.
Les opérations associées aux objets ne sont jamais globales. Par conséquent, la portée de leurs instructions est toujours limitée à un seul objet. De ce fait, le langage textuel utilisé pour formaliser la sémantique des objets peut être syntaxiquement simple.
Reste le problème des aspects dynamiques d'une base d'objets, c'est à dire du déclenchement d'opérations inter-objets. Dans ce cas, l'utilisation d'un second réseau sémantique serait cohérente mais les graphiques obtenus risquent d'être difficiles à maintenir (typiquement, la méthode IGLOO présentée dans [BURNEAU 89] aboutit à des schémas de la dynamique inexploitables). Mise sous forme textuelle, l'analyse de la dynamique se rapproche suffisamment de la formalisation des opérations pour y être intégrée.

D.II.3. Démarche

Les caractéristiques des Systèmes Orientés Objets permettent d'adopter une démarche particulière lors de l'étape de conception. En effet, l'encapsulation des

objets en permet la réutilisation, ce qui amène les notions de bibliothèques ou de librairies à être intégrées dans l'étape de conception.

[GLASSEY 89] propose des règles particulières :
- chaque objet doit être dédié à une fonction unique
- les interactions entre objets doivent être simples, logiques, cohérentes et non ambiguës
- les classes doivent être construites de telle sorte qu'elles soient aussi réutilisables que possible.

D.II.4. Outils

Il existe actuellement encore peu d'outils appliqués à la conception par l'application du modèle orienté objets. [FLORY 89] propose un certain nombre de principes fondamentaux, relevant davantage des ateliers de conception que de l'assistance méthodologique. [JOCHEM 89] précise que les méthodes négligent souvent les aspects d'organisation et de contrôle de la construction de la base de connaissances.

D.III. La méthode O*

La méthode O*, présenté dans [CAUVET 89a] et [CAUVET 89b], est une adaptation de la modélisation proposée dans la méthode REMORA à la conception orientée objets.

D.III.1. Modèle

Le modèle de cette méthode, baptisé ***O*_modèle***, considère que tout concept apparaissant dans l'énoncé de la méthode est un objet. Parmi ces concepts, les notions d'objet, d'événement, d'opération sont assimilés à des objets ; plus encore, le modèle lui-même et jusqu'à la base de données obtenue sont aussi des objets.
Les constructeurs du modèle sont l'héritage, l'agrégation et le regroupement (qui correspond à l'association). Un schéma méta-circulaire basé sur ces constructeurs prédéfinit un ensemble de types d'objets (domaine, entité, événement et action) et un ensemble de structures d'objets (agrégat, collection, énuméré ou simple).
Le comportement des objets est basé sur les concepts d'action et d'événement, de la même façon que dans la méthode REMORA.

D.III.2. Langage

Le schéma conceptuel possède une «syntaxe» de base relativement simple (figure 7). Trois arcs représentent l'héritage (flèche simple), l'agrégation (flèche simple à plusieurs points de départ) et le regroupement (flèche double).
Les schémas dynamiques sont identiques à ceux de REMORA et possèdent donc le même handicap pour ce qui est des dynamiques complexes : le problème crucial de la lisibilité subsiste.

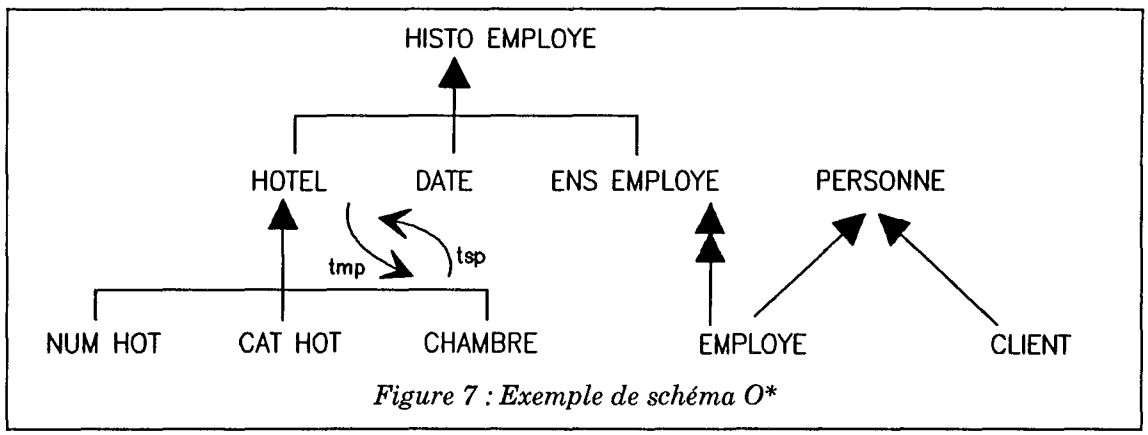


*Figure 7 : Exemple de schéma O**

D.III.3. Démarche

Les étapes de la méthode O* sont au nombre de trois : inventaire des objets du monde réel, structuration des objets et affinement.
La première étape aboutit à un schéma statique pour chaque objet et à un schéma statique intégrant l'ensemble des objets ; la seconde modifie ces schémas pour les rendre "bien structurés" ; la troisième étape permet la hiérarchisation des types, la validation de la complétude des schémas et la création des inventaires (spécifications).
Durant ces étapes, les contraintes de cardinalité sur les relations qui existent entre les objets font l'objet d'un traitement formel qui aboutit nécessairement à une structure standard d'héritage, d'agrégation et de regroupement.

D.III.4. Adéquation aux SEM2G

La simplicité des O*_schémas, comparée à la complexité des équipements techniques qui doivent être intégralement formalisés, laisse craindre que la représentation de la connaissance profonde à l'aide de O*_schémas soit complexe. En particulier, le fait que les relations statiques entre objets ne puissent être exprimées qu'à l'aide de trois concepts (héritage, agrégation ou regroupement) est beaucoup moins pratique que le concept de référence entre objets présent dans le modèle orienté objets : à l'aide de ces références, donc d'attributs auxquels il

attribue un nom, un concepteur possède une grande liberté d'expression sémantique de la relation entre les objets. Ainsi, pour formaliser le fait qu'un module électro-mécanique contient des composants, le modèle orienté objets permet de créer des attributs duaux, nommés CONTIENT et CONTENU_DANS, attachés l'un au module et l'autre au composant ; les noms donnés à ces attributs en expriment la sémantique. Par contre, dans O*, ce concept «contient/contenu» ne peut être directement formalisé, puisqu'il doit tout d'abord être assimilé à une relation d'agrégation, avec la perte de sémantique correspondante.

Le problème inverse se présente pour les schémas dynamiques : de même que pour REMORA, le modèle dynamique Objet-Evénement-Action s'applique difficilement à la représentation de connaissances de diagnostic.

D.IV. La méthode J.S.D.

[JACKSON 89] propose une description partielle de la méthode systémique J.S.D. (Jackson Software Development), se concentrant sur les aspects pertinents pour une comparaison avec le modèle orienté objets ; la démarche de cette méthode est décrite en détail dans [FILLOQUE 89] et [PAROT 88].

D.IV.1. Modèle

Dans J.S.D., le modèle des données est explicitement séparé du modèle des traitements : le premier est une abstraction de l'univers réel tandis que le second est composé de l'ensemble des informations de contrôle et de sortie du système conçu. Le modèle de J.S.D. s'appuie essentiellement sur le concept de fonction, la description du monde réel étant réalisée en termes d'ordonnancement des événements dans le temps ; le modèle des données est dérivé de cette description et n'en est jamais indépendant : les entités sont toujours définies en même temps que leurs fonctions. Le modèle de J.S.D. est donc fondé sur une définition très détaillée des processus asynchrones. D'après [JACKSON 89], un processus peut être assimilé à un objet du modèle orienté objets : un processus possède un état local, qui peut être modifié uniquement par le texte de ses opérations. En d'autres termes, les objets, dans le modèle de J.S.D., ne sont définis que par leur partie dynamique. L'encapsulation définie dans le modèle orienté objets n'est pas respectée dans J.S.D. (tout processus peut consulter l'état des autres processus). Corollairement, le concept de classe est déformé pour correspondre à la définition proposée de l'objet : les classes ne sont pas organisées en hiérarchies d'héritage -ce concept n'existe pas-, mais partagent entre elles des «rôles» (des processus), un rôle n'étant pas attaché à une classe particulière.

D.IV.2. Démarche

Une première étape est destinée à modéliser l'univers réel. A partir des spécifications obtenues auprès des utilisateurs finals, cette étape aboutit à une liste d'actions et d'entités valides, puis à un diagramme de structure qui présente la vie de chaque entité.
Une seconde étape permet la modélisation du système par assemblage des différentes entités définies dans un réseau de processus communicants incorporant toutes les fonctionnalités du système final. C'est dans cette étape que sont définis les rôles des entités dans chaque processus.
La troisième étape de J.S.D. concerne l'implémentation des diagrammes obtenus.

D.IV.3. Langage

Le langage de représentation de J.S.D. est une convention graphique fondée sur la représentation des processus sous forme de graphes. De nombreux attributs graphiques portent à la fois sur les arcs du graphe (expression du parallélisme, des mécanismes de question-réponse, ...) et sur la représentation des concepts (une fonction pourra être représentée par un losange, un cercle, un cercle double, (...) selon les différents schémas de communication auxquels elle appartient).

Ainsi, la figure 8 montre un schéma de structure formalisant un auto-commutateur téléphonique. Les rectangles mettent en évidence les différents processus nécessaires ; en particulier UN_GENERATEUR est un processus interne qui a pour charge de générer toutes les actions propres au système (sonnerie, connexion, raccrochage distant, ...) De plus, le processus UN_FILTRE a pour rôle d'empêcher, par exemple, d'envoyer un message DEBUT_SONNERIE à UN_TELEPHONE en cours de conversation. Les flots de données entre les processus sont les suivants :

- DS1 est le flot des actions simulées, générées par le système
- DS2 est le flot des actions communes à UN_TELEPHONE et à UN_APPEL
- DS3 est la demande de génération d'une action interne
- DS4 est la réponse rendue par UN_FILTRE à UN_GENERATEUR lorsque ce dernier envoie un message à UN_TELEPHONE.
- DS5 est le flot des actions demandées par UN_GENERATEUR à UN_RESEAU.
- DS6 est la réponse de UN_RESEAU aux demandes de UN_GENERATEUR.

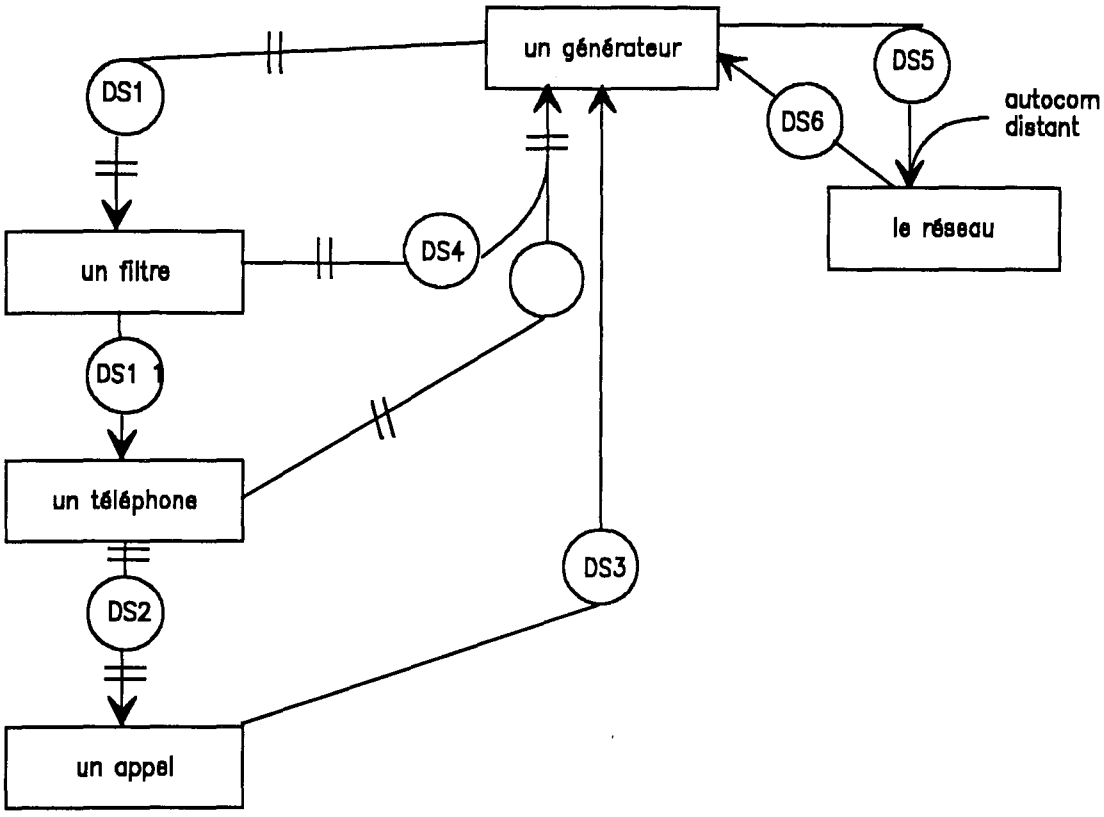


Figure 8 : Exemple de schéma J.S.D. représentant un autocommutateur

D.IV.4. Application aux SEM2G

De même que pour la méthode traditionnelle SADT, le principe de la décomposition fonctionnelle est très mal adapté à l'analyse de produits fondés autant sur la modélisation d'un univers réel que sur la formalisation de raisonnements experts. L'importance donnée aux fonctions dans J.S.D. nuit donc à l'adéquation de cette méthode pour l'analyse des systèmes experts. D'ailleurs [JACKSON 89] affirme explicitement que «la méthode J.S.D. n'est pas destinée à gérer des abstractions mathématiques telles que des angles, des nombres, des polygones et des ensembles», ce qui illustre clairement son inadaptation aux problèmes posés dans le contexte de l'informatique technique.

D.V. La méthode HOOD

La méthode HOOD, décrite dans [HEITZ 87] et [LAI 89], est fondée sur la méthode OOD (Object Oriented Design) proposée par [BOOCH 86], à laquelle elle ajoute le concept de hiérarchisation (d'où son nom, Hierarchical Object Oriented Design). HOOD est une méthode de conception descendante orientée objets, supportée par une notation proche de ADA.

D.V.1. Modèle

Le modèle de HOOD est naturellement fondé sur une formalisation du concept d'objet. Considéré dans son aspect statique, un objet est défini classiquement par un état et une interface. Considéré dynamiquement, un objet possède des opérations qui peuvent s'effectuer séquentiellement ou de façon concurrente ; dans ce deuxième cas, le protocole de l'interaction des flots de contrôle utilise la notation et la sémantique ADA du rendez-vous. Ce modèle dynamique impose la gestion ***d'exceptions***, levées par les objets lorsque leurs opérations sont achevées avec succès.

Le concept de classe dans OOD est standard, c'est-à-dire que la classe est une unité générique qui permet de produire des objets qui possèdent ses caractéristiques. Par contre, dans HOOD, le concept de classe est beaucoup moins général, la classe étant vue essentiellement comme un ensemble (liste, table) d'objets, chaque objet possédant les mêmes caractéristiques que les autres objets définis dans la même «classe» : le concept de classe est fortement amoindri et, dans certains articles, il n'est même pas mentionné [HEITZ 87].

Corollairement, le concept d'héritage tel qu'il est défini dans le modèle orienté objets standard n'existe pas dans HOOD. Par contre, cette méthode définit une relation entre les objets, nommée ***Include*** (dans [HEITZ 87]) ou ***Son-Of*** (dans [LAI 89]), qui «permet de décomposer un objet en un ensemble d'objets enfants qui collectivement fournissent les mêmes fonctionnalités que le parent. Une opération du niveau parent peut être implémentée par une ou plusieurs opérations sur les enfants ; dans ce dernier cas un enfant spécifique, du nom de l'opération sur le parent, contrôle les opérations sur les autres enfants.» On le voit, cette représentation n'équivaut en rien à la relation d'héritage. Si elle présente certains avantages, comme nous le verrons en D.V.5, nous observons que sa mise en oeuvre est relativement complexe.

Le modèle de HOOD inclut également une description purement dynamique des objets, alors appelés ***objets actifs***. La relation ***Include***, ainsi qu'une relation ***Uses***, sont définies sur les objets actifs : la première définit le flot de contrôle des opérations à l'intérieur d'un objet, tandis que la seconde décrit les interactions et les exceptions qui peuvent exister entre deux objets.

D.V.2. Langages

Un schéma conceptuel graphique représente les objets passifs, dont les opérations sont exécutées séquentiellement et les objets actifs, qui agissent de façon autonome. Ce schéma conceptuel est doublé -et complété- par une

description textuelle des objets. Cette description textuelle fait en particulier intervenir, pour la description des opérations, une syntaxe de type ADA.

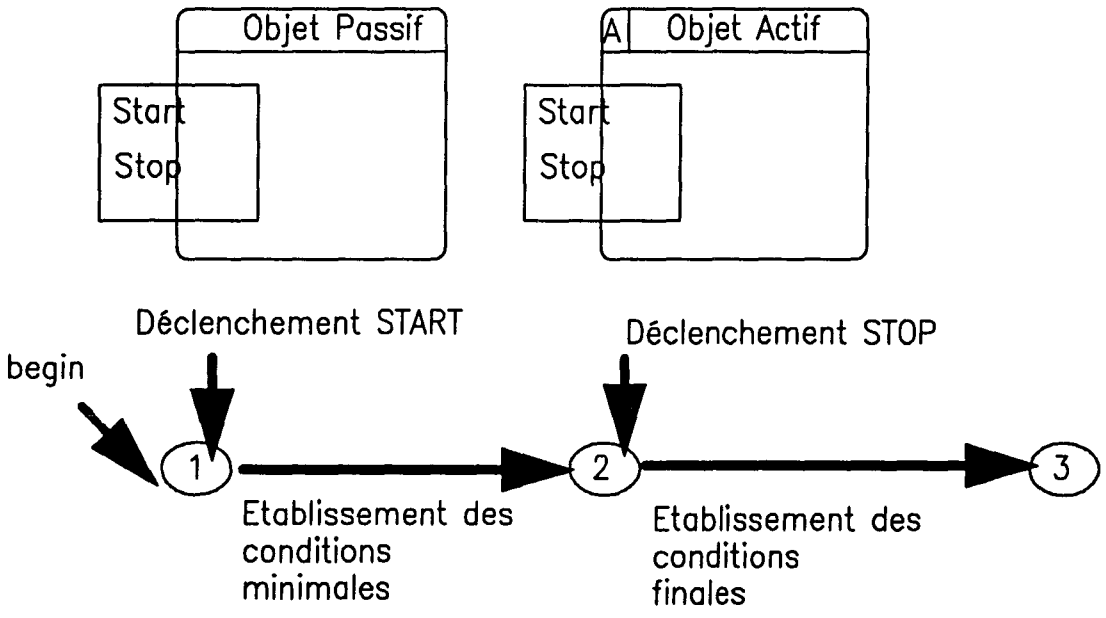


Figure 9 : Exemple de schémas HOOD

D.V.3. Démarche

HOOD n'est pas une méthode d'analyse : cette méthode ne peut être appliquée qu'après des étapes d'analyse et de spécification fonctionnelle. Elle permet alors la conception architecturale et détaillée des composants des gros logiciels spécifiés. La démarche de conception est descendante, c'est-à-dire que la conception est d'abord générale, puis de plus en plus détaillée. Au niveau le plus fin, les produits de la conception ressemblent fort à une implantation en ADA.
Chaque étape de conception est fondée sur la relation Include entre objets parents et objets enfants. Après définition et analyse du sous-problème à résoudre, une stratégie informelle de résolution est élaborée, menant à un premier ensemble d'objets, d'opérations et de contraintes. Cette stratégie est ensuite formalisée par l'identification des objets utiles de leurs attributs et de leurs opérations, ce qui aboutit à une représentation graphique. Enfin, les interfaces des objets sont formalisées, de même que les descriptions des objets parents, aboutissant à des «squelettes» d'unités de code dans le langage cible.

D.V.4. Outils

[LAI 89] présente un outil de développement, réalisé sur Macintosh à l'aide du logiciel HyperCard. Cet outil supporte tous les concepts de HOOD et a pour but de fournir une documentation standard adaptée au formalisme de la méthode.

HyperHood consiste en un éditeur graphique, un éditeur de textes et une base de données de types ADA.
De plus, une variante de HyperHood, nommée HyperHood++, permet d'unifier les concepts de HOOD et du modèle orienté objets en apportant à la première les notions d'héritage et de classe telles qu'elles sont définies dans le second.

D.V.5. Application aux SEM2G

Certaines critiques sont exprimées par [LOTT 90] ; elles concernent essentiellement le manque de souplesse dû au fait que HOOD est très proche du langage ADA. Nous ne pouvons effectivement que déplorer cet état de fait, car les spécifications obtenues avec HOOD peuvent difficilement être implantées dans un autre langage.
Nous constatons que la méthode HOOD s'éloigne largement du modèle orienté objets standard, à un point tel que des outils particuliers sont nécessaires pour unifier les deux modèles. Or, nous avons montré dans le chapitre C les avantages de l'utilisation du modèle standard lors de la formalisation d'un univers réel. Aussi devons-nous étudier l'impact de la perte des concepts de classe et d'héritage, ainsi que les avantages liés à l'utilisation de la relation Include, pour déterminer si HOOD peut être utilisée pour concevoir un SEM2G :

- le concept de classe, en tant qu'unité générique de description d'un ensemble d'objets, permet de ne pas prendre en compte chaque cas particulier des entités réelles, mais de les décrire globalement. Considérée dans la méthode HOOD, la classe n'est plus qu'un ensemble, l'objet reprenant son importance initiale. La suppression du concept de représentation générique d'un ensemble d'objets nous semble préjudiciable à la facilité de formalisation d'un univers réel.
- le concept d'héritage permet de construire incrémentalement des définitions de classes. L'absence de ce concept dans HOOD alourdit considérablement le processus de construction des classes en imposant une redéfinition complète de leurs ensembles de propriétés.
- par contre, la relation Include répond parfaitement au critère (2) mentionné dans le chapitre A. En effet, la possibilité de représenter un objet comme une composition de plusieurs objets différents, possédant des interfaces spécifiques, autorise la représentation de plusieurs domaines de connaissance dans ces objets enfants. Notons cependant que la relation Include intervient fondamentalement dans la démarche descendante de HOOD, chaque transition vers un niveau de conception plus détaillé

donnant naissance à un nouvel ensemble d'objets enfants, ce qui ne correspond plus à nos besoins d'expertises multiples.

Notons également que HOOD est particulièrement orientée vers le langage ADA à l'exclusion de tout autre, même à un niveau conceptuel élevé. Il est probable qu'une implantation dans un autre langage présenterait beaucoup de difficultés liées à la traduction, non seulement syntaxique, mais aussi conceptuelle, des notions qui sont propres à ADA.
Enfin, HOOD a pour objectif la conception de gros logiciels temps réel. Or, nous avons mentionné dans le chapitre A que, le diagnostic étant différé par rapport au fonctionnement de l'équipement, il n'est pas utile de *simuler* celui-ci. Dès lors, une part importante du contrôle de la dynamique proposé par HOOD devient inadaptée pour la réalisation d'un SEM2G.

D.VI. La méthode ROME

ROME est proposée dans [CARRE 89-90a-90b]. Il ne s'agit pas à proprement parler d'une méthode de conception orientée objets, mais d'une méthode de représentation d'objets multiple et évolutive : par conséquent c'est le modèle de ROME qui présente une importance primordiale, les auteurs ne proposant pas de méthode de conception. Nous avons cependant inclus ce modèle de données dans ce chapitre car il correspond à nos préoccupations en matière de modélisation orientée objets.

D.VI.1. Modèle

ROME est défini comme un modèle méta-circulaire initialement basé sur le modèle orienté objets standard. Les particularités suivantes sont ensuite introduites :

- les opérations sont encapsulées dans l'objet et n'appartiennent pas directement à son interface. C'est un concept de ***sélecteur***, c'est-à-dire une association (nom de message - opération), qui définit l'interface des objets. De la sorte, des opérations peuvent être définies sur un objet sans appartenir à son interface : elles ne pourront être manipulées que par l'objet, de même que les attributs.
- le modèle inclut explicitement le concept de classe abstraites.
- une stratégie graphique d'héritage multiple sans conflits permet la notion de ***point de vue*** sur une classe : plusieurs formalisations d'une même classes peuvent exister concurremment.

La notion de représentation multiple et évolutive est fondée sur une modification de la notion de base des modèles orientés objets, l'instanciation d'une classe et d'une seule pour un objet donné. Trois sortes de classes sont proposées : les ***r_classes*** sont des classes de représentation non instanciables (classes abstraites) ; les ***i_classes*** sont des ***r_classes*** qui savent s'instancier selon le modèle qu'elles définissent en tant que ***r_classes*** ; une classe particulière, ***Object***, est la classe la plus générale, dont tout objet est représentant.

D.VI.2. Langage

Les constructeurs sont proposés sous forme d'expressions LISP mais il n'existe pas de langage conceptuel.

D.VI.3. Démarche, outil

ROME ne contient pas de démarche méthodologique de conception de bases d'objets. A fortiori il n'existe pas d'outils de conception. Par contre, un outil de programmation, fondé sur la représentation méta-circulaire de ROME, permet l'implantation, au travers d'une interface graphique dédiée, des concepts proposés dans ce modèle.

D.VI.4. Application aux SEM2G

Le modèle ROME est un modèle de données et ne permet pas la conception d'un SEM2G. Cependant, dans ce modèle, plusieurs concepts rejoignent les préoccupations de la conception de systèmes experts. Ces concepts peuvent donc être inclus, directement ou indirectement, dans le modèle de données d'une méthode de conception dédiée.

D.VII. Conclusion

Ce chapitre ne présente pas un panoramique complet des méthodes de conception orientées objets, mais se focalise sur certaines méthodes ou adaptations de méthodes fondées sur le modèle orienté objets : nous pouvons citer, parmi celles que nous n'avons pas explicitement traitées, la méthode OOSD (Object Oriented Structured Design de [WASSERMAN 89-90], la méthode MCO de [CASTELLANI 91] ...

Les méthodes de conception orientée objets que nous avons analysées ne permettent pas, individuellement, de réaliser de façon pratique la conception d'un système expert de seconde génération. Cependant, le modèle orienté objets présente de nombreux avantages pour ce qui est de la modélisation d'un univers réel. Aussi retenons-nous ce modèle, que nous envisageons d'utiliser pour créer

une méthode de conception dédiée aux SEM2G. Le chapitre suivant récapitule les avantages et les inconvénients des quatre méthodes étudiées et pose les bases d'une nouvelle méthode.

Chapitre E
Récapitulatif et Conclusion

Les tableaux présentés en page 56 mettent en évidence les avantages et les inconvénients que nous avons cités pour chacune des méthodes étudiées. A partir de ce tableau, nous pouvons définir certaines caractéristiques que doit posséder une méthode de conception dédiée à l'élaboration de systèmes experts de seconde génération.

E.I. Le modèle

Le premier tableau présente, pour chaque méthode, l'adéquation du modèle proposé à la modélisation d'un équipement, à la modélisation d'un raisonnement de diagnostic et, dans la troisième colonne, l'existence de concepts qui permettent à plusieurs concepteurs de travailler simultanément. Seul le modèle de la méthode HOOD correspond à nos trois critères ; rappelons cependant que ce modèle est relativement éloigné d'un modèle orienté objets «standard».
Le modèle de la méthode envisagée doit permettre de formaliser des connaissances techniques, en particulier pour ce qui est de la représentation statique du fonctionnement de l'équipement à maintenir. Ayant retenu le modèle orienté objets, nous devons nous attacher à nous en éloigner aussi peu que possible. Les critères présentés au chapitre A ainsi que l'étude des méthodes existantes nous amènent à prendre en compte la notion de point de vue, ou d'aspect, de telle sorte que plusieurs concepteurs puissent observer de façon différente un même objet. De plus la notion d'évolutivité des objets semble ouvrir de nouveaux horizons à la modélisation en permettant de formaliser statiquement des entités dont l'aspect change au cours du temps.

E.II. Le langage

Le second tableau présente, pour chaque méthode, l'existence d'un langage graphique et d'un langage textuel et juge de la lisibilité des langages proposés. Toutes les méthodes observées proposent un langage conceptuel fondé sur une représentation graphique. Il semble en effet évident que ce type de représentation est plus maniable qu'un texte. Une méthode de conception de SEM2G doit donc posséder elle aussi un langage conceptuel graphique. Deux

alternatives s'offrent alors : si le langage graphique est suffisamment simple, comme c'est le cas pour SADT, une représentation textuelle apporte des commentaires sur des schémas qui sont suffisamment clairs en eux-mêmes. Si par contre les représentations graphiques sont trop complexes, une formalisation textuelle permet d'affiner la compréhension des schémas et est indispensable à l'expression de la sémantique de ces schémas.
Il nous semble préférable de proposer un langage simple, aux composantes uniformes et intuitives, qui ne soit pas doublé d'une représentation textuelle. Nous constatons à nouveau que les langages proposés dans HOOD, formellement définis, correspondent à cette définition. Nous avons cependant noté que les spécifications obtenues sont très proches de ADA et par conséquent difficilement exploitables si l'étape physique de la conception est réalisée à l'aide d'un autre système orienté objets.

E.III. La démarche

Le troisième tableau présenté en page 56 présente la capacité qu'offre chaque démarche à formaliser les équipements électro-mécaniques puis le raisonnement, avant de préciser si ces démarches prennent en compte la conception simultanée par plusieurs intervenants. La démarche constitue généralement le coeur des méthodes. Nous retenons en particulier la démarche de REMORA, qui inclut un grand nombre de règles de normalisation des schémas. Nous considérons cependant que ces règles devraient être appliquées dès la création d'un schéma, plutôt que lors du passage d'un schéma à un autre. Nous notons que les méthodes étudiées ne possèdent en général une démarche adéquate à la conception de SEM que lorsque le modèle lui-même est adéquat, ce qui semble normal pour autant que la démarche d'une méthode soit fondée sur son modèle. De plus, les méthodes présentées, à l'exception de HOOD, n'incluent pas dans leur démarche méthodologique le fait que plusieurs concepteurs doivent travailler simultanément et indépendamment l'un de l'autre, ce qui constitue une contrainte forte de la conception de SEM2G. HOOD, par contre, étant destinée à la conception de gros logiciels techniques, prend en compte les contraintes imposées par la conception d'un SEM2G.

E.IV. Les outils

Quels que soient le modèle, le langage et la démarche choisis, un outil informatique d'assistance à la conception permet d'assurer, de façon peu ou prou automatique, le respect des règles de cohérence et la maintenance des schémas conceptuels. La méthode proposée doit donc être associée à un tel outil.

E.V. Conclusion

Cette étude des méthodes de conception, ainsi que la comparaison entre les méthodes orientées Systèmes d'Informations et celles orientées objets, ne saurait être complète dans la mesure où elle ne considère que quelques modèles de conception. Il est en effet difficile de fournir une liste exhaustive des méthodes existantes. Au contraire, nous préférons nous référer aux nombreuses publications sur ce sujet. Ainsi, [ALAGIC 89] compare un modèle orienté objets à un modèle relationnel tandis que [BIRAMA 89] ou [JAULENT 90] proposent des analyses comparatives de plusieurs méthodes.
De même, il existe un nombre important de méthodes de conception dédiées au modèle relationnel : citons MERISE [LE 82-85], IDA [BODART 85], ou AXIAL [PELLAUMAIL 85-86] ...

La conclusion de cette étude préliminaire nous amène à considérer HOOD comme la seule méthode étudiée qui soit adaptée à la conception de systèmes de maintenance de seconde génération. Cependant, nous avons constaté dans cette méthode plusieurs points gênants ; en particulier, son modèle est suffisamment éloigné du modèle orienté objets -et suffisamment proche du langage ADA- pour empêcher que tous les concepts en soient exploités. C'est pourquoi, sur la base de cette étude des méthodes de conception existantes, la partie II présente une méthode de conception orientée objets, créée spécifiquement pour la réalisation de systèmes experts de seconde génération.

Modèle

	Modélisation de l'équipement	Modélisation du raisonnement	Multi-Concepteurs
SADT	oui et non (pas de règles de cohérence)	non (complexe)	oui
REMORA	difficile (pas de SI)	non (complexe)	non
Modèle OO	oui, par définition	oui (opérations)	non
O*	oui (concepts objets minimaux)	non (complexe)	non
JSD	non (trop d'importance à la fonction)	non (complexe)	non
HOOD	oui (concepts éloignés du modèle OO)	oui et non (temps réel)	oui (relation Children)
ROME	oui (modèle OO)	oui (modèle OO et évolution)	oui (concept particulier)

Langage

	Graphique	Textuel	Lisibilité des schémas
SADT	oui (bien défini)		très bonne
REMORA	oui (bien défini)	oui (dynamique)	facilement réduite
Modèle OO	*pas de langage*----------	----------	----------
O*	oui	oui	facilement réduite
JSD	oui (complexe)	non	faible
HOOD	oui (bien défini)	oui (bien défini)	beaucoup de concepts
ROME	*pas de langage*----------	----------	----------

Démarche

	Modélisation de l'équipement	Modélisation du raisonnement	Multi-Concepteurs
SADT	oui (bien définie)	non	non
REMORA	oui (règles)	non (trop complexe)	non
Modèle OO	*pas de démarche*----------	----------	----------
O*	oui	non	non
JSD	oui (mais par ses fonctions)	oui (toujours par les fonctions)	non
HOOD	oui	oui	oui
ROME	*pas de démarche*----------	----------	----------

Partie II
Spécification et Conception Orientées Objets de SystèmeS Experts (SECOOSSE)

Nous avons étudié, dans la partie précédente, deux grandes familles de méthodes de conception : les MCSI et les MCOO. Les premières sont basées sur des modèles plus ou moins "personnels" ou adaptés, souvent issus du modèle Entité-Relation ou du modèle relationnel pour ce qui est du modèle de données, tandis que les secondes trouvent leur unité dans le modèle orienté objets.
Nous avons conclu de cette étude que les modèles et démarches présentés s'adaptent difficilement au contexte des Systèmes Experts de Maintenance de Seconde Génération. Nous avons observé de plus que le modèle orienté objets est adapté à la création d'outils logiciels dans ce domaine.
Par conséquent nous en sommes arrivé à la conclusion que nous devions développer notre propre méthode de conception, fondée sur le paradigme objet, qui n'apporte que peu de contraintes -pour permettre la représentation de n'importe quelle forme de connaissance profonde- tout en intégrant un modèle suffisamment riche : nous envisageons effectivement des problèmes de conception de logiciels importants, mettant en oeuvre plusieurs concepteurs simultanément ; de plus la représentation de la connaissance d'équipements de radiodiffusion impose de pouvoir représenter non seulement des structures de gestion, mais aussi des structures mathématiques et des traitement complexes.
Nous proposons ainsi une méthode nommée *Spécification et Conception Orientée Objet de SystèmeS Experts* (SECOOSSE), fondée sur un enrichissement du modèle orienté objets. Le chapitre A présente le modèle de SECOOSSE. Le chapitre B en présente le langage, qui permet la formalisation d'un univers réel, du problème posé et de la solution apportée. Ce langage est fondé sur des composantes graphiques, sémantiquement riches et proches des habitudes des utilisateurs de la méthode. Sur la base du modèle et du langage de SECOOSSE, la démarche présentée dans le chapitre C est relativement intuitive. Elle se rapproche, elle aussi, des réflexes de conception acquis par les informaticiens.

Modèle, langage et démarche sont ensuite utilisés pour produire un schéma conceptuel exprimant le modèle lui-même de SECOOSSE, exprimé à l'aide du langage graphique. Ce méta-schéma est présenté dans le chapitre D.

Enfin, le chapitre E présente un outil, nommé SEISME, associé à la méthode SECOOSSE. Cet outil a été réalisé dans le même esprit que les systèmes de maintenance : à partir d'une réalité -les concepts de SECOOSSE- et de connaissances détenues par des intervenants humains -les créateurs de SECOOSSE, plusieurs domaines de connaissance ont été dégagés et formalisés.

Dans ce contexte, l'outil SEISME a été conçu uniquement à l'aide des composantes de SECOOSSE.

Chapitre A
Le modèle de SECOOSSE

Le modèle de SECOOSSE est fondé sur le modèle orienté objets présenté dans la première partie. Ce modèle de base est enrichi de notions nouvelles, rendues indispensables par deux aspects majeurs de la conception d'applications : les données manipulées ne possèdent pas toujours une structure fixe, statique au cours du temps ; par conséquent il est nécessaire de prendre en compte l'évolution des entités de la réalité dans une modélisation. Par ailleurs, les modélisations ne sont pas systématiquement utilisées dans un seul but ; par conséquent, la collaboration de plusieurs spécialistes est indispensable pour la création de logiciels puissants - citons, dans le domaine des SEM2G, des spécialistes des équipements de radiodiffusion, de l'interface graphique ou du raisonnement informatique : le modèle de notre méthode doit être enrichi pour inclure la description multi-concepteurs de données.

Après une brève introduction, nous présentons en A.II le langage prédicatif qui sera utilisé pour exprimer formellement le modèle de SECOSSE. Le paragraphe A.III présente ensuite l'adaptation du modèle orienté objets à notre méthode de conception ; le paragraphe A.IV présente alors une déformation de ce modèle, afin que soient prises en compte les entités de la réalité dont l'ensemble des caractéristiques n'est pas figé au cours du temps. Le modèle de base alors obtenu est enrichi en A.V par la notion d'héritage, décomposée selon plusieurs formes : nous spécifions tout d'abord la sémantique de la factorisation et de la spécialisation simple. Puis nous spécifions la notion de spécialisation multiple, destinée à permettre à plusieurs concepteurs de travailler simultanément et indépendamment sur une modélisation unique de la réalité. Le paragraphe A.VI explique ensuite pourquoi nous ne proposons pas de modèle de la dynamique des objets. Enfin, le paragraphe A.VII présente sous une forme condensée une synthèse des prédicats qui définissent le modèle de SECOOSSE.

A.I. Introduction

Ce chapitre définit précisément le modèle de SECOOSSE. Il est usuel, lors de la présentation d'un modèle, d'en définir exactement la sémantique ; en particulier, le modèle doit définir les ***mécanismes*** qui l'animent. Cependant, nous ne proposons pas le modèle d'un nouveau système orienté objets, mais celui d'une méthode de conception. Notre modèle doit donc permettre à l'utilisateur de formaliser un univers ; par contre, il est inutile que ce modèle exprime les mécanismes qui permettent de «réaliser» cette formalisation, car c'est là le rôle du langage de programmation qui sera ensuite utilisé.

C'est pourquoi le modèle qui est présenté dans ce chapitre est un modèle structurel, en ce sens qu'il décrit les relations entre concepts, mais qu'il n'explique pas la sémantique des mécanismes. Les relations entre concepts sont décrites statiquement, mais leurs propriétés axiomatiques -ce qu'une relation implique en termes de réécriture sur les autres concepts ou relations- ne sont pas présentées.

Un modèle sémantique doit s'intéresser aux mécanismes, donc à ses pré et post-conditions de fonctionnement ; cette étude amène nécessairement à un choix quant au ***contexte de typage***. En effet, le fonctionnement d'un système orienté objet n'est garanti -donc le modèle sémantique n'est valable- que si l'on considère un ***typage fort*** : tout doit être fixé et connu à l'avance. Au contraire, notre modèle structurel définit les caractéristiques de chaque entité au moment où il en est besoin : le problème du contexte de typage ne se posera que lors de l'utilisation d'un langage.

A.II. Langage prédicatif

La description du modèle de SECOOSSE est exprimée en langue naturelle et est illustrée par un langage *prédicatif*. L'intérêt du langage prédicatif réside dans la possibilité d'exprimer des faits en dehors de toute interprétation.

A.II.1. Calcul des prédicats du premier ordre

Le calcul des prédicats du premier ordre a pour but de formaliser la notion de formule logique avec variables et quantificateurs. Ce calcul est du premier ordre car les variables peuvent prendre comme valeurs les éléments d'un ensemble simple, par opposition au calcul des prédicats du second ordre, dans lequel les valeurs peuvent elles-mêmes être des prédicats.

Un sous-ensemble de l'ensemble des prédicats du premier ordre est appelé l'ensemble des *clauses de Horn*. L'algèbre sur les clauses de Horn consiste en une simplification de l'algèbre des prédicats du premier ordre, en particulier parce que la forme des clauses de Horn est fortement contrainte par rapport à celle des formules du langage des prédicats du premier ordre. De plus, certains langages informatiques, en particulier Prolog, sont fondés sur l'algèbre des clauses de Horn. Or, comme nous le verrons dans la troisième partie de ce rapport, c'est ce langage qui a été retenu pour la réalisation des maquettes et prototypes. Nous avons par conséquent retenu le langage des clauses de Horn pour exprimer notre modèle conceptuel afin de rapprocher, autant que possible, l'expression théorique de l'implantation pratique.

A.II.1.a) Vocabulaire du calcul des prédicats du premier ordre

On appelle ***constante*** tout atome du langage. Appelons $\mathcal{C}$ l'ensemble des constantes.

On appelle $\mathcal{R}_i$ l'ensemble des ***relations à i arguments***, $\mathcal{R}$ étant l'ensemble de toutes les relations :

$$\mathcal{R} = \bigcup_{i>0} \mathcal{R}_i$$

Une ***variable*** possède un nom qui est un atome. Appelons $\mathcal{V}$ l'ensemble des variables.

Un ***quantificateur*** est une fonction d'une variable. Appelons $\mathcal{Q}$ l'ensemble des quantificateurs :

$$\mathcal{Q} = \{\forall X / X \in \mathcal{V}\} \cup \{\exists X / X \in \mathcal{V}\}$$

Un ***symbole booléen*** est une fonction d'une ou deux variables. Appelons $\mathcal{S}$ l'ensemble des symboles booléens :

$$\mathcal{S} = \{\neg R \ / \ R \in \mathcal{R}\} \cup \{R_1 \text{ et } R_2 \ / \ R_1 \in \mathcal{R},\ R_2 \in \mathcal{R}\} \cup \{R_1 \text{ ou } R_2 \ / \ R_1 \in \mathcal{R},\ R_2 \in \mathcal{R}\}$$

A.II.1.b) Formules du langage des prédicats du premier ordre

Dès lors, les ***formules*** du langage des prédicats du premier ordre sont obtenues à partir de l'ensemble des variables propositionnelles $\mathcal{P}$:

$\mathcal{P} = C \cup S \cup Q \cup R$

Les formules sont construites à l'aide de la grammaire suivante :

F	→	*(F)* | *F ou F* | *F et F* | ¬ *(F)* | ∃*A F* | ∀*A F* | *L* | ∃*A L* | ∀*A L*	*/* Formules */*
L	→	*A(Arg)* | ¬ *A(Arg)*	*/* Littéraux */*
Arg	→	*A* | *A,Arg*	*/* Arguments */*
A	→	*toute composition de caractères*	*/* Atomes */*

Dès lors, on appelle ***prédicat*** une formule à laquelle une sémantique a été associée, c'est-à-dire que le prédicat peut être évalué en *vrai* ou *faux*.

A.II.1.c) Clauses de Horn

On appelle ***littéral*** tout terme du langage qui est composé à l'aide de la seconde règle de la grammaire précédente.

On appelle ***clause*** une formule sans quantificateur qui est une *disjonction* de littéraux, c'est-à-dire une formule composée exclusivement à l'aide de la fonction *ou* :

α_1 ou α_2 ou ... ou α_k ou ¬ α_{k+1} ou ... ou ¬ α_n

Une clause peut également être notée sous la forme suivante :

(α_1 ou α_2 ou ... ou α_k) ← (α_{k+1} et ... et α_n)

α_1, α_2,..., α_k sont dits littéraux positifs

α_{k+1}, ..., α_n sont dits littéraux négatifs

On appelle ***clause de Horn*** une clause admettant au plus un littéral positif :
$\alpha_0 \leftarrow (\alpha_1$ et ... et $\alpha_n)$

On appelle ***fait*** une clause qui affirme que, pour toutes les valeurs possibles des variables d'un littéral α, le prédicat α est vrai. Un fait α est noté :
$\alpha \leftarrow$

A.II.2. Prédicats fondamentaux du modèle de SECOOSSE

Nous allons donc définir le modèle de SECOOSSE à l'aide de clauses de Horn. Nous nous plaçons, pour ce faire, dans un univers de base dans lequel ne sont définis que deux types : les identificateurs et les nombres. Ainsi, le prédicat ***identificateur***(X) signifie que X est un identificateur, tandis que le prédicat ***nombre***(N) signifie que N est un nombre.
De plus, afin de ne pas préjuger de la richesse de l'environnement dans lequel est défini le modèle de SECOOSSE, nous proposons un troisième prédicat, ***prédéfini***(T), qui indique que T est un type prédéfini, donc existant indépendamment de SECOOSSE.

Sur la base de ces trois prédicats, nous allons définir une base de prédicats qui définira SECOOSSE. Cependant, parmi ces prédicats, nous allons voir apparaître des *faits*. Ces faits ne sont pas spécifiques de la définition de notre modèle. Au contraire, ils dépendent pleinement des concepteurs qui exploiteront le modèle proposé : en tout état de cause, une telle base de faits formalise un schéma conceptuel particulier.

A.III. Modèle fondamental

Le modèle utilisé dans SECOOSSE est le modèle orienté objets. Nous rappelons et formalisons sous forme de prédicats les notions que nous retenons de ce modèle.

Le modèle orienté objets est construit à partir du modèle centré objets, auquel sont ajoutés les classes, puis l'héritage. Le concept initial du modèle de SECOOSSE est la classe, présentée en A.III.1. Les notions de classe et d'objet étant interdépendantes, nous présentons également dans ce paragraphe le concept d'objet.

Les propriétés des classes, attributs et opérations, sont présentées en A.III.2.

A.III.1. Objets et classes

Nous appelons ***objet*** toute entité de l'univers réel du problème. Comme l'exprime le modèle orienté objets standard, il est possible de regrouper et de décrire un ensemble d'objets à l'aide d'une ***classe***, qui décrit alors, formellement, chacun de ces objets (appelés ***instances*** de la classe).

Les concepts d'objet et de classe sont deux composantes fondamentales du modèle orienté objet. C'est pourquoi, sous forme prédicative, il s'agit de faits : nous notons ***classe(C)*** le fait que C soit une classe, et ***objet(C,O)*** le fait que O soit instance de la classe C.

A.III.2. Propriétés

Comme l'exprime le modèle orienté objets standard, une classe est la description générique de ses instances. En effet, à chaque classe sont attribuées des ***propriétés***. Une propriété peut être statique (elle est alors appelée ***attribut***) ou correspondre à un comportement (il s'agit alors d'une ***opération***). Nous décrivons dans les paragraphes suivants les concepts d'attribut et d'opération.

A.III.2.a) Attributs

Un attribut associé une classe (nous dirons parfois "attribut d'une classe") consiste en une description générique de ***valeurs*** possédées par les instances de la classe. En effet, nous considérons un objet comme un ensemble de valeurs. Chacune des valeurs d'un objet est décrite par un attribut de sa classe ; réciproquement, pour chaque attribut d'une classe, chaque instance de cette classe possède une valeur, éventuellement indéfinie.

De même que pour les objets, chaque attribut possède une existence propre. Cependant, lors de la conception d'un logiciel, il est indispensable que des noms significatifs soient attribués. C'est pourquoi nous introduisons le concept de ***nom*** d'attribut.

Nous distinguons deux sortes d'attributs :

- d'une part, des attributs ***simples*** mémorisent des valeurs que nous dirons "de base" : textes, identificateurs, ... Nous ne désirons pas différencier ces différents types de base. En effet, selon l'expérience de chaque concepteur, certains types de base sont plus souvent utilisés que d'autres. De même, en fonction du problème à résoudre, certains types, pourtant évolués, peuvent être considérés comme "de base" (par exemple la pile, la table, ...)
- d'autre part, certains attributs décrivent des ***relations*** entre objets : les valeurs des objets, pour ces attributs, sont des ***références*** à d'autres objets. De tels attributs de relation possèdent alors nécessairement un ***domaine*** : chaque attribut de relation sait à quelle classe doivent appartenir les objets référencés par ses valeurs correspondantes.

Les attributs peuvent enfin être distingués en fonction de leurs ***cardinalités***, minimum et maximum : la valeur que possède un objet pour un attribut peut être unique (la cardinalité de l'attribut est 1) ou au contraire consister en un ensemble d'éléments (le nombre d'éléments étant limité par les cardinalités de l'attribut). La notion de cardinalité des attributs est très importante dans le contexte de la conception, puisqu'elle permet d'exprimer des contraintes, en particulier sur les relations entre objets, contraintes qui proviennent de l'univers réel. Nous considérons cependant, dans un but d'homogénéité, que toute valeur consiste en un ensemble d'éléments ; le nombre d'éléments dans une valeur doit se trouver dans les limites de la cardinalité de l'attribut. Dans le cas particulier des attributs de cardinalité 1, les valeurs correspondantes sont des singletons. Nous définissons donc le prédicat ***attribut_simple***(C,A), qui définit l'attribut A comme un attribut simple porté par la classe C ; en particulier cet attribut possède des cardinalités minimum et maximum et un domaine qui est un type prédéfini :

(1)
attribut_simple(C,A) ← ∃*I* ∃*N1* ∃*N2* ∃*D*
identificateur(I)
et nom_propriété(A,I)
et nombre(N1)
et nombre(N2)
et cardinalités(A,N1,N2)
et prédéfini(D)
et domaine(A,D).

Nous définissons de même le prédicat ***attribut_relation***(C,A), l'attribut A, possédé par la classe C, ayant alors des cardinalités minimum et maximum et un domaine qui est une classe C' ; cet attribut peut, ou non, posséder un attribut de relation *dual*, c'est-à-dire un attribut A', attaché à la classe C', tel que tout objet O' référencé dans l'attribut A de l'objet O référence lui-même O dans son attribut A' :

(2)
attribut_relation(C,A) ← ∃*I* ∃*N1* ∃*N2* ∃*C'*
identificateur(I)
et nom_propriété(A,I)
et nombre(N1)
et nombre(N2)
et cardinalités(A,N1,N2)
et classe(C')
et domaine(A,C').

attribut_relation(C,A) ← ∃*I* ∃*N1* ∃*N2* ∃*C'* ∃*A'*
identificateur(I)
et nom_propriété(A,I)
et nombre(N1)
et nombre(N2)
et cardinalités(A,N1,N2)
et classe(C')
et domaine(A,C')
et domaine(A',C)
et attribut_relation(C',A')
et dual(A,A')
et dual(A',A).

Nous définissons alors le prédicat ***attribut***(C,A), qui définit A comme un attribut attaché à la classe C sans prendre en compte le fait que A soit simple ou de relation, comme suit :

(3)
attribut(C,A) ← *attribut_simple(C,A).*

attribut(C,A) ← *attribut_relation(C,A).*

Nous voyons que la définition d'un attribut fait intervenir les faits suivants :

nom_propriété(Propriété,Identificateur)
cardinalités(Attribut,CardinalitéMini,CardinalitéMaxi)
domaine(AttributRelation,ClasseDomaine)
dual(AttributRelation,AttributRelation)

Notons que les mécanismes de valorisation d'un attribut, de lecture d'une valeur, de vérification de domaine ou de cardinalité ne relèvent pas du modèle syntaxique que nous présentons. Ils ne sont en effet guère importants lors de la conception d'une application !

A.III.2.b) Opérations

Une opération associée à une classe (nous écrirons parfois "opération d'une classe") est une représentation d'un comportement particulier dont sont capables les instances de cette classe. La nature de ce comportement dépend exclusivement de l'univers réel en cours de formalisation. De même, la représentation formelle de ce comportement dépasse le cadre de notre modèle et entre dans le domaine de l'analyse de programmation, que nous choisissons de déléguer à une méthode existante, choisie au gré du concepteur.
Dès lors, nous ne considérons plus les opérations que comme des entités *nommées*, attachées aux classes. Nous décrivons l'attachement d'une opération O à une classe C à l'aide du fait ***opération***(C,O), un fait ***nom_propriété***(O,N) définissant le nom de chaque opération.

A.III.2.c) Encapsulation

Le modèle orienté objets standard définit la notion ***d'interface*** d'un objet comme l'ensemble des propriétés de la classe de cet objet, auxquelles les autres objets peuvent accéder. Dès lors, et réciproquement, un objet ne peut accéder qu'aux propriétés d'interface d'un autre objet. On appelle ***encapsulation*** le fait que les propriétés d'un objet, qui n'appartiennent pas à son interface, ne peuvent être manipulées que par l'objet lui-même. Les concepts d'interface et d'encapsulation constituent dans le cadre de ce modèle deux notions fondamentales, puisqu'elles définissent la ***vision*** que possède un objet d'un autre objet : il s'agit là, très exactement, de l'aspect syntaxique d'un modèle. C'est pourquoi nous définissons notre modèle sur la base de deux prédicats, définis dans les paragraphes qui suivent :

- ***propriété(C,P)*** signifie que la propriété P est définie sur la classe C.
- ***voit(O',O'',P)*** signifie que l'objet O' peut accéder à la propriété P définie sur l'objet O".

A.III.2.c.i) Propriétés définies sur une classe

Nous avons vu que les propriétés définies sur une classe peuvent être des attributs ou des opérations. Nous définissons donc le prédicat ***propriété***(C,P) par les deux clauses :

(4)
$$\begin{array}{|lll} \textit{propriété(C,P)} & \leftarrow & \textit{opération(C,P).} \\ \textit{propriété(C,P)} & \leftarrow & \textit{attribut(C,P).} \end{array}$$

A.III.2.c.ii) Propriétés accessibles par un objet

Comme nous venons de le définir, un objet O' ne peut accéder à une propriété P sur un objet O" que si P appartient à l'interface de la classe de O". Or, le modèle orienté objets standard définit l'interface d'une classe comme l'ensemble des opérations de cette classe, de sorte que les accès aux valeurs sont toujours réalisés explicitement à travers une opération sur l'objet cible. Dès lors, nous pouvons exprimer la vision que possède O', de O", de la façon suivante :

- dans le cas où l'accès est réalisé par un objet sur lui-même, toutes les propriétés peuvent être vues
- dans les autres cas, l'accès ne peut être réalisé que sur les opérations

(5)
$$\begin{array}{|lll} \textit{voit(O,O,P)} & \leftarrow & \exists C \\ & & \textit{classe(C)} \\ & & \textit{et objet(C,O)} \\ & & \textit{et propriété(C,P).} \end{array}$$

(6)
$$\begin{array}{|lll} \textit{voit(O',O'',P)} & \leftarrow & \exists C'' \\ & & \textit{classe(C'')} \\ & & \textit{et objet(C'',O'')} \\ & & \textit{et opération(C'',P).} \end{array}$$

En fait, la première définition nous amène à une remarque : toutes les instances d'une classe connaissent le même ensemble de propriétés, attributs comme opérations ; dès lors, pourquoi une instance d'une classe ne pourrait-elle pas accéder à un attribut d'une autre instance de la même classe, ce que l'on exprimerait en remplaçant la clause (5) par :

(7) *voit(O',O",P)* ← ∃*C*
classe(C)
et objet(C,O')
et objet(C,O")
et propriété(C,P).

En fait ce relâchement de la contrainte d'encapsulation n'imposerait plus le passage par une opération de l'objet cible lors de l'accès à un attribut. Or de telles opérations d'accès possèdent certains avantages :

- il est possible de choisir entre une représentation dynamique (l'opération et pas d'attribut) et une représentation statique (opération + attribut) pour un concept de l'univers réel : donnons l'exemple de l'âge, qui peut être calculé à chaque fois qu'il en est besoin (une opération suffit) ou qui peut être mémorisé (attribut) et mis à jour périodiquement. Un changement de représentation n'influe en rien sur la façon dont les autres objets accèdent au résultat.
- il est possible de définir que, lors de chaque accès à la valeur, un traitement doit être effectué. Ce traitement est typiquement réalisé par l'opération d'accès ; si un autre objet pouvait accéder directement à l'attribut, ce traitement ne serait pas effectué.

Ces deux arguments montrent que les opérations d'accès aux attributs présentent des avantages. Aussi refusons-nous qu'un objet puisse accéder directement aux attributs d'un autre objet de la même classe, puisque les opérations d'accès seraient alors «court-circuitées».

A.III.2.c.iii) Propriétés privées et propriétés publiques

En définissant l'interface d'un objet, nous avons implicitement distingué les propriétés en propriétés publiques, appartenant à l'interface (les opérations) et en propriétés privées (les attributs). Cette dichotomie de l'ensemble des propriétés pourrait être affinée par l'étude :
- d'attributs publics
- d'opérations privées.

Nous venons de justifier que notre modèle n'inclut pas les attributs publics, en signalant que l'accès par une opération garantit à la fois la souplesse de conception et la cohérence des traitements.
Par contre, le concept ***d'opérations privées*** apporte certains avantages. En effet, nous reprenons à [CARRE 89] l'idée selon laquelle toutes les opérations

d'un objet ne sont pas réellement utiles dans son interface. Au contraire, certaines opérations sont absolument privées : citons particulièrement le cas de la décomposition, modulaire ou récursive, d'une opération en plusieurs opérations plus simples. Autant l'opération complète peut appartenir à l'interface de l'objet qui la porte, autant chaque «sous-opération» n'a de signification que dans un contexte particulier, mis en place par l'opération complète. Par conséquent, ces «sous-opérations» ne doivent en aucun cas être déclenchées indépendamment de l'opération complète : elles ne peuvent donc pas appartenir à l'interface de l'objet. Cette distinction entre opérations publiques et opérations privées possède de plus l'avantage de permettre des modifications aisées durant l'étape de conception : en effet, quelle que soit l'organisation des opérations privées, l'interface de l'objet demeure identique. Par conséquent, l'analyse de la classe correspondante n'entraîne pas la modification des spécifications des autres classes.

Dès lors, le fait ***opération***(C,O) est remplacé par deux faits, ***opération_publique***(C,O) et ***opération_privée***(C,O), qui définissent O comme une opération respectivement publique et privée attachée à la classe C. Nous pouvons alors définir ***opération***(C,O) comme un prédicat :

(8)

opération(C,O) ← ∃*I*
identificateur(I)
et nom_propriété(O,I)
et opération_publique(C,O).

opération(C,O) ← ∃*I*
identificateur(I)
et nom_propriété(O,I)
et opération_privée(C,O).

Cette modification de la définition du prédicat opération(C,O) ne modifie en rien la définition de l'ensemble des propriétés attachées à une classe : le prédicat ***propriété***(C,P) reste défini comme à l'équation 4.

Nous devons cependant redéfinir le prédicat ***voit***(O',O",P) de telle sorte que l'objet O' ne puisse accéder à la propriété P sur O" que si P est une opération publique ; nous rappelons cependant qu'un objet O peut accéder à toutes les propriétés définies sur sa classe :

(9) | *voit(O,O,P)* ← ∃*C*
classe(C)
et objet(C,O)
et propriété(C,P).

(10) | *voit(O',O",P)* ← ∃*C"*
classe(C")
et objet(C",O")
et opération_publique(C",P).

A.III.2.d) Homonymie

Nous avons attribué un nom à chaque propriété. En effet, durant la conception d'un logiciel, chaque attribut, chaque propriété possède une sémantique qui lui est propre dans l'application construite : cette sémantique est habituellement résumée dans le nom de la propriété. Il peut dès lors arriver que deux propriétés associées à une même classe possèdent le même nom. Etudions les conséquences des différents cas d'homonymie :

- deux attributs, P' et P", possèdent le même nom. Une instance de la classe qui possède ces deux attributs ne peut deviner lequel doit être considéré, l'action est donc aléatoire : ceci n'est pas envisageable dans le contexte de la conception de logiciels, donc ce cas de figure ne peut exister dans notre modèle.
- un attribut A et une opération O possèdent le même nom. Ce cas serait le moins critique, car les mécanismes d'accès aux attributs et de déclenchement des opérations pourraient reconnaître la propriété qu'il convient de traiter. Ce cas est cependant source de conflits de spécification et n'est pas non plus admis dans le modèle.
- deux opérations P' et P" possèdent le même nom. L'objet cible du déclenchement, ne sachant laquelle des opérations déclencher, aurait à nouveau un comportement aléatoire : ce cas est également exclu de notre modèle.

Posons le prédicat ***même_nom(P',P")*** :

(11) | *même_nom(P',P")* ← ∃*I*
identificateur(I)
et nom_propriété(P',I)
et nom_propriété(P",I).

Nous pouvons alors redéfinir le prédicat ***propriété(C,P)*** de telle sorte que les cas d'homonymie ne soient pas admis :

(12)
propriété(C,P) ← *attribut(C,P)*
et ¬(homonyme_existe(C,P)).

propriété(C,P) ← *opération(C,P)*
et ¬(homonyme_existe(C,P)).

Le prédicat *homonyme_existe* est alors décrit par :

(13)
homonyme_existe(C,P) ← ∃*P'*
propriété(C,P')
et même_nom(P,P').

A.III.3. Les faits

Rappelons ici les prédicats que nous avons définis comme étant des faits, créés par les utilisateurs du modèle et définissant les concepts gérées par SECOOSSE :

objet(Classe,Objet)
classe(Classe)
cardinalité(Attribut,Min,Max)
domaine(Attribut,Classe)
opération_publique(Classe,Opération)
opération_privée(Classe,Opération)
nom_propriété(Propriété,Nom)

A.III.4. Les prédicats

Les prédicats suivants constituent le coeur du modèle de SECOOSSE :
attribut(Classe,Attribut)
attribut_simple(Classe,Attribut)
attribut_relation(Classe,Attribut)
opération(Classe,Opération)
propriété(Classe,Propriété)
voit(Objet1,Objet2,Propriété)

Les prédicats suivants sont un peu plus utilitaires mais restent indispensables à la définition de SECOOSSE :
même_nom(Propriété1,Propriété2)
homonyme_existe(Classe,Propriété)

A.IV. Objets évolutifs

Les classes permettent de mettre en évidence une structuration statique des objets ; les propriétés attribuées aux objets sont alors valables à tout moment. Or le fonctionnement de tout système est basé sur le changement d'état des objets de ce système. Certaines méthodes, comme REMORA [ROLLAND 87], sont intégralement construites à partir de cette constatation ; c'est alors le changement des *valeurs* des objets qui est pris en compte.
Certains objets peuvent cependant changer d'état à un niveau plus élevé que leurs valeurs : leurs ensembles de propriétés eux-mêmes peuvent être modifiés. Considérons l'exemple du relais nommé "K37" ; lorsque "K37" est dans l'état Travail, les caractéristiques électriques du signal qui commande sa bobine peuvent être mémorisées : elles ont alors une signification. Par contre, lorsque K37 est au repos, aucun signal ne le traverse : dès lors, la caractérisation électrique est caduque. La formalisation d'un relais au travail contient les attributs «Tension du signal de commande» et «Intensité du signal de commande», propriétés que ne contient pas la description d'un relais au repos.
Cependant, il s'agit là du ***même*** relais qui peut à tout moment passer du repos au travail et réciproquement : la représentation de ce relais peut donc évoluer à chaque instant mais son identité, *a fortiori* son identifiant, restent constants. Nous appelons ***objets évolutifs*** ces objets dont la sémantique n'est pas fixée et constante.

Donnons un autre exemple d'objet évolutif : au démarrage d'une session de diagnostic, aucune information n'est connue quant à l'état de défaillance des différents modules qui composent l'équipement. Ensuite, une mesure met en évidence que certains modules génèrent le signal attendu : ces modules évoluent donc de l'état ***inconnu*** à l'état ***innocent***. D'autres mesures, ainsi que des règles expertes, incriminent des modules, dont l'état n'est plus ***inconnu***, mais ***en_panne***. Lorsque vient l'heure de la réparation, chaque module en_panne doit indiquer la marche à suivre pour aboutir à sa guérison : la description de ces modules inclut donc une opération nommée ***réparer***. Par contre, les modules innocents n'ont pas à connaître l'opération ***réparer*** ! Là encore, un module, tout en conservant son identité, peut être décrit différemment selon qu'il se trouve dans un état ou dans un autre.

Nous constatons que la description statique des objets à l'aide de classes, telle qu'elle est présentée dans le modèle orienté objets standard, ne suffit pas pour représenter ces objets évolutifs. Il est donc nécessaire d'enrichir le modèle de

SECOOSSE pour en permettre la description. Plusieurs propositions dans ce sens ont été faites : par exemple [CHEVAL 89] propose de gérer la description de l'évolution des objets à l'aide d'une base de règles et d'un moteur d'inférences ; nous rejetons d'emblée cette proposition qui, faisant intervenir un raisonnement indéterminé, ne peut convenir à un contexte de conception, dans la mesure où le concepteur doit être en mesure de prévoir et de contrôler la totalité des concepts qu'il introduit. Par opposition, [LEONARD 89] propose une abstraction (discutée également dans [LALANDA 89], nommée *abstraction d'alternative*, qui définit un modèle conceptuel pour l'évolution des objets. Cet auteur définit trois formes d'évolution possibles :

- les ***alternatives de concrétisation*** permettent essentiellement des choix lors de la réalisation du produit logiciel. En effet, supposons qu'une classe PILE soit définie : bien que les propriétés de cette classe soient fixées (opérations Empiler et Dépiler, par exemple), son implantation peut être réalisée de différentes façons : au moyen d'une table, d'une liste, d'un fichier, ...
- les ***alternatives de comportement*** permettent de spécifier différentes sémantiques possibles pour des objets qui ont un point commun. Cette notion correspond en fait au concept de spécialisation.
- les ***alternatives d'évolution*** permettent de spécifier des objets dont la sémantique change au cours du temps. Cette abstraction permet de définir des objets qui possèdent plusieurs définitions successives (non simultanées) tout en conservant leur identité.

Le concept d'alternative de concrétisation ne concerne pas la conception proprement dite ; la notion d'alternative de comportement correspond, nous l'avons dit, à la relation de spécialisation. Par contre, l'idée d'alternatives d'évolution répond parfaitement au problème des objets évolutifs que nous avons soulevé. C'est pourquoi nous décidons d'inclure dans le modèle de SECOOSSE une notion d'évolution qui corresponde à cette abstraction. Pour intégrer le concept d'évolution à notre modèle, nous allons modifier les définitions de classe et d'objet de telle sorte que chaque objet puisse être évolutif, et chaque classe puisse être instanciée par des objets évolutifs. Notons dès à présent que les objets du modèle standard sont des objets évolutifs... qui n'évoluent pas.

A.IV.1. Objets et classes

Nous émettons l'hypothèse que l'évolution d'un objet n'est pas continue, mais discrète ; les différentes étapes de son évolution correspondent à des structurations différentes des propriétés qui constituent la description de l'objet. Nous nommons alors ***alternative d'évolution*** l'une de ces étapes. Cependant, certaines propriétés sont valables durant toute la vie d'un objet, quelles que soient les alternatives d'évolution dans lesquelles il se trouve : ces propriétés constituent la partie statique de la description de l'objet. Si un objet ne possède pas d'alternative d'évolution, seule cette partie statique est utilisée pour décrire l'objet.

Pour conserver une représentation homogène, nous choisissons de représenter chaque ensemble de propriétés, qui définit une alternative d'évolution ou la partie statique de la description d'un ensemble d'objets, par une classe. Deux catégories de classes sont alors nécessaires :

- une ***classe non sémantique*** est une classe qui possède un ensemble de propriétés ne suffisant pas à décrire des objets. Chaque ensemble de propriétés, propre à chacune des alternatives d'évolution, est décrit à l'aide d'une classe non sémantique, que nous appelons ***classe alternative***.
- une ***classe sémantique***, par contre, possède un ensemble de propriétés suffisant pour décrire un objet indépendamment de son évolution.

Le concept de classe tel qu'il est défini dans le modèle orienté objets standard correspond alors simplement à la classe sémantique. Cependant, pour modéliser l'évolution des objets, nous formalisons également une relation particulière entre les classes alternatives et les classes sémantiques. Cette ***relation d'alternative*** est nécessaire pour formaliser, à l'aide de classes alternatives, les différentes alternatives d'évolution d'un objet qui est instance d'une classe sémantique. La relation d'alternative est définie comme suit :

> *«Une instance d'une classe sémantique est en même temps instance d'une et une seule classe alternative liée à cette classe sémantique. L'instance peut changer de classe alternative tout en restant la même instance de la classe sémantique».*

Dès lors, l'ensemble des propriétés décrivant un objet, instance d'une classe C, dans l'alternative A, correspond à la réunion de l'ensemble des propriétés de C et de A.

A.IV.2. Formalisation

Une classe peut être soit une classe sémantique, soit une classe alternative. Le prédicat ***Classe***(C) n'est donc plus un fait :

(14)
classe(C) ← *classe_sémantique(C).*
classe(C) ← *classe_alternative(C).*

La description de la dynamique individuelle des instances d'une classe est alors décrite par une séquence de faits :

- ***alt_initiale***(CS,CA) : lorsqu'une classe sémantique CS est instanciée, l'objet créé est dans l'alternative CA

- ***transition_possible***(CA1,CA2) : lorsqu'un objet est dans l'alternative CA1, alors il peut passer dans l'alternative CA2.

Nous définissons un prédicat "utilitaire", qui indique qu'une classe sémantique CS et une classe alternative CA peuvent être utilisées conjointement pour décrire des objets :

(15)
a_pour_alt(CS,CA) ← *alt_initiale(CS,CA).*
a_pour_alt(CS,CA) ← *transition_possible(CA,CA')* *et a_pour_alt(CS,CA').*

La relation qui lie un objet à une classe est toujours définie par le fait ***Objet***(Classe,Objet). Pour définir explicitement l'alternative d'évolution dans laquelle se trouve un objet, et suivant la définition de la relation d'alternative, nous choisissons de matérialiser par *deux* clauses ***Objet***(C,O) le fait qu'un objet est instance à la fois de sa classe sémantique et d'une classe alternative.

A.IV.3. Propriétés

Nous n'apportons aucune modification à la définition des propriétés, qui restent formalisées par les faits et prédicats suivants :

objet(Classe,Objet)
classe(Classe)

cardinalité(Attribut,Min,Max)
domaine(Attribut,Classe)
opération_publique(Classe,Opération)
opération_privée(Classe,Opération)
nom_propriété(Propriété,Nom)

attribut(Classe,Attribut)
opération(Classe,Opération)
propriété(Classe,Propriété)
même_nom(Propriété1,Propriété2)
voit(Objet1,Objet2,Propriété)

A.IV.4. Encapsulation

Puisque nous avons défini l'évolution d'un objet comme un *changement partiel de classe* (une partie de l'objet restant dans la même classe, une autre partie changeant de classe), lorsque cet objet est considéré à un instant donné, il est simultanément instance de deux classes : une classe sémantique, statique, et une classe alternative. Pour chacune de ces classes, les prédicats ***propriété***(C,P) et ***voit***(O',O",P) restent absolument valides, parce qu'il existe deux clauses ***objet***(O,C) pour un même objet O.

A.IV.5. Homonymie

Par contre, le modèle doit interdire que deux propriétés, définies l'une sur une classe sémantique, l'autre sur une classe alternative liée, possèdent le même nom. Si tel était le cas, le prédicat ***Voit***(O',O",P) indiquerait, de façon non déterminée, l'une ou l'autre des propriétés homonymes. C'est pourquoi nous posons une nouvelle définition du prédicat ***homonyme_existe*** :

(16)

homonyme_existe(C,P) ← ∃ *P'*
même_nom(P,P')
et propriété(C,P').

homonyme_existe(C,P) ← ∃ *P'* ∃ *CA*
même_nom(P,P')
et a_pour_alt(C,CA)
et propriété(CA,P').

A.V. Modèle de l'héritage

Dans les modèles orientés objets, les classes peuvent être organisées en une hiérarchie, appelée ***hiérarchie d'héritage***. Cette notion d'héritage est une réalisation de l'abstraction de généralisation/spécialisation.

A.V.1. Vocabulaire

La hiérarchie d'héritage permet à chaque classe de connaître explicitement sa classe mère, appelée ***super-classe***. Réciproquement, une classe fille est appelée ***sous-classe***. Par abus de langage, toutes les classes ancêtres sont aussi appelées super-classes, tandis que les classes descendantes sont nommées sous-classes.

La hiérarchie d'héritage peut attribuer à une classe plusieurs parents directs, chacun des parents directs transmettant ses propriétés (propres ou elles-mêmes héritées) à la classe concernée : on parle alors d'***héritage multiple***.

Nous proposons plusieurs formulations différentes de la propriété de l'héritage :
- l'héritage consiste à transmettre toutes les propriétés d'une classe à chacune de ses sous-classes.
- une instance d'une classe est également instance de toutes les super-classes de cette classe.
- l'ensemble des propriétés d'une classe est la réunion des ensembles de propriétés de toutes ses super-classes.
- l'héritage permet de manipuler uniformément les instances de plusieurs classes en les considérant comme instances d'une super-classe commune.

Nous nous proposons d'étudier dans les paragraphes A.V.2 et A.V.3 les différentes formes et utilisations possibles de l'héritage. Cette étude aboutira à la conclusion qu'une forme particulière d'héritage est nécessaire lors de la conception de logiciels complexes, faisant intervenir plusieurs experts.

A.V.2. Etude de l'héritage simple

Nous appelons héritage simple la forme d'héritage qui consiste à associer à une classe au maximum une super-classe. Par contre, une classe peut être associée à plusieurs sous-classes.

L'héritage simple peut être considéré selon plusieurs points de vue :

(1) il peut être utilisé dans un but de réutilisation (de partage) de la connaissance d'une classe, par plusieurs de ses sous-classes. Plusieurs contextes peuvent aboutir à cette vision de l'héritage :

(1a) un comportement général est défini sur une classe ; ses sous-classes possèdent toutes le même comportement. Par exemple, la classe des RELAIS, en général, possède les propriétés TRAVAILLER (établir le contact sur les circuits interrompus par un relais) et REPOSER (ouvrir les circuits interrompus). Ces propriétés sont générales à tous les relais, qu'il s'agisse de micro-relais, de relais 12V ou de relais 220V.

(1b) des classes indépendantes possèdent des comportements identiques : une seule version de ces comportements est alors spécifiée, sur une super-classe commune, dont c'est le seul rôle. Par exemple, un contact d'un relais est ouvert ou fermé ; de même un interrupteur manuel est ouvert ou fermé. Le contact et l'interrupteur ne peuvent être formalisés dans une même classe ; pourtant ils possèdent des propriétés identiques (lorsqu'ils sont fermés, le signal est transmis alors qu'il ne l'est pas lorsqu'ils sont ouverts). Il est alors possible de créer une classe particulière, portant ces propriétés communes, dont héritent les classes CONTACT et INTERRUPTEUR.

(2) l'héritage peut être utilisé pour définir des sous-classes dont chacune possède une définition différente d'une même propriété. Cette propriété n'est alors pas explicitement définie sur la classe. Par contre, cette classe peut spécifier que toutes ses sous-classes possèdent la propriété. Par exemple, l'action de TRAVAILLER consiste à fermer les contacts du relais. Mais, en fonction de la nature du relais, l'action ETABLIR_CONTACTS est différente (un relais 12V alimente une bobine dont le champ entraîne un bras mécanique qui ferme les contacts, tandis qu'un micro-relais alimente la base de transistors ...). Tout relais possède une propriété ETABLIR_CONTACTS mais cette propriété est définie sur chaque sous-classe de RELAIS.

(3) l'héritage peut également être considéré comme une solution pour spécifier des «connaissances par défaut», selon les aspects (1a) et (2) : certaine connaissance étant spécifiée sur une classe selon l'aspect (1a), les sous-classes possèdent toutes cette connaissance. Il peut cependant arriver que la spécification de cette même connaissance sur certaines sous-classes soit différente de la spécification générale. Dans ce cas, c'est naturellement la spécification la plus spécialisée de la propriété qui est considérée. On appelle ***surcharge*** cette forme de raffinement d'une spécification générale élaborée par défaut. Un exemple classique, en dehors des relais : "tous les

oiseaux volent" (la propriété VOLER est définie sur la classe OISEAU), mais "une autruche ne vole pas" (la classe AUTRUCHE, sous-classe de OISEAU, porte une définition particulière de VOLER).

En (1a), (2) et (3), nous pouvons observer que la relation entre une classe et une sous-classe est une relation sémantique entre la classe et la sous-classe : les instances de la classe sont définies plus précisément par un classement dans les sous-classes. Cette relation permet de définir des comportements communs (1a), des réactions différentes à un même stimulus (2) ou, très important lorsqu'on spécifie un univers réel, de décrire un comportement *a priori* général mais éventuellement précisé dans certaines sous-classes (3). Cette relation, que nous nommons ***spécialisation*** d'une classe (ou inversement ***généralisation*** d'un ensemble de classes), permet de spécifier incrémentalement les classes.
Notons qu'en cas d'homonymie de propriétés définies sur une classe et sur une sous-classe de cette classe, la définition (3) de l'héritage est obligatoirement appliquée : l'homonymie de propriétés dans un graphe de spécialisation amène toujours à un processus de surcharge, qui considère la définition la plus spécialisée qui s'applique à un objet.

Au contraire, la définition (1b) n'associe aucune sémantique à la relation entre classe et sous-classes. L'usage de l'héritage selon cette définition permet seulement de spécifier, une seule fois et sur une seule classe, une propriété qui s'applique à plusieurs classes. Nous nommons ***factorisation*** ce mécanisme de réduction de l'ensemble des définitions de propriétés des classes. Autant les sous-classes, par la factorisation, possèdent toujours une sémantique, autant la super-classe par factorisation, que nous nommerons ***classe de factorisation***, ne possède qu'un ensemble incomplet de propriétés : aucune sémantique n'est définie sur une classe de factorisation. La factorisation consiste essentiellement en un outil de représentation. Notons que, par définition, il ne peut exister d'homonymie, donc de surcharge, entre les propriétés d'une classe de factorisation et celles de l'une de ses sous-classes.

A.V.3. Etude de l'héritage multiple

L'héritage multiple permet d'associer plusieurs super-classes à une classe. Nous avons vu que, dans le cas de l'héritage simple, une super-classe peut être considérée comme une généralisation d'une classe, ou comme solution pour un regroupement de propriétés communes à plusieurs classes (factorisation). Nous allons montrer dans ce paragraphe que ces notions de spécialisation-

généralisation et de factorisation sont encore valides lorsque l'héritage multiple est utilisé. Nous mettrons cependant en évidences certaines lacunes dans la définition de l'héritage multiple.

Observons, de la même façon que nous l'avons fait en A.III.2, les différents points de vue sous lesquels l'héritage multiple peut être utilisé :

(1) Considérons un ensemble de classes E_c et deux sous-ensembles, S_1 et S_2, de E_c. Nommons $S_{1,2}$ l'ensemble $S_1 \cap S_2$ et considérons le cas $S_{1,2} \neq \varnothing$.
Emettons alors l'hypothèse que toutes les classes de S_1 possèdent la propriété P_1, toutes les classes de S_2 possédant la propriété P_2 : dès lors, les classes de $S_{1,2}$ possèdent à la fois les propriétés P_1 et P_2.
L'héritage simple utilisé pour la factorisation nous permet de construire une classe C_1, possédant la propriété P_1 ; les classes de S_1 sont alors factorisées par C_1. De même les classes de S_2 peuvent être factorisées par une classe C_2 possédant la propriété P_2.
Cependant, nous observons alors que les classes de $S_{1,2}$ sont *simultanément* factorisées par C_1 et par C_2 : elles possèdent plusieurs super-classes de factorisation.
L'héritage multiple permet donc de réaliser plusieurs factorisations indépendantes sur une même classe. Nous pouvons alors noter que la factorisation par une seule super-classe est un cas particulier de la factorisation par plusieurs super-classes. C'est pourquoi nous redéfinissons l'héritage, utilisé en tant qu'outil de ***factorisation multiple***, comme le regroupement de propriétés dans n (n>0) super-classes. De la même façon qu'en A.III.2, il ne peut y avoir surcharge des propriétés factorisées.

(2) Lorsqu'un univers réel est observé par plusieurs individus, il peut arriver que chacun d'entre eux, en fonction de son domaine d'expertise, définisse un même objet de façon différente : de façon évidente, le domaine d'expertise considéré influe sur la connaissance des objets. Notamment, les Systèmes Experts de Seconde Génération sont fondés sur une représentation objective des entités de la réalité. Cependant, cette représentation est ensuite manipulée par plusieurs expert : citons notamment l'expert en raisonnement, chargé de construire le moteur d'inférence ; l'expert en maintenance est chargé d'exprimer les heuristiques ; d'autres intervenants ont pour rôle de construire une interface, ... Considérons cette situation sur l'exemple de la classe des RELAIS :

- l'entité structurelle réelle est formalisée à l'aide de cette classe RELAIS, qui possède des propriétés propres à ces entités : TRAVAILLER, REPOSER, ...
- un expert en diagnostic considère un relais comme instance de la classe RELAIS_DIAGNOSTIC, qui possède les propriétés METTRE_EN_PANNE, REPARER, ...
- un expert en ergonomie considère un relais comme une instance de la classe RELAIS_GRAPHIQUE, qui possède à nouveau ses propres propriétés, COLORIER, DESSINER, ...

Dès lors, l'héritage multiple permet d'établir des relations entre les classes RELAIS et RELAIS_GRAPHIQUE ainsi qu'entre RELAIS et RELAIS_DIAGNOSTIC. Ces relations ont une signification particulière, différente de la spécialisation/généralisation, car on ne peut écrire que la classe RELAIS spécialise la classe RELAIS_GRAPHIQUE : cette relation apporte exclusivement la possibilité de considérer une instance de RELAIS comme étant également instance des super-classes de RELAIS. Cependant, le concept d'héritage multiple définit ces différentes relations exactement comme s'il s'agissait d'une ***généralisation multiple*** de la classe RELAIS.

Considéré comme une solution pour la factorisation multiple (1), l'héritage multiple apporte une grande souplesse conceptuelle en permettant d'associer, par le processus de factorisation, une même propriété à plusieurs classes.

Par contre, l'héritage multiple en tant que généralisation multiple (2) n'apporte qu'une solution partielle au problème des points de vue que peuvent posséder plusieurs concepteurs d'un même objet. Cette solution est insatisfaisante pour plusieurs raisons. Etudions en effet le cas où l'une des super-classes doit être spécialisée par le concepteur qui l'a créée (par exemple, la classe RELAIS_DIAGNOSTIC est spécialisée par deux sous-classes, nommées RELAIS_EN_PANNE et RELAIS_INNOCENT). Selon les définitions habituelles de l'héritage, la classe RELAIS "hérite de" la classe RELAIS_DIAGNOSTIC, mais en aucun cas des classes RELAIS_EN_PANNE ou RELAIS_INNOCENT : la conceptualisation de l'univers du diagnostic doit être réalisée sur une seule classe, ce qui peut être insuffisant !

Etudions de plus le cas où plusieurs concepteurs définissent des propriétés homonymes sur les super-classes de RELAIS qui leur sont dévolues. Deux cas seulement peuvent se présenter :

- une propriété de même nom est également définie sur RELAIS ; le principe de surcharge peut alors être appliqué et une instance de RELAIS accédera toujours à cette propriété
- les propriétés homonymes ne sont pas surchargées sur RELAIS : à laquelle de ces propriétés homonymes sur les super-classes une instance de RELAIS va-t-elle accéder ? De nombreux écrits, sur ce sujet, n'ont apporté que des réponses partielles, voire insuffisantes, à ce problème ***d'homonymie dans les graphes d'héritage multiple***. Lors de leur réalisation, certains langages orientés objets vont jusqu'à un choix de la propriété en fonction de la date de création de chaque super-classe ! De tels artifices ne sont naturellement pas envisageables dans le contexte d'une méthode de conception ...

A.V.4. Nos conclusions et nos choix sur l'héritage

En conclusion de cette brève étude sur l'héritage, nous retenons les concepts suivants :

- d'une part, une relation de ***factorisation***, simple ou multiple, permet de lier une classe à plusieurs super-classes. Cette relation permet à une classe de considérer comme siennes toutes les propriétés définies sur les super-classes de factorisation.

- d'autre part, une relation de ***spécialisation/généralisation*** entre une classe et une super-classe offre une démarche conceptuelle intuitive pour définir incrémentalement une classe à partir de spécifications plus larges.

Par contre, nous ne retenons pas le concept d'héritage multiple en tant que formalisation de la généralisation multiple d'une classe : ce concept n'apporte pas un confort conceptuel suffisant -en particulier en ce qui concerne la formalisation des points de vues différents sur une entité du monde réel- eu égard au problème d'homonymie des propriétés, qui limite considérablement la liberté de chaque concepteur en imposant un dialogue permanent entre les différents domaines de représentation.

A.V.5. Notre proposition pour représenter un objet selon plusieurs domaines de connaissance

Considérons à nouveau l'exemple présenté en A.V.3 (page 83) : les relais sont formalisés à partir de l'univers réel ; un expert en diagnostic les définit selon son

point de vue ; un expert en ergonomie des systèmes informatiques décrit à son tour les relais en fonction de ses besoins.
Par rapport à cet exemple de base, ajoutons les hypothèses qui invalident l'héritage multiple en tant que démarche conceptuelle de représentation. En particulier, chaque concepteur doit être libre de définir les spécialisations nécessaires à la formalisation de son point de vue sur les relais ; de plus, chaque concepteur doit être libre de créer et de nommer les propriétés comme il l'entend. Nous définissons donc plus précisément l'exemple de la façon suivante :

- une classe RELAIS issue de la formalisation objective, spécialisée par les classes RELAIS_12V et RELAIS_220V
- une classe RELAIS définie par l'expert en diagnostic, spécialisée par RELAIS_EN_PANNE et RELAIS_INNOCENT
- une classe RELAIS définie par l'expert en interfaces, spécialisée par RELAIS_ROUGE et RELAIS_VERT.

Nous devons dès lors proposer un concept qui autorise toute instance de la classe RELAIS à être également instance des classes "points de vue". Qui plus est, notre modèle doit permettre à une instance de RELAIS d'appartenir simultanément à plusieurs spécialisations de ces "points de vue". Enfin notre modèle doit autoriser qu'il existe des propriétés homonymes définies sur différents points de vue.

Une première analyse montre que ce cahier des charges ne peut être réalisé au moyen des concepts d'héritages retenus, c'est à dire la spécialisation et la factorisation. C'est pourquoi nous proposons ensuite la notion de ***spécialisation multiple*** qui, à l'inverse de ce que propose la définition de l'héritage multiple en tant que généralisation multiple, considère chaque point de vue comme une spécialisation de l'objet réel.

A.V.5.a) Application des concepts standards aux points de vue

La seule solution applicable à notre problème, n'utilisant que les concepts de factorisation et de spécialisation, consiste à créer autant de classes artificielles qu'il existe de combinaisons de spécialisations de chaque point de vue, la classe RELAIS étant elle-même considérée comme un aspect particulier :

RELAIS_12V_ET_EN_PANNE_ET_ROUGE
RELAIS_12V_ET_EN_PANNE_ET_VERT
RELAIS_12V_ET_INNOCENT_ET_ROUGE

RELAIS_12V_ET_INNOCENT_ET_VERT
RELAIS_220V_ET_EN_PANNE_ET_ROUGE
RELAIS_220V_ET_EN_PANNE_ET_VERT
RELAIS_220V_ET_INNOCENT_ET_ROUGE
RELAIS_220V_ET_INNOCENT_ET_VERT

La factorisation permet alors de regrouper les propriétés propres à chaque aspect : RELAIS_12v, RELAIS_EN_PANNE, etc. Pour représenter *n* aspects différents d'une classe, le $i^{ème}$ aspect consistant en S_i spécialisations, il faut créer un nombre très important de classes :

$\prod_{i=1}^{n} S_i$ classes de combinaison

$\sum_{i=1}^{n} S_i$ classes de factorisation

Cette solution est inacceptable, techniquement dans la mesure où le nombre de classes nécessaires croît prodigieusement lorsque le nombre d'aspects augmente et conceptuellement au vu de la complexité de la démarche, donc des risques d'erreurs ou d'oublis correspondants. Mentionnons également les difficultés de mise à jour de la spécification conceptuelle obtenue, par exemple pour supprimer un aspect ...

A.V.5.b) Définition du concept de spécialisation multiple

L'intégration de la notion de ***spécialisation multiple*** dans un modèle de données destiné à la conception de SEM2G apporte non seulement une minimisation du nombre de concepts à manipuler (par la diminution du nombre de classes), mais autorise également une forte indépendance entre les travaux des différents concepteurs. Entièrement fondée sur les notions de super-classe et de sous-classe, la spécialisation multiple ne modifie pas fondamentalement le concept d'héritage, mais l'enrichit pour apporter un confort accru lors de la phase de conception. En effet, la spécialisation multiple est destinée à autoriser plusieurs concepteurs à définir un même ensemble d'objets de façon différente. Le concept proposé apporte une solution, intégrée au modèle orienté objets, à un problème qui existe depuis très longtemps : par exemple, les auteurs de

[SPACCAPIETRA 90] proposent un ensemble de définitions qui permettent d'intégrer des vues différentes dans les bases de données. Cependant les concepts proposés sont fondés sur le modèle Entité-Relation et sont difficilement adaptables, en tant que tels, au modèle orienté objets. De plus, ces auteurs prennent en compte un problème particulier de l'intégration de vues : le fait qu'une entité peut être modélisée par des concepts différents (par exemple, un concepteur considère cette entité comme une classe alors qu'un autre en dispose en tant qu'attribut). Cette position du problème dépasse le cadre de notre étude. De la même façon, [ABITEBOUL 90] propose la notion d'objets imaginaires pour formaliser des valeurs, des attributs, des objets et des classes virtuels ; mais notre propos est exactement opposé, puisque nos objets, loin d'être virtuels ou imaginaires, doivent exister réellement dans chaque aspect !
C'est pourquoi nous choisissons de formaliser chaque domaine de connaissance, ou aspect, en relation avec une modélisation de l'univers réel. Le lien qui existe alors entre un aspect et un concept du réel est représenté à l'aide de notre concept de *spécialisation multiple*. La modélisation du réel doit être réalisée avant les spécifications des différents aspects : dans le contexte des SEM2G, cette modélisation peut être réalisée en collaboration avec le constructeur de l'installation à maintenir, avec l'aide des documentations de cet équipement, tout en prenant en compte les avis des techniciens de maintenance.
Nous commençons par montrer que cette relation entre classes "point de vue" et classes "objectives" est une relation de spécialisation particulière. Nous étudions ensuite comment sont partagées les propriétés entre un ensemble de classes liées par la spécialisation multiple, ce qui nous amènera à discuter les concepts de propriétés privées et publiques mises en évidence en A.III.2.c.iii, page 70. Nous résolvons ensuite l'un des points qui nous ont fait éliminer l'héritage multiple de notre modèle : l'homonymie des propriétés.

A.V.5.b.i) Nature de la relation entre classes "point de vue" et classes "objectives"

Chaque concepteur, nous l'avons vu, crée sa propre hiérarchie pour décrire son point de vue sur un objet. Naturellement, en construisant ainsi son schéma conceptuel, le concepteur peut utiliser des notions qui proviennent de la représentation "objective". A ces propriétés réelles viennent s'ajouter celles qui correspondent aux traitement dans le domaine de connaissance considéré. Cette démarche de construction correspond exactement à la démarche de spécialisation présentée en A.V.2 (page 79).
Cependant, dans le cas présent, il existe plusieurs domaines de connaissances qui spécialisent la spécification objective : c'est pourquoi nous avons choisi de

nommer le concept proposé ***spécialisation multiple***. Cette dénomination exprime également l'opposition entre le concept présenté et la généralisation multiple, que nous avons rejetée en A.V.4 (page 84).

Nous définissons la notion de spécialisation multiple en respectant, initialement, les principes fondamentaux de l'héritage :

> *«Toute instance d'une classe est simultanément instance de ses super-classes par la spécialisation multiple»*

Cette définition correspond très exactement à la définition de l'héritage donnée en A.V.1 (page 79). Naturellement, corollairement, toute instance de la classe aspect, est également instance de la classe objective (issue de la formalisation objective de la réalité).

Par contre nous enrichissons cette définition avec la réciproque, qui garantit que chaque objet créé dans la base d'objets sera toujours représenté selon tous les points de vue :

> *«Toute instance d'une classe est également instance de chacune de ses sous-classes par la spécialisation multiple»*

Dès lors, chaque concepteur, lorsqu'il étudie un objet, n'a à considérer que la relation de spécialisation pour connaître les propriétés que possède cet objet.

Ainsi, l'exemple qui a servi de support à la position de notre problème peut être résolu à l'aide de la spécialisation multiple de la façon suivante :

- la classe RELAIS modélise les entités réelles
- la classe RELAIS_DIAGNOSTIC est un point de vue de la classe RELAIS, à laquelle elle est liée par la spécialisation multiple ; la classe RELAIS_DIAGNOSTIC possède deux sous-classes par spécialisation, RELAIS_INNOCENT et RELAIS_EN_PANNE.
- la classe RELAIS_GRAPHIQUE est un point de vue de la classe RELAIS, à laquelle elle est liée par la spécialisation multiple ; la classe RELAIS_GRAPHIQUE possède deux sous-classes par spécialisation, RELAIS_ROUGE et RELAIS_VERT.

Dès lors, une instance qui représente un relais est simultanément instance de la classe objective et de chaque classe aspect ou d'une de ses sous-classes. C'est par cette «multi-instanciation» qu'il est possible de combiner les sous-classes de chaque point de vue. Ainsi, un objet peut être simultanément instance de la combinaison suivante de classes :

- RELAIS_INNOCENT, lorsqu'elle est considérée par l'expert en diagnostic
- RELAIS_VERT, lorsqu'elle est considérée par l'ergonome
- tout en restant un RELAIS à part entière, avec ses propriétés TRAVAILLER et REPOSER.

Notons que le concept de spécialisation multiple peut avant tout être utilisé en tant que spécialisation/généralisation : par exemple, une propriété commune à tous les aspects peut être généralisée sur la classe objective, même si elle est ensuite surchargée sur certains aspects. Une critique possible à cette démarche consiste à remarquer que la représentation objective de la réalité perd de son objectivité dès lors que des propriétés y sont ajoutées par pure commodité. Nous notons cependant que notre univers de conception se résume à la réalité et aux différents domaines de connaissance formalisés. Si une connaissance est valable dans tous les domaines considérés, alors il s'agit d'une connaissance objective, digne d'être représentée sur la classe objective (ce qui correspond à un univers clos, dans lequel tout ce qui n'est pas explicite n'existe pas).

Ajoutons que la hiérarchie de spécialisation multiple et de spécialisation simple peut être définie incrémentalement durant l'étape conceptuelle : il est tout à fait possible d'ajouter des aspects à une hiérarchie existante ou d'ajouter des sous-classes à une classe aspect, puisque chaque aspect est conçu indépendamment des autres. Par contre, par opposition aux travaux présentés dans [RIEU 91], un objet appartient toujours *simultanément* à tous les aspects : il n'est pas possible, durant le fonctionnement de la base d'objets, de définir incrémentalement de nouveaux aspects pour caractériser les objets.

A.V.5.b.ii) Partage des propriétés dans un ensemble de classes liées par la spécialisation multiple

Conséquemment à la définition précédente, et de la même façon que dans la définition de la spécialisation simple, un objet, instance d'une classe aspect, hérite des propriétés de ses super-classes par la spécialisation multiple. Ceci permet en particulier à chaque concepteur d'avoir accès à toutes les propriétés qui définissent les objets dans leur univers réel.

Cependant se pose le problème de la communication entre les aspects ainsi créés. En effet, la spécialisation multiple, comme la spécialisation simple, ne permet jusqu'ici que la propagation dans le sens *classe* → *sous-classe*. Or il est impossible de construire une application faisant intervenir plus d'un expert, sans que les entités créées par chaque concepteur communiquent entre elles : à quoi servirait une interface graphique si le diagnostic s'effectuait sans faire appel à ses fonctions ?

Dans l'absolu, un objet qui est instance d'une classe aspect est également toujours, d'après les définitions précédentes, instance de toutes les autres classes aspects : il possède donc chacune des propriétés définies dans chacun des aspects (aux propriétés homonymes près, sur lesquelles nous nous étendons dans le paragraphe A.V.5.b.iv). Cependant, un concepteur ne peut savoir quelles sont les propriétés qu'a définies un autre concepteur, sauf s'il peut connaître ces propriétés par la relation de spécialisation entre sa classe aspect et la classe objective.

Dès lors, la communication entre classes aspects doit être réalisée par l'intermédiaire de la classe objective : chaque classe aspect ***publie***, sur la classe objective, les propriétés auxquelles peuvent accéder toutes les autres classes aspects. Cette affirmation est en contradiction avec les habitudes quant à la notion d'héritage : il est usuel en effet de ne considérer la propagation d'une propriété dans une hiérarchie d'héritage que des ancêtres vers les descendants. Nous justifions cependant cette notion de propagation dans le sens *sous-classe* → *classe* en étendant la notion ***d'héritage*** à celle de ***communication*** entre différents aspects d'un même objet. D'autre part, le fait que ce soit la classe objective qui connaisse les propriétés de ses sous-classes autorise un contrôle de la communication entre les aspects. Notons que la contrepartie de cet avantage est que la communication se fait nécessairement d'un aspect vers tous les aspects et qu'il n'est pas possible de publier une propriété destinée exclusivement à un sous-ensemble des aspects.

Prenant désormais en compte cette notion de communication, nous pouvons énumérer les propriétés que connaît un objet lorsqu'il est considéré dans un aspect particulier ; il s'agit :

- des propriétés attachées à la classe aspect correspondante
- des propriétés héritées par spécialisation

- des propriétés héritées par multi-spécialisation, c'est à dire :
  . les propriétés de la classe objective (attachées à cette classe ou à nouveau héritées)
  . les propriétés publiées par les autres aspects sur la classe objective.

Cette notion de propriétés partagées entre les aspects nous amène à étudier de nouveau la notion de propriétés privées et publiques.

A.V.5.b.iii) Retour sur les propriétés publiques et privées

Nous avons défini les propriétés comme étant accessibles (utilisables) seulement par les instances de la classe qui les porte, ou au contraire accessibles par les instances d'autres classes : nous avons rangé les attributs et les opérations privées parmi les propriétés privées, les opérations publiques constituant l'ensemble des propriétés d'interface d'une classe.
Nous avons également observé dans le paragraphe précédent que certaines propriétés peuvent être publiées par une classe aspect. Cependant, nous pouvons immédiatement affirmer que toutes les propriétés d'une classe aspect ne sont pas indispensables à la communication avec une autre classe aspect :

- par définition, une propriété privée ne peut être manipulée que par un objet de la classe à laquelle cette propriété est attachée. Un objet, vu par un concepteur, peut-il accéder à une propriété privée définie par un autre concepteur ? Nous répondons négativement, en rappelant qu'une propriété privée n'a généralement aucun sens intrinsèque, mais qu'elle fait partie d'un ensemble cohérent, géré par une opération publique.
- au contraire, une propriété publique peut, par définition, être manipulée par n'importe quel objet : *a fortiori* un objet peut accéder à une propriété publique même si elle est définie dans un autre aspect. Par conséquent toutes les opérations publiques attachées à une classe sont publiées sur les super-classes par spécialisation multiple.

Nous constatons que la distinction entre les propriétés publiques et les propriétés privées offre peu de choix dans la définition de la communication entre aspects : aucune propriété privée n'est publiée, toute opération publique l'est.
Aussi introduisons-nous une notion intermédiaire que nous nommons ***propriété publiée***. Une propriété publiée est une propriété, attribut ou opération, qui est définie sur une classe C, qui est normalement héritée par les sous-classes de C, mais *qui est également connue des super-classes par spécialisation multiple comme une propriété privée*. Cette propriété publiée est donc accessible par les

autres aspects d'un même objet (les aspects étant des sous-classes de la classe objective), mais n'est pas visible par des objets extérieurs à la hiérarchie de définition. Il s'agit en quelque sorte de propriétés «semi-privées».

Ainsi, considérons à nouveau la hiérarchie qui définit les relais et en particulier la classe RELAIS_GRAPHIQUE. Cette classe possède une opération, nommée DESSINER, chargée de représenter un relais à l'écran. Il s'agit d'une opération publique, dans la mesure où l'ordre de dessin provient d'un gestionnaire d'interface extérieur à la définition du relais. Par contre, la séquence de dessin d'un relais consiste à charger une image particulière qui sera placée sur l'écran : l'opération publique DESSINER fait appel, entre autres, à une opération privée, CHARGER_IMAGE, qui n'a de sens que dans le contexte de la représentation d'un relais à l'écran. Aucun autre objet qu'un relais graphique ne peut donc faire appel à l'opération CHARGER_IMAGE.
Par contre, considérons une opération nommée ILLUSTRER_REPARATION, définie sur RELAIS_GRAPHIQUE. Cette opération affiche une séquence de photographies qui illustrent la réparation du relais et peut être publique. Cependant, l'explication de la réparation n'a de sens que si le relais traité est en panne. Dès lors, l'opération ILLUSTRER_REPARATION ne doit pas être déclenchée par n'importe quel objet à n'importe quel moment. Au contraire, seule une instance de RELAIS_EN_PANNE, dans l'aspect RELAIS_DIAGNOSTIC, doit pouvoir déclencher cette opération. Dès lors, la propriété ILLUSTRER_REPARATION doit être publiée sur RELAIS, afin qu'elle soit héritée par la classe RELAIS_EN_PANNE, mais elle ne doit pas être publique, afin d'être cachée à l'extérieur de la définition des relais : c'est pourquoi ILLUSTRER_REPARATION est une propriété publiée, visible exclusivement par les classes qui définissent les relais.

A.V.5.b.iv) Homonymie dans une représentation multiple

Dans le modèle à base de classes, nous avons vu que deux propriétés ne pouvaient être homonymes si elles étaient définies sur la même classe (A.III.2.d, page 72).
Nous avons également défini l'homonymie de propriétés dans une hiérarchie de spécialisation comme correspondant à une surcharge, c'est à dire que la définition qui est utilisée est la plus proche de la classe de l'objet cible dans le graphe d'héritage (A.V.2, page 79).

Cependant, lorsque le concept de spécialisation multiple est inclus dans le modèle, de nouveaux cas d'homonymie de propriétés peuvent apparaître. Ces cas d'homonymie ne concernent que les opérations publiées par des classes aspects sur une classe objective. En effet, tout autre cas d'homonymie est résolu par le principe de surcharge, car il est toujours possible de se placer dans le contexte d'une hiérarchie de spécialisation, comme nous l'avons indiqué en A.V.5.b.i, page 87.

Dès lors, le problème à résoudre est celui de *n* aspects, publiant sur une même classe *n* propriétés homonymes. Deux problèmes sont alors posés :

- quelle est la priorité d'une propriété publiée sur une propriété héritée par spécialisation ?
- comment choisir, parmi *n* propriétés publiées, celle qui sera utilisée ?

Ce problème rejoint en fait celui de la résolution de l'homonymie dans une hiérarchie de généralisation multiple, avec cependant une différence majeure : ne sont publiées par un concepteur que des propriétés qui permettent l'interfaçage de ses classes avec celles des autres concepteurs. Or, en nous replaçant dans un contexte de méthode de conception, nous observons que les différents concepteurs doivent nécessairement se rencontrer pour se communiquer leurs interfaces respectives. Aussi résolvons-nous le problème de l'homonymie d'une façon inélégante pour ce qui est du modèle, mais adaptée et logique en ce qui concerne la démarche méthodologique : en interdisant à deux concepteurs de *publier* des propriétés de même nom. Naturellement, cette restriction n'interdit aucunement que des propriétés homonymes soient définies dans chaque aspect.

Sur la base de cette contrainte, nous définissons qu'une propriété publiée est considérée en priorité par rapport à une propriété héritée. En effet, un objet est simultanément décrit par toutes ses classes aspects, que l'on peut considérer globalement comme une grande classe : nous appliquons alors le principe de surcharge à cette classe globale.

Dès lors, une remarque importante concerne la suppression de la généralisation multiple dans le modèle de SECOOSSE. En effet, puisque les conflits d'homonymie ont été résolus par une règle méthodologique et non dans le modèle, cette solution aurait pu être appliquée à l'utilisation de l'héritage multiple. Rappelons cependant l'un des inconvénients de l'héritage multiple, mentionné en page 83 : considérons la classe RELAIS_DIAGNOSTIC, spécialisée par

RELAIS_INNOCENT et RELAIS_EN_PANNE, et la classe RELAIS_GRAPHIQUE, spécialisée par RELAIS_VERT et RELAIS_ROUGE. L'utilisation de l'héritage multiple pour définir la classe RELAIS amène à lui attribuer deux super-classes, RELAIS_DIAGNOSTIC et RELAIS_GRAPHIQUE ; dès lors, la classe RELAIS *n'hérite pas* des classes RELAIS_INNOCENT, RELAIS_EN_PANNE, RELAIS_ROUGE ou RELAIS_VERT : les propriétés définies sur ces classes ne peuvent être exploitées ! Si, pour éviter cet écueil, la classe RELAIS hérite directement de ces quatre classes, alors les concepteurs n'ont plus la possibilité de définir, indépendamment les uns des autres, des propriétés homonymes et, de plus, un relais sera *toujours à la fois* innocent et en panne, rouge et vert ...

A.V.6. Définition formelle de l'héritage

Observons désormais de quelle façon l'héritage influe sur le modèle des systèmes orientés objets que nous avons présenté :

- Nous avons déjà différencié, dans le paragraphe A.IV, les classes sémantiques des classes alternatives, dont l'ensemble de propriétés ne modélise pas un ensemble d'entités réelles ; l'héritage introduit le concept de classes de factorisation, qui sont également des classes non sémantiques.
- l'introduction de propriétés publiées nous amène à redéfinir l'ensemble des faits et de prédicats que nous avions proposé en A.III
- les cas d'homonymie de propriétés héritées selon plusieurs relations (factorisation, spécialisation simple ou multiple) doivent être résolus par la définition de priorités entre ces héritages
- l'héritage de propriétés modifie l'ensemble des propriétés qui peuvent être considérées sur une classe
- la spécialisation et la spécialisation multiple modifient l'ensemble des propriétés que voit un objet sur lui-même

  les trois formes d'héritage que nous avons retenues entraînent également une modification de l'ensemble des propriétés publiques définies sur une classe.

A.V.6.a) Formalisation de la hiérarchie

Une hiérarchie d'héritage est formalisée par les faits suivants :

factorise(C,C') qui signifie que la classe de factorisation C factorise les propriétés d'un ensemble de classes, parmi lesquelles C'.

multi_spécialise(C,C') qui signifie que C' est une classe objective, dont un aspect est représenté par la classe C.

spécialise(C,C') qui signifie que la classe C spécialise la classe C'.

A.V.6.b) Objets et classes

Nous avons déjà mis en évidence, en formalisant les objets évolutifs, qu'il existe deux catégories de classes : une *classe sémantique* modélise un concept de l'univers du discours, tandis qu'une classe non sémantique, en particulier une *classe alternative*, n'est utilisée que pour décrire un ensemble de propriétés. Certaines classes, parmi les classes sémantiques, sont utilisées pour représenter exactement, sans biais, les entités réelles et nous les avons nommées *classes objectives* dans les paragraphes qui précèdent. Précisons ici qu'une classe objective n'est rien d'autre qu'une classe sémantique qui possède des classes aspects.
Par ailleurs, le concept de factorisation tel que nous l'admettons dans notre modèle nous amène à considérer une nouvelle sorte de classes non sémantiques : il s'agit des classes qui sont utilisées lors du mécanisme de factorisation. Par construction, ces ***classes de factorisation*** ne possèdent pas un ensemble sémantique de propriétés ; par ailleurs les classes de factorisation ne correspondent pas aux classes alternatives, en particulier parce que la relation de factorisation est indépendante du temps, contrairement à la relation d'alternative. C'est pourquoi nous sommes amenés à ajouter ce concept à notre modèle, à l'aide d'un nouveau fait, ***classe_factorisation***(C), qui indique que C est une classe de factorisation. Nous redéfinissons alors le prédicat ***classe***(C) à l'aide des trois catégories de classes possibles :

(17)
$$\begin{cases} classe(C) \leftarrow classe\_factorisation(C). \\ classe(C) \leftarrow classe\_alternative(C). \\ classe(C) \leftarrow classe\_sémantique(C). \end{cases}$$

Nous avons défini le concept de spécialisation multiple de telle sorte que chaque concepteur ait la possibilité de formaliser un univers réel en fonction de son domaine d'expertise. Ce concept permet notamment à chacun des experts d'exploiter la totalité des concepts présentés jusque-là : classes, propriétés, spécialisation, factorisation, ...

Seul le concept d'objet, en tant qu'instance d'une classe, n'a pas été défini dans le contexte de la spécialisation multiple. Or, il est souhaitable -sinon indispensable- que chaque concepteur ait la possibilité de déclarer une instance aussi souplement qu'il utilise la spécialisation, c'est à dire en particulier sans tenir compte des travaux des autres concepteurs.
De plus, chaque concepteur a la possibilité de définir une hiérarchie de spécialisation simple en dessous de la classe aspect qu'il formalise. Dès lors, un objet de la classe objective ne peut pas être systématiquement considéré comme une instance directe des classes aspects ; au contraire, dans chaque domaine d'expertise, cet objet peut être instance de n'importe quelle sous-classe de la classe aspect. Notons qu'il n'est pas question, ici, de *cohérence sémantique* de cette appartenance d'une instance à plusieurs classes : ce problème ne peut être résolu que par les concepteurs, en fonction de leurs domaines d'expertise.

Il n'est pas possible, par conséquent, de décrire une instance d'une classe objective par un fait unique :

objet(la_classe_objective,l_objet).

En effet, ce fait ne permet pas de connaître exactement l'ensemble des classes dont l'objet est instance dans chaque domaine de connaissance.

C'est pourquoi, afin d'autoriser chaque concepteur à manipuler de façon identique chaque objet, et dans le but de décrire précisément les relations d'instanciation entre objets et classes, nous définissons qu'il existe une relation d'instanciation entre un objet et une classe dans chacun des domaines de connaissance considérés.

Considérons cette relation de multi_instanciation sur l'exemple des relais. La classe RELAIS est une classe objective qui formalise le comportement réel de ces entités. Deux classes aspects représentent les relais dans un contexte de diagnostic (RELAIS_DIAGNOSTIC) et d'ergonomie (RELAIS_GRAPHIQUE).

Le concepteur du diagnostic définit qu'un relais est soit innocent (classe RELAIS_INNOCENT) soit en panne (classe RELAIS_EN_PANNE) : ces deux classes spécialisent la classe RELAIS_DIAGNOSTIC.

De même, le concepteur de l'interface définit qu'un relais est soit rouge (classe RELAIS_ROUGE) soit vert (classe RELAIS_VERT) : ces deux classes spécialisent RELAIS_GRAPHIQUE.

Considérons alors le relais R1, qui est innocent et vert : vu par le diagnostic, R1 est instance de la classe RELAIS_INNOCENT ; vu par l'interface, cet objet est instance de RELAIS_VERT. Dans tous les cas, considéré objectivement, R1 est instance de RELAIS. Dès lors, l'existence de R1 est formalisée par l'ensemble de faits suivant :

*objet(*RELAIS,*R1).*
*objet(*RELAIS_INNOCENT,*R1).*
*objet(*RELAIS_VERT,*R1).*

De la même façon, un relais R2 innocent et rouge sera formalisé par les faits :

*objet(*RELAIS,*R2).*
*objet(*RELAIS_INNOCENT,*R2).*
*objet(*RELAIS_ROUGE,*R2).*

Chaque concepteur a alors à sa disposition un fait ***objet(C,O)*** qui concerne son domaine de connaissance.

A.V.6.c) Formalisation des propriétés

Nous avons défini une nouvelle catégorie de propriétés, les ***propriétés publiées***. La caractéristique d'une propriété publiée est d'être connue par une classe objective, donc d'être vue par les autres aspects d'un même objet, mais de ne pouvoir être manipulée par d'autres objets. Les propriétés publiées correspondent donc à des propriétés privées, dont la portée est élargie de la classe à l'ensemble de l'agrégat qui définit les différents aspects d'un objet.

Nous conservons, naturellement, le fait ***objet***(Classe,Objet) et le prédicat ***classe***(Classe), qui constituent le coeur de notre modèle. De même, la description des propriétés est inchangée, à ceci près que, pour une propriété P et pour une classe objective C, peut être défini un fait ***propriété_publiée***(C,P).

A.V.6.d) Homonymie (surcharge) et encapsulation

L'homonymie de propriétés héritées par des relations d'héritage différentes, impose tout d'abord l'étude du phénomène de surcharge, avant l'étude de l'encapsulation.

Notre hypothèse de base est que, en cas d'homonymie de propriétés attachées à une classe, que ce soit un attachement direct ou un attachement par héritage, une seule de ces propriétés doit être considérée. Nous justifions cette hypothèse par les points suivants :

- il eût été possible de considérer la totalité des propriétés homonymes ; cependant ces propriétés peuvent être, indifféremment, des opérations ou des attributs : le sens attribué à l'accès à une propriété est différent s'il s'agit de la lecture d'une valeur ou de la réalisation d'un comportement.
- ne considérer qu'un sous-ensemble de l'ensemble des propriétés homonymes aurait amené à un choix arbitraire.
- considérer les propriétés homonymes en fonction de leur nature (attribut ou opération, voire propriété privée ou publique) ne simplifie en rien le problème, seul le nombre de propriétés homonymes étant alors éventuellement restreint.

Il nous faut résoudre les problèmes suivants :

- une classe peut posséder une propriété de nom N et hériter d'une autre propriété également nommée N, voire de plusieurs autres propriétés de même nom.
- des propriétés homonymes peuvent être définies sur plusieurs niveaux d'une hiérarchie d'héritage.

Dans un contexte où l'héritage est indifférencié, il est couramment admis, dans de tels cas d'homonymie de propriétés, que la propriété la plus «proche» de la classe de l'objet cible et possédant le nom donné, est celle qui est déclenchée. Notons que c'est justement le flou sur la définition de «la plus proche» qui interdit une résolution formelle des problèmes d'homonymie dans le cas de l'héritage multiple.

Lorsque nous avons défini l'héritage, nous n'avons considéré que trois concepts : la factorisation, la spécialisation et la spécialisation multiple. Au contraire du

concept général d'héritage multiple, qui permet seulement à une classe d'hériter de plusieurs super-classes, nous pouvons établir des contraintes sur nos différentes formes d'héritage :

- rappelons tout d'abord qu'une classe ne peut porter de propriétés homonymes qui lui soient directement attachées.

- dans le cas de la factorisation :
  . une classe C ne peut hériter de propriétés homonymes par factorisation, puisque lorsque la relation de factorisation est construite, chaque propriété de C est soit laissée sur C, soit portée vers une super-classe par factorisation.
  . une classe peut posséder plusieurs super-classes par factorisation : il s'agit alors d'une *factorisation multiple* plutôt que d'une généralisation multiple. La différence tient dans le fait qu'une propriété de la classe C ne peut être définie qu'une fois, soit sur C elle-même, soit, par construction, sur une et une seule des classes de factorisation. Dès lors il ne peut y avoir d'homonymie des propriétés factorisées. De plus, il existe une différence fondamentale entre la factorisation multiple et la généralisation multiple : nous avons défini en A.V.4 (page 84) que la relation de factorisation permet de considérer les propriétés factorisées *comme si elles étaient définies sur la classe elle-même* ; il n'existe pas de notion explicite d'héritage de propriété dans le phénomène de factorisation, alors que les propriétés sont héritées dans un graphe de généralisation multiple.

- dans le cas de la spécialisation : par définition, une classe possède au maximum une super-classe par spécialisation. Si les problèmes d'homonymie sont résolus pour cette super-classe, alors ils le sont également pour la classe considérée.

- dans le cas de la spécialisation multiple :
  . une classe aspect ne peut posséder qu'une super-classe objective par la spécialisation multiple.
  . nous l'avons dit, la spécialisation multiple est considérée par une classe aspect comme une relation de spécialisation simple, telle qu'elle est définie en A.V.2 (page 79).

. une classe objective peut «hériter» de propriétés publiées par plusieurs classes aspects, mais nous avons interdit que ces propriétés publiées soient homonymes.

Nous pouvons donc constater qu'il n'est pas possible qu'une classe hérite de plusieurs propriétés homonymes par la même relation d'héritage. Nous choisissons donc d'établir un ordre sur les relations d'héritage, cet ordre étant celui qui est utilisé pour sélectionner une propriété parmi un ensemble de propriétés homonymes. L'ordre que nous proposons ici est une définition plus précise de la surcharge telle qu'elle a été présentée plus haut. En particulier, plus la définition d'une propriété est «proche», au sens de la *distance dans un graphe* d'héritage, de la classe d'un objet, plus cette propriété a de chances d'être sélectionnée. Cependant, cette notion de distance dans le graphe doit être pondérée par la sémantique associée aux arcs de ce graphe, c'est-à-dire les concepts de factorisation, de spécialisation et de spécialisation multiple. En effet :

- la factorisation ne constitue qu'un moyen artificiel de reporter la définition d'une propriété vers une classe différente de sa classe d'origine. Conceptuellement, la propriété reste attachée directement à la classe qui a été factorisée.

- la spécialisation multiple n'est qu'un outil qui permet la représentation d'une classe selon plusieurs points de vue ; il est évident que les propriétés qui proviennent des différents domaines de connaissance (c'est à dire les propriétés publiées sur la classe objective) doivent être considérées comme étant définies sur la classe objective elle-même.
- enfin, la spécialisation est un concept qui permet -essentiellement- la réutilisation de la connaissance exprimée sur les super-classes.

Nous n'avons guère mentionné, jusque là, les propriétés qui sont définies sur les classes alternatives d'une classe donnée. Mais nous avons montré que ces propriétés ne sont différentes des propriétés directement attachées à la classe que par le caractère temporaire de leur utilisation. Nous pouvons donc inclure les propriétés définies sur une classe alternative dans notre relation d'ordre, que nous définissons comme suit :

(18)

plus_proche(C,C,C") ←

plus_proche(C,C',C") ← *a_pour_alt(C,C')*
et ¬(a_pour_alt(C,C")).

plus_proche(C,C',C") ← *factorise(C',C)*
et ¬(a_pour_alt,C,C"))
et ¬(factorise(C",C)).

plus_proche(C,C',C") ← *multi_spécialise(C,C')*
et ¬(a_pour_alt,C,C"))
et ¬(factorise(C",C))
et ¬(multi_spécialise(C,C")).

plus_proche(C,C',C") ← *spécialise(C,C')*
et ¬(a_pour_alt,C,C"))
et ¬(factorise(C",C))
et ¬(multi_spécialise(C,C")).

Nous devons désormais redéfinir le prédicat ***propriété(C,P)*** de telle sorte que les propriétés héritées par une classe soient considérées comme attachées à cette classe, tout en tenant compte de l'ordre que nous avons établi sur la relation d'héritage ; cet ordre est en effet utilisé dans le cas de propriétés héritées homonymes :

(19)

propriété(C,P) ← *attribut(C,P)*
et ¬(homonyme_existe(C,P)).

propriété(C,P) ← *opération(C,P)*
et ¬(homonyme_existe(C,P)).

propriété(C,P) ← *propriété_publiée(C,P)*
et ¬(homonyme_existe(C,P)).

propriété(C,P) ← *propriété_héritée(C,P)*
et ¬(propriété_surchargée(C,P)).

Le prédicat ***propriété_héritée***(C,P) signifie que P est une propriété définie sur une super-classe de C :

(20)

propriété_héritée(C,P) ← ∃C'
classe_factorisation(C')
et factorise(C',C)
et propriété(C',P).

propriété_héritée(C,P) ← ∃C'
classe_sémantique(C')
et multi_spécialise(C,C')
et propriété(P,C').

propriété_héritée(C,P) ← ∃C'
classe_sémantique(C')
et spécialise(C,C')
et propriété(C',P).

Par ailleurs le prédicat ***propriété_surchargée***(C,P') indique que la propriété P', définie sur la classe C' est surchargée par une propriété C", définie sur une classe P" plus proche de C que ne l'est C' :

(21)

propriété_surchargée(C,P') ← ∃C' ∃C" ∃P"
classe(C')
et propriété(C',P')
et classe(C")
et propriété(C",P")
et plus_proche(C,C',C")
et même_nom(P',P").

A.V.6.f) Vision entre objets

Dès lors que le prédicat ***propriété*** est redéfini pour prendre en compte les nouvelles spécifications, la définition du prédicat ***voit*** (équation 6) reste valide. Cette définition indique, rappelons-le, que :

- un objet voit toutes les propriétés qui sont définies sur sa classe ; désormais ces propriétés sont soit directement attachées à cette classe, soit héritées.
- un objet ne voit que les opérations publiques d'un autre objet.

Il est inutile d'expliciter, dans ce prédicat, le cas des propriétés publiées, qui font l'objet des mêmes règles que les autres propriétés bien que leur définition consiste en une délégation vers la classe aspect qui publie ; les sous-classes de la classe objective héritent des propriétés publiées comme des autres propriétés.

A.VI. Modèle de la dynamique

Quels que soient leur domaine d'application ou leur modèle conceptuel, les méthodes de conception ont toujours un objectif double : d'une part, la définition des données permet de formaliser la partie statique d'un univers réel ; d'autre part, des caractéristiques dynamiques de l'univers réel doivent être représentées. A quoi correspondent ces «caractéristiques dynamiques» ? C'est dans la réponse à cette question que diffèrent les grandes classes de méthodes de conception. Ainsi, l'analyse de la dynamique dans les méthodes de conception de systèmes d'informations, telles qu'elles sont présentées dans la première partie de ce rapport, réside principalement dans une représentation formelle de l'activité existant dans l'univers modélisé : la dynamique consiste alors en un ensemble de traitements et dans l'expression des interactions entre ces traitements (événements, conditions, contrôle, ...) (Cf. les méthodes SADT ou REMORA). Dans le cas de méthodes de conception destinées à des univers scientifiques ou techniques, la formalisation de la dynamique doit de plus prendre en compte les notions de temps réel, d'exécution parallèle, synchrone ou asynchrone, comme dans la méthode HOOD.

Dans tous les cas, la notion de ***dynamique*** d'une application recouvre à la fois l'expression des *traitements* que doit réaliser cette application et l'expression des *interactions* entre ces traitements. Le modèle d'une méthode de conception doit donc contenir ces deux concepts, tandis que sa démarche fournit des guides méthodologiques pour les analyser.

La section A.VI.1 rappelle comment le modèle de SECOOSSE inclut la formalisation des traitements à l'aide du concept d'opération. La section A.VI.2 décrit comment l'interaction entre les traitements est formalisée dans le modèle orienté objets «standard», puis la section A.VI.3 propose un modèle conceptuel minimal, simplifié, de la dynamique.

A.VI.1. Formalisation des traitements

Dans la logique orientée objets, les traitements que doit réaliser une application sont répartis sur tous les objets qui composent l'application. Chaque objet possède un ensemble de comportements, la coopération de ces comportements permettant de réaliser tous les traitements. La section A.III.2.b page 68 définit, dans le modèle de SECOOSSE, qu'une opération est «une représentation d'un comportement particulier dont sont capables les instances (d'une) classe». C'est

donc à l'aide de ce concept ***d'opération*** que les traitements d'une application sont formalisés.

A.VI.2. La dynamique dans le modèle orienté objets

La dynamique d'une base d'objets, selon le modèle orienté objets, est définie comme une coopération entre les objets présents. Le principe général en est le suivant : durant l'élaboration du résultat d'un traitement, chaque objet n'est capable d'effectuer que les comportements qui lui sont associés. Si un objet a besoin d'un traitement qu'il ne sait pas réaliser, il fait appel à un autre objet qui, lui, saura l'effectuer.
Plus techniquement, la demande de réalisation d'un traitement à un objet par un autre objet est formalisée à l'aide du concept de ***message*** : un message est un couple

<objet destinataire,libellé de message>

Quand l'objet destinataire reçoit le message, il en interprète le libellé pour déterminer le traitement à appliquer et, éventuellement, les différents paramètres de la réalisation de ce traitement.
Il est couramment admis que chaque message reçu aboutit à la réalisation d'une seule opération, mais ce modèle peut facilement être étendu de telle sorte que la réception d'un message entraîne la réalisation de plusieurs opérations.
De la même façon, la définition du message peut, ou non, être étendue à un triplet :

<objet_émetteur,objet_destinataire,libellé de message>.

L'idée générale de la coopération par les messages est de permettre à un objet O_1 de déclencher un traitement T sur un objet O_2 sans connaître exactement les opérations impliquées par le traitement T. Selon la classe de l'objet O_2, un même message entraînera l'exécution d'opérations différentes sans que l'objet O_1 soit informé de la forme exacte du traitement réalisé : cette fonctionnalité est nommée *polymorphisme* des traitements, c'est-à-dire qu'un même traitement peut posséder plusieurs formes.

A.VI.3. La dynamique dans le modèle de SECOOSSE

Le modèle de la dynamique de SECOOSSE proposé ici est une simplification du modèle de la dynamique dans les systèmes orientés objets. Considérons tout d'abord certaines hypothèses et remarques concernant la conception d'un logiciel :

- le modèle de la dynamique fondé sur les messages, présenté dans la section précédente, permet théoriquement aux concepteurs de s'affranchir de contraintes d'implantation. En particulier, ce modèle permet de concevoir une application destinée à fonctionner sur un système parallèle ou en temps réel aussi bien qu'un ensemble de traitements qui seront effectués sur une plate-forme séquentielle. En réalité, il est impensable de concevoir une application dans laquelle plusieurs traitements seraient effectués en parallèle, sans prendre en compte des notions telles que la synchronisation des tâches, la gestion des priorités, et autres spécificités d'un modèle parallèle.
  En fait, nous voulons montrer ici que la modélisation de la dynamique d'une application dépend beaucoup de la façon dont cette application sera implantée.

- Nous constatons que la plupart des systèmes orientés objets existant actuellement permettent seulement une exécution séquentielle des traitements.

- Enfin, nous prenons en compte une remarque de simple bon sens : les produits de la conception feront l'objet d'une implantation.
  Si une *preuve* de ces produits de conception peut être réalisée durant l'étape de conception, alors l'implantation peut être réalisée directement. Si par contre la dynamique conceptuelle ne peut être prouvée, alors c'est durant l'étape d'implantation que seront détectées les «bogues», ce qui amènera à modifier l'analyse de la dynamique, puis à changer l'implantation avant de réaliser une nouvelle vérification.
  Les techniques de vérification de programmes faisant l'objet d'études à part entière (Cf. par exemple [WALLER 91]), nous n'avons pas inclus de *preuves de conception* dans SECOOSSE, réduisant ainsi l'utilisateur au cycle "Conception-Essai-Erreur".

- Dans le même ordre d'idées, nous constatons qu'il existe un contexte spécifique des Systèmes Experts de Maintenance, dans lequel la séparation des intervenants en deux catégories (concepteurs et développeurs) disparaît largement pour faire place à des interventions unifiées dans lesquelles la mise sous forme conceptuelle de l'expertise, la réalisation correspondante et la validation de la base de connaissances obtenue constituent un processus continu.

Nous constatons également que le modèle de la dynamique peut être simplifié grâce au raffinement de la formalisation des objets dans le modèle statique. Le concept d'objets évolutifs, par exemple, permet de supprimer l'expression explicite de contraintes du type «l'opération P ne peut être déclenchée sur l'objet O que si O est dans un état E». De la même façon, le concept de spécialisation du modèle orienté objets automatise le respect de contraintes du type «selon que l'objet O appartient à une classe ou à une autre, le traitement T qui lui est appliqué consiste en T1 ou en T2».

Ces remarques nous amènent à définir un contexte de conception proche d'un univers bien connu, dans lequel l'unité du traitement s'appelle *procédure* ou *fonction* : il s'agit de la programmation dite «structurée». Or, on peut s'interroger sur la différence qui existe entre un couple <objet,opération> et un couple <donnée,procédure> : dans les systèmes orientés objets, les traitements sont véhiculés d'objet en objet tandis que dans la programmation conventionnelle les données sont véhiculées de procédure en procédure. Les différences entre ces deux univers sont tellement réduites que la plupart des langages conventionnels de programmation acceptent désormais une couche orientée objets qui n'est guère plus qu'une extension syntaxique !

C'est pourquoi nous réfutons le modèle de la dynamique fondé sur les messages, pour proposer un modèle beaucoup plus simple, fondé sur des connaissances universelles, dans lequel un objet est considéré comme une donnée et une opération est traitée comme une procédure ou une fonction. Pour cette raison, nous nous permettons de ne pas redéfinir formellement ce modèle (Cf. [LIVERCY 78], [FINANCE 79], [MEYER 84]).

Comme nous le verrons en C.III.2, l'adoption d'un modèle procédural habituel comme modèle de la dynamique présente également l'avantage de fournir l'ensemble des démarches de conception associées. De plus, cette solution permet aux concepteurs de sélectionner l'environnement de leur choix pour exprimer le contenu des opérations ainsi que leurs interactions, les autorisant ainsi à formaliser la dynamique en fonction du contexte exact dans lequel sera réalisée l'application.

A.VII. Récapitulatif

A.VII.1. Primitives

identificateur(I)
nombre(N)
prédéfini(Type)

A.VII.2. Faits

objet(Classe,Objet)

classe_factorisation(Classe)
classe_alternative(Classe)
classe_sémantique(Classe)

alt_initiale(Classe,ClasseAlternative)
transition_possible(CAlt1,CAlt2)

factorise(C,C')
multi_spécialise(C,C')
spécialise(C,C')

cardinalités(Attribut,Min,Max)
domaine(Attribut,Classe)
dual(Attribut1,Attribut2)
valeur_par_défaut(Attribut,Valeur)

opération_publique(Classe,Opération)
opération_privée(Classe,Opération)

nom_propriété(Propriété,Nom)

propriété_publiée(Classe,Propriété)

A.VII.3. Prédicats

classe(C) ← *classe_factorisation(C).*

classe(C) ← *classe_alternative(C).*

classe(C) ← *classe_sémantique(C).*

a_pour_alt(CS,CA) ← *alt_initiale(CS,CA).*

a_pour_alt(CS,CA) ← *transition_possible(CA,CA')*
et a_pour_alt(CS,CA').

attribut_simple(C,A) ← ∃*I* ∃*N1* ∃*N2* ∃*D*
identificateur(I)
et nom_propriété(A,I)
et nombre(N1)
et nombre(N2)
et cardinalités(A,N1,N2)
et prédéfini(D)
et domaine(A,D).

attribut_relation(C,A) ← ∃*I* ∃*N1* ∃*N2* ∃*C'*
identificateur(I)
et nom_propriété(A,I)
et nombre(N1)
et nombre(N2)
et cardinalités(A,N1,N2)
et classe(C')
et domaine(A,C').

attribut_relation(C,A) ← ∃*I* ∃*N1* ∃*N2* ∃*C'* ∃*A'*
identificateur(I)
et nom_propriété(A,I)
et nombre(N1)
et nombre(N2)
et cardinalités(A,N1,N2)
et classe(C')
et domaine(A,C')
et attribut_relation(C',A')
et dual(A,A')
et dual(A',A).

opération(C,O) ← ∃*I*
identificateur(I)
et nom_propriété(O,I)
et opération_publique(C,O).

opération(C,O) ← ∃*I*
identificateur(I)
et nom_propriété(O,I)
et opération_privée(C,O).

attribut(C,A) ← *attribut_simple(C,A).*

attribut(C,A) ← *attribut_relation(C,A).*

propriété(C,P) ← *attribut(C,P)*
et ¬(homonyme_existe(C,P)).

propriété(C,P) ← *opération(C,P)*
et ¬(homonyme_existe(C,P)).

propriété(C,P) ← *propriété_publiée(C,P)*
et ¬(homonyme_existe(C,P)).

propriété(C,P) ← *propriété_héritée(C,P)*
et ¬(propriété_surchargée(C,P)).

voit(O,O,P) ← ∃*C*
classe(C)
et objet(C,O)
et propriété(C,P).

voit(O',O'',P) ← ∃*C''*
classe(C'')
et objet(C'',O'')
et opération(C'',P).

propriété_héritée(C,P) ← $\exists C'$
classe_factorisation(C')
et factorise(C',C)
et propriété(C',P).

propriété_héritée(C,P) ← $\exists C'$
classe_sémantique(C')
et multi_spécialise(C,C')
et propriété(P,C').

propriété_héritée(C,P) ← $\exists C'$
classe_sémantique(C')
et spécialise(C,C')
et propriété(C',P).

homonyme_existe(C,P) ← $\exists P'$
même_nom(P,P')
et propriété(C,P').

homonyme_existe(C,P) ← $\exists P' \exists CA$
même_nom(P,P')
et a_pour_alt(C,CA)
et propriété(CA,P').

propriété_surchargée(C,P') ← $\exists C' \exists C'' \exists P''$
classe(C')
et propriété(C',P')
et classe(C'')
et propriété(C'',P'')
et plus_proche(C,C',C'')
et même_nom(P',P'').

plus_proche(C,C,C'') ←

plus_proche(C,C',C'') ← *a_pour_alt(C,C')*
et $\neg$*(a_pour_alt(C,C'')).*

plus_proche(C,C',C'') ← *factorise(C',C)*
et $\neg$*(a_pour_alt,C,C''))*
et $\neg$*(factorise(C'',C)).*

plus_proche(C,C',C'') ← *multi_spécialise(C,C')*
et $\neg$*(a_pour_alt,C,C''))*
et $\neg$*(factorise(C'',C))*
et $\neg$*(multi_spécialise(C,C'')).*

plus_proche(C,C',C'') ← *spécialise(C,C')*
et $\neg$*(a_pour_alt,C,C''))*
et $\neg$*(factorise(C'',C))*
et $\neg$*(multi_spécialise(C,C'')).*

Chapitre B
Le langage conceptuel de SECOOSSE

Comme [LISSANDRE 90] le fait remarquer, «*si l'on veut qu'une méthode apporte les avantages escomptés, il est absolument nécessaire de la choisir en fonction des habitudes et des procédures existantes, ainsi que de l'expérience acquise par ses futurs utilisateurs : une évolution est toujours plus efficace et moins coûteuse qu'une révolution.*» En effet, tout informaticien possède une «culture de base» qui doit être exploitée au maximum dans les nouvelles procédures qu'il doit appliquer. C'est ce souci qui a guidé notre choix des langages utilisés pour représenter le modèle de SECOOSSE.

Un réflexe naturel porte les utilisateurs de systèmes orientés objets à représenter sous forme de graphes les connaissances qu'ils formalisent. Pour citer [HABRIAS 90], «*il semble que, dans le monde de l'informatique, les graphiques aient des vertus télépathiques ; il suffirait de faire des ronds ou des rectangles pour que le schéma soit qualifié de conceptuel ...* (suit la description du BUBTANGLE, qui unifie les adeptes du BUBBLE (la bulle) et du RECTANGLE...)». Cette remarque n'est pas à négliger, en particulier parce qu'elle met indirectement en évidence ce réflexe qui porte les concepteurs à exploiter les formalisations graphiques.

A partir de ces constatations, il nous a semblé indispensable d'offrir aux concepteurs un langage graphique adapté au modèle de SECOOSSE. Plutôt que de proposer une nouvelle terminologie graphique (une de plus ...) nous avons préféré exploiter une symbolique existante. Or il existe de très nombreuses propositions destinées à illustrer des formalisations du monde réel (nous en avons par exemple présenté trois dans la première partie de ce rapport). Notre propos ici *n'est pas* de mettre en concurrence ces propositions, mais d'intégrer un langage graphique aussi simple que possible à notre méthode. Aussi, plutôt qu'une étude comparative des avantages et inconvénients de toutes sortes de réseaux sémantiques, nous proposons dans ce chapitre un modèle graphique adapté au modèle de SECOOSSE, fondé sur le modèle ***Information Analysis*** (IA), proposé dans la méthode NIAM (Nijssen's Information Analysis Method), présentée dans [HABRIAS 88], [HABRIAS 89], [HABRIAS 91a]. Notons que ce

modèle a déjà été utilisé dans le contexte d'une méthode de conception (Cf. [HABRIAS 91b]).

Le paragraphe B.I en présente les caractéristiques principales et en montre l'adéquation au modèle de SECOOSSE. Nous déduirons cependant de cette étude que le modèle IA n'est pas suffisamment riche. Aussi proposerons-nous dans le paragraphe B.II des enrichissements qui permettront d'exprimer finement la sémantique de l'application.

B.I. Langage conceptuel graphique : le modèle IA

Le langage graphique IA est présenté, entre autres, dans [VERHEIJEN 82] et [HABRIAS 88]. Basé sur un réseau sémantique, il comporte notamment des représentations puissantes de relations entre objets et de contraintes.

B.I.1. Réseau sémantique de IA

Le langage conceptuel graphique IA repose sur une représentation, dans un modèle binaire, d'un énoncé exprimé en langage naturel. Une phrase en langage naturel comporte des informations sur des entités réelles ou abstraites. Cependant, cette phrase ne met pas en évidence les objets impliqués, mais les référence par des noms (des mots) qui, comme le précise [TABOURIER 86], «sont fondamentalement distincts des objets qu'ils représentent : le mot "chien" ne mord pas, le mot "épine" ne pique pas, le mot "rose" ne sent rien». Dans IA, deux catégories de références sont distinguées :

- des mots «abstraits» référencent des entités dont la description ne peut être directement donnée : par exemple, le mot «chien» décrit une entité ; cette entité possède quatre pattes, une truffe et deux oreilles, mais cette description ne fait appel qu'au sens commun, elle n'est pas explicitement contenue dans le mot «chien». Ces mots abstraits sont appelés ***NOLOTs*** dans le modèle IA (pour NOt Lexical ObjecT).
- par opposition, il existe des mots qui décrivent directement un concept : «longueur», «nombre de pattes» sont des mots dont la description est directement donnée par leur sémantique. IA nomme ces mots des ***LOTs*** (Lexical ObjecT). Selon le modèle, les LOTs construisent en particulier une représentation totale ou partielle des NOLOTs.

Toujours dans le domaine du langage naturel, les auteurs du modèle IA postulent que toute phrase, aussi complexe qu'elle soit, peut être réécrite en un ensemble

de phrases de la forme *Sujet Verbe Complément*. Le sujet et le complément sont alors des entités, qui sont formalisées à l'aide des NOLOTs. Par contre, le verbe est un mot particulier qui exprime l'idée qui associe les deux entités. Cette notion d'idée d'association entre NOLOT appartient donc également au modèle :

- lorsqu'il s'agit d'une relation entre deux entités formalisées à l'aide de NOLOTs, cette relation porte le nom d'***idée-type***. Dans cette idée type, chacun des NOLOTs joue un ***rôle***.
- lorsqu'il s'agit d'une relation qui attache un LOT à un NOLOT, la relation porte le nom de ***pont de dénomination***. Dans ce cas, le rôle du NOLOT par rapport au LOT est implicitement un rôle de *possession*, le LOT ayant un rôle de *dénomination*, ou de *caractérisation*, du NOLOT.

Enfin, le modèle de données IA possède une composante qui ne dérive pas directement du langage naturel : en effet, IA inclut une relation de *généralisation / spécialisation* entre NOLOTs.

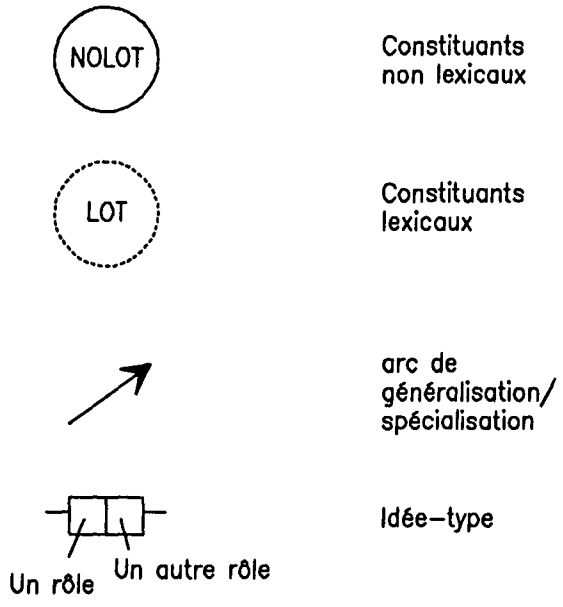


Figure 10 : Représentation graphique des concepts de IA;

Ces concepts de base du modèle IA sont formalisés graphiquement comme le montre la figure 10 :

- les NOLOTs sont représentés par un cercle plein
- les LOTs sont représentés par un cercle pointillé

- les idées entre NOLOTs sont représentées par deux boîtes accolées. Chacune de ces boîtes est destinée à contenir un mot qui décrit le rôle dans l'idée du NOLOT correspondant.
- les ponts de dénomination ont la même forme que les idées-types mais il n'est pas indispensable de mentionner les rôles du LOT et du NOLOT, ces rôles étant toujours identiques (POSSEDE et CARACTERISE).
- l'arc de généralisation/spécialisation est représenté par une flèche entre deux NOLOT, le NOLOT le plus général étant désigné par la flèche.

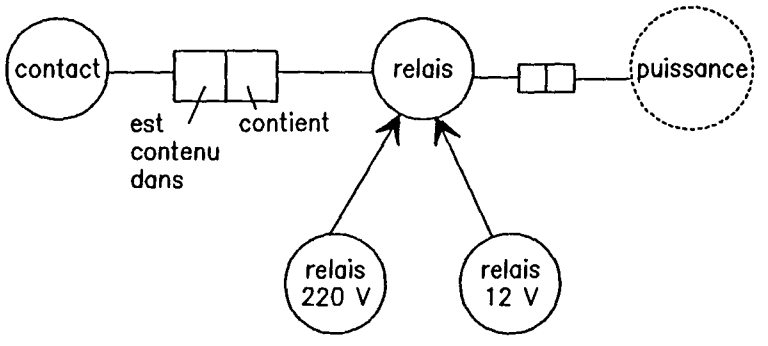


Figure 11 : Exemple de schéma conceptuel

La figure 11 montre un exemple de schéma IA. Dans cet exemple, RELAIS et CONTACT sont deux NOLOTs. Ces NOLOTs sont liés entre eux par une idée, dans laquelle le rôle de RELAIS est CONTIENT et le rôle de CONTACT est EST_CONTENU_DANS. Cette idée provient, par exemple, de la phrase :

*«Un relais **contient** des contacts»*

la réciproque étant

*«Un contact **est contenu dans** un relais».*

PUISSANCE est un LOT. On observe que ce LOT, qui compose la description de RELAIS, est attaché à ce NOLOT par une sorte de petite idée, nommée dans IA ***Pont de Dénomination***. Un pont de dénomination correspond à une idée entre un NOLOT et un LOT, le rôle du NOLOT étant «a» et celui du LOT étant «de» :

*«Un relais **a** une puissance», «Une puissance est celle **de** un relais»*

B.I.2. Contraintes d'intégrité sur les relations

Le modèle IA permet l'expression de nombreuses contraintes exprimées sur les relations qui existent entre les NOLOT : ces contraintes sont donc attachées aux idées-type ou aux rôles. Elles sont toujours exprimées en fonction des occurrences des objets dans les idées-types, c'est-à-dire qu'elles portent sur la présence, ou l'absence, d'une instance dans un rôle.

B.I.2.a) Contrainte d'unicité sur les rôles

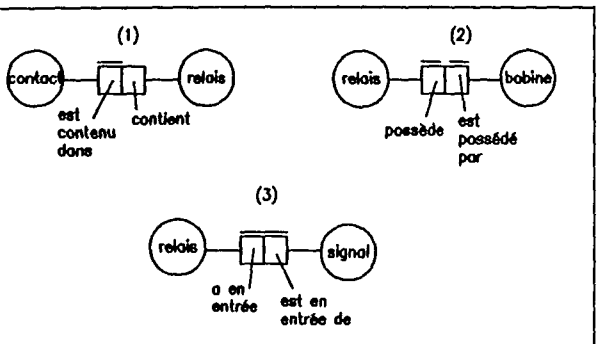


Figure 12 : Contraintes d'unicité sur les rôles

Il est possible d'associer une contrainte d'unicité à un rôle. Cette contrainte, représentée par une flèche (parfois une simple barre) au dessus du rôle, signifie qu'un objet au maximum peut être concerné par ce rôle. Ainsi, dans le cas (1) de la figure 12, un contact EST_CONTENU_DANS ***au maximum*** un relais.

Par opposition, l'absence de contrainte sur un rôle signifie que plusieurs objets peuvent être concernés : par exemple, toujours dans le cas (1), un relais CONTIENT ***des*** contacts.

Naturellement, une contrainte d'unicité peut être exprimée sur deux rôles réciproques : dans le cas (2), un relais POSSEDE ***au maximum*** une bobine et une bobine EST_POSSEDEE_PAR ***au maximum*** un relais.

Dans le cas où aucun des rôles ne possède de contrainte d'unicité, plutôt que de ne porter aucune contrainte, le modèle IA propose de tracer une flèche -ou une barre- sur la totalité de l'idée : dans le cas (3), un relais A_EN_ENTREE (zéro, un ou plusieurs) signaux et un signal EST_EN_ENTREE_DE (zéro, un ou plusieurs) relais.

B.I.2.b) Contrainte d'unicité entre idées

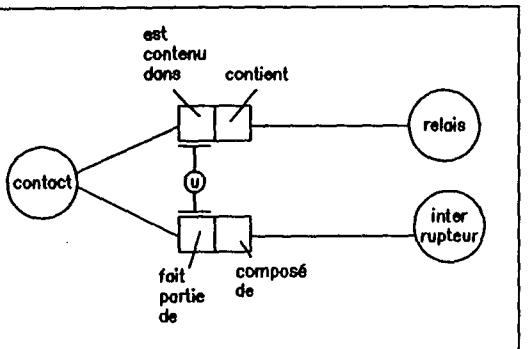


Figure 13 : Contrainte d'unicité entre idées

La contrainte d'unicité entre idées permet d'exprimer qu'un objet est associé à un seul autre objet bien que plusieurs idées existent. Cette contrainte est représentée par un «∪», cerclé, lié aux rôles, comme indiqué figure 13. Sur cette figure, cette contrainte indique qu'un contact, soit EST_CONTENU_DANS un relais, soit FAIT_PARTIE_DE un interrupteur, mais ne

peut être associé à la fois à un relais et à un interrupteur. Par conséquent, l'intersection des populations des rôles CONTIENT et COMPOSE_DE est vide.

B.I.2.c) Contrainte de totalité sur les rôles

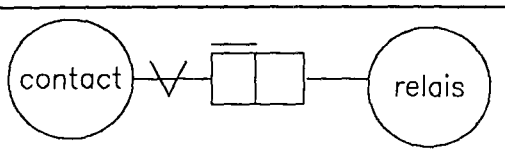


Figure 14 : Contrainte de totalité

La contrainte de totalité indique que toute instance du NOLOT apparaît ***au moins une fois*** dans la population du rôle qui la porte. Le nombre maximum d'apparitions est fixé par la contrainte d'unicité du rôle. Cette contrainte est représentée par un «∀» sur l'arc qui relie le rôle au NOLOT : la figure 14 indique que ***tout*** contact est_contenu_dans un relais. Remarquons qu'une contrainte de totalité ne peut jamais être associée au rôle d'un LOT dans un pont de dénomination. En effet, par définition, une instance d'un LOT ne peut exister sans qu'il y ait au moins une instance de NOLOT possédant cette instance du LOT.

B.I.2.d) Contrainte de totalité entre relations

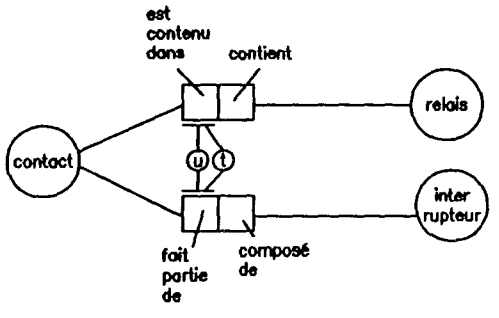


Figure 15 : Contrainte de totalité entre relations

La contrainte de totalité entre rôles de relations différentes exprime le fait qu'une instance du NOLOT associé à ces rôles doit apparaître au moins une fois dans une des populations de ces rôles.
Cette contrainte est représentée par un «T», cerclé, lié aux rôles. La figure 15 indique qu'un contact, outre qu'il est lié soit à un relais soit à un interrupteur (contrainte «∪»), doit ***obligatoirement*** être lié à l'un des deux.

B.I.2.e) Contrainte d'égalité entre rôles

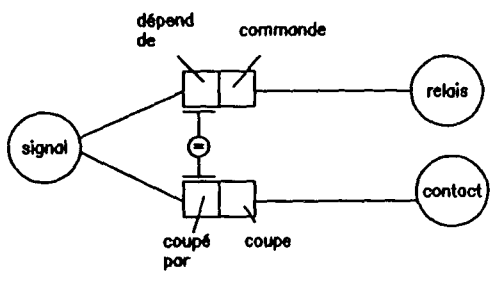


Figure 16 : Contrainte d'égalité entre rôles

La contrainte d'égalité indique que toute instance du NOLOT qui apparaît dans l'un des rôles doit apparaître dans tous les rôles liés par la contrainte.
Cette contrainte est représentée par un «=» cerclé, lié aux rôles ou aux idées-type. La figure 16 indique que tout signal COUPE_PAR un contact DEPEND également d'un relais.

B.I.2.f) Contrainte d'exclusion entre rôles

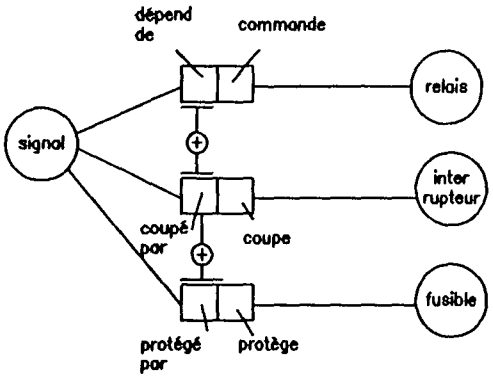


Figure 17 : Contrainte d'exclusion entre rôles

La contrainte d'exclusion indique que toute instance du NOLOT ne peut apparaître que dans l'un des rôles. A la différence de la contrainte d'unicité entre rôles («∪»), cette contrainte n'impose pas qu'il n'existe qu'une relation ; par contre elle interdit la simultanéité de certaines associations d'objets.

Cette contrainte est représentée par un «+» cerclé, lié aux rôles ou aux idées-type. Ainsi, la figure 17 indique que :

- un signal peut être à la fois DEPENDRE_DE un relais et être PROTEGE_PAR un fusible car il n'y a pas de contrainte d'exclusion entre ces rôles
- aucun signal ne peut à la fois DEPENDRE_DE un relais et être COUPE_PAR un interrupteur
- aucun signal ne peut être à la fois COUPE_PAR un interrupteur et PROTEGE_PAR un fusible.

B.I.2.g) Contrainte d'inclusion entre rôles

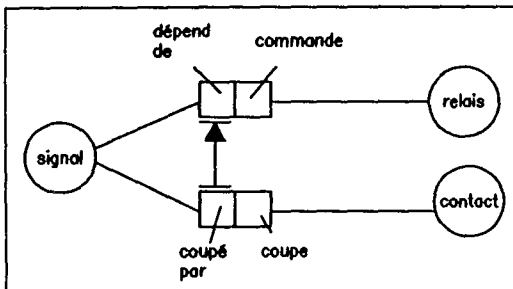


Figure 18 : Contrainte d'inclusion

Elle signifie que toute instance apparaissant dans le rôle inclus doit apparaître aussi dans le rôle incluant. Elle est représentée par une flèche allant du rôle inclus vers le rôle incluant. Ainsi, la figure 18 indique que si un signal est COUPE_PAR un contact, alors ce signal DEPEND_DE un relais. Par contre, par opposition avec la contrainte d'égalité, un signal peut dépendre d'un relais sans être coupé par un contact.

B.I.3. Contraintes d'intégrité sur l'héritage

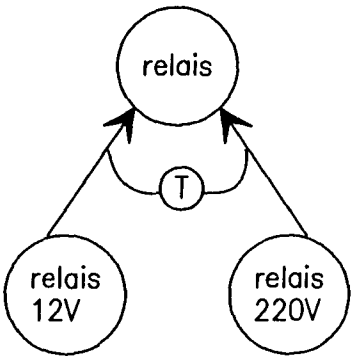


Figure 19 : Contrainte de totalité du sous-typage

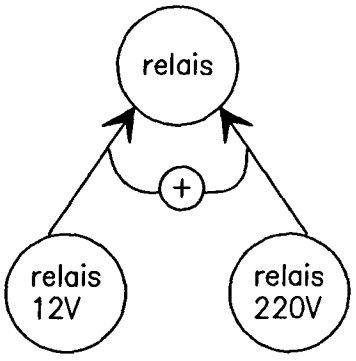


Figure 20 : Contrainte d'exclusion du sous-typage

B.I.3.a) Contrainte de totalité

Elle exprime le fait que toute instance de la super-classe doit être instance d'au moins une des sous-classes. Cette contrainte est représentée graphiquement comme indiqué sur la figure 19. Cette figure indique que tout relais est soit un RELAIS_12v, soit un RELAIS_220V. Une conséquence de cette contrainte est qu'il ne peut exister d'objets RELAIS.

B.I.3.b) Contrainte d'exclusion

Elle exprime le fait qu'une instance de la super-classe ne peut être instance de plus d'une sous-classe parmi celles qui sont contraintes. Cette contrainte est représentée graphiquement comme indiqué sur la figure 20, qui indique qu'un RELAIS ne peut être à la fois RELAIS_12V et RELAIS_220V. La contrainte d'exclusion n'implique pas la contrainte de totalité.

B.I.4. Adéquation et lacunes de IA

Nous allons montrer dans cette section que le langage Information Analysis est adapté pour la représentation du modèle de données de SECOOSSE, en décrivant successivement comment représenter les classes, les attributs et les relations de spécialisation, de factorisation et de spécialisation multiple.

B.I.4.a) Classe

Figure 21 : Représentation de la classe Relais

Les NOLOTs de la méthode NIAM correspondent exactement à la notion de classe de SECOOSSE. Par conséquent, le langage conceptuel de SECOOSSE utilise le graphisme des NOLOTs pour représenter les classes sémantiques (figure 21). Les classes non sémantiques, alternatives ou de factorisation, sont représentées avec le même symbole, mais les bords du cercle sont épais.
Pour augmenter la lisibilité des schémas conceptuels, il peut être nécessaire de dupliquer la représentation d'une classe. Nous proposons de différencier clairement les définitions de classes des duplications qui en sont faites, en remplissant le fond de la classe dupliquée d'un motif quelconque (hachures, gris, ...) et en rappelant, dans cette classe dupliquée, le nom du schéma dans lequel la classe est définie.

B.I.4.b) Attribut-valeur

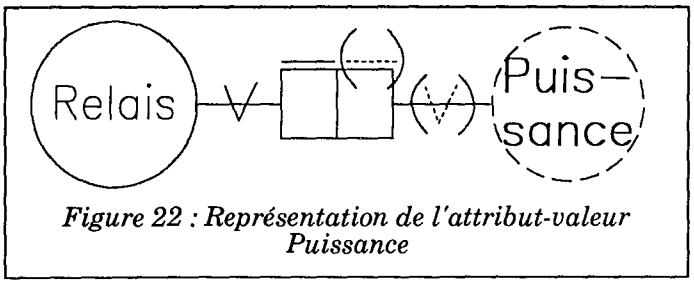


Figure 22 : Représentation de l'attribut-valeur Puissance

La notion de LOT, telle qu'elle est présentée dans NIAM, et la notion d'attribut-valeur, telle que notre modèle la définit, représentent le même concept. C'est pourquoi nous retenons le graphisme des LOTs pour représenter les attributs-valeur (figure 22). Il n'est pas nécessaire d'exprimer la contrainte de totalité «V» ou la contrainte d'unicité sur les rôles des attributs dans les ponts de dénomination ; en effet, d'après le modèle orienté objet, une valeur donné (considérée en tant qu'«instance» de l'attribut) ne peut être attachée qu'à un objet. Par contre les cardinalités minimum et maximum de l'attribut sont mentionnées à l'aide des contraintes de totalité et d'unicité sur le rôle de l'objet. Dans un but de simplification des schémas conceptuels, nous suggérons de ne pas nommer les rôles des ponts de dénomination, ces rôles étant usuellement «a» et «de».

B.I.4.c) Attribut-relation

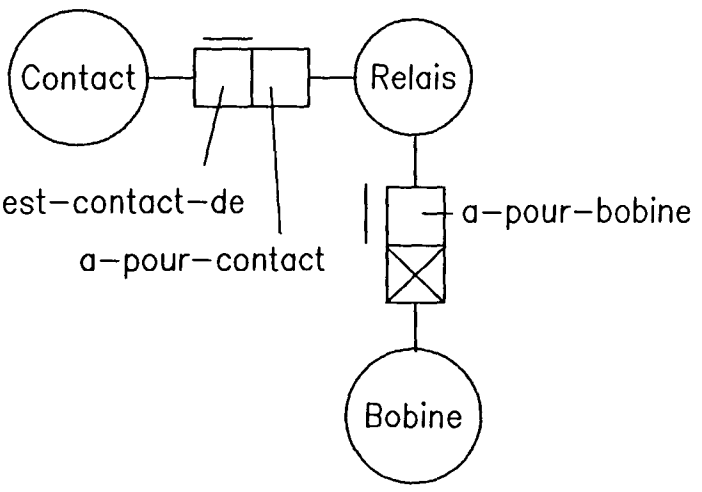


Figure 23 : Représentation des attributs-relation

Un attribut-relation permet de mémoriser, pour un objet donné, la référence d'un autre objet. Les attributs-relation, nous l'avons vu, sont associés à une classe, leur domaine étant l'ensemble des instances d'une autre classe.
Or la notion d'idée-type, telle que définie dans NIAM, ressemble fortement à deux attributs-relation réciproques, représentés simultanément dans un même graphisme. Nous suggérons de remplacer les flèches sur les rôles et idées (contraintes de cardinalité) par des traits pour simplifier et alléger l'écriture.
Nous pouvons distinguer deux cas (figure 23) :

- l'utilisateur désire réellement deux attributs-relation duaux, manipulés réciproquement. Dès lors, le graphisme de IA est parfaitement adapté, puisqu'il met en évidence les domaines des deux attributs et leurs contraintes de cardinalité. Nous retiendrons donc ce graphisme.
- l'utilisateur ne désire qu'un attribut-relation. Nous choisissons de conserver le graphisme des idées-type, seul le rôle correspondant à l'attribut étant nommé et possédant des contraintes ; l'autre rôle est remplacé par une croix. Cette solution permet d'exprimer simplement le domaine de l'attribut relation. Les contraintes sur le rôle dual sont alors, naturellement, inutiles.

B.I.4.d) Factorisation, spécialisation, spécialisation multiple

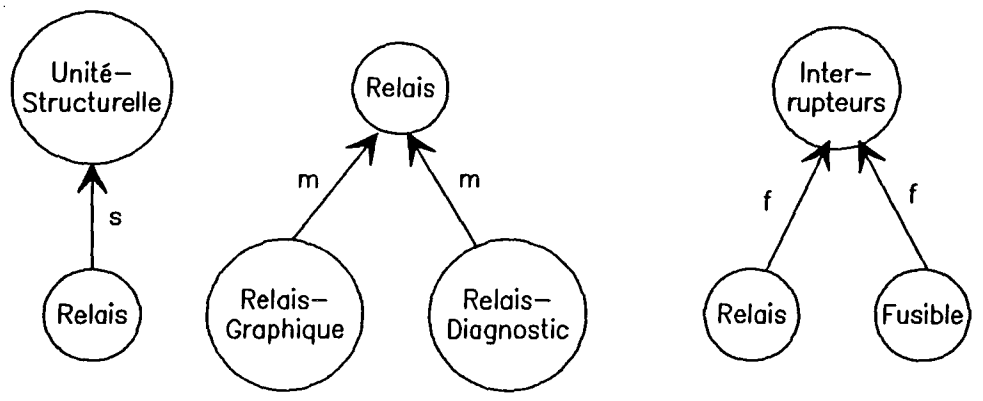


Figure 24 : Représentation de la spécialisation, de la spécialisation multiple et de la factorisation

Le langage graphique IA ne permet de noter qu'un type d'héritage. Or, nous avons défini trois sortes de relations d'héritage statique entre classes : la factorisation, la spécialisation et la spécialisation multiple.
Nous suggérons que, plutôt que d'utiliser des arcs différents par leur graphisme, il est préférable d'utiliser un seul pictogramme, chaque catégorie d'arc possédant une étiquette différente. Ainsi, nous proposons les arcs "s" de spécialisation, les arcs "m" de multi-spécialisation et les arcs "f" de factorisation (figure 24).

B.I.4.e) Les contraintes de NIAM

Les contraintes de NIAM sur les rôles et les idées-type sont absolument utilisables dans le contexte d'une méthode orientée objets, puisqu'elles permettent de détailler la description des attributs-relation.
Par contre, les contraintes sur l'héritage ne présentent guère d'intérêt dans ce contexte, dans la mesure où exclusion et totalité de l'instanciation des sous-classes sont explicitement exprimées dans le modèle orienté objets.

B.I.4.f) Conclusion

Les concepts majeurs de SECOOSSE pouvant être représentés à l'aide de IA, nous retenons ce langage graphique et son modèle sous-jacent pour les schémas conceptuels de notre méthode. Certains concepts ne sont cependant pas représentés dans ce langage graphique : les opérations, les classes dont la sémantique évolue dans le temps n'appartiennent pas aux concepts de NIAM. Nous devons donc inclure de nouvelles entités graphiques pour représenter la totalité des concepts de notre modèle : tel est le but du paragraphe suivant.

B.II. Nos propositions pour le langage conceptuel graphique

Nous avons montré dans le paragraphe précédent que le modèle de données Information Analysis est adapté à la représentation des concepts du modèle orienté objets. L'ayant retenu pour la représentation des concepts de SECOOSSE, nous devons cependant l'enrichir de pictogrammes pour représenter la totalité de notre modèle. C'est pourquoi nous ajoutons, aux conventions graphiques adoptées précédemment, les notions d'opération et d'évolution des objets.

B.II.1. Représentation des opérations

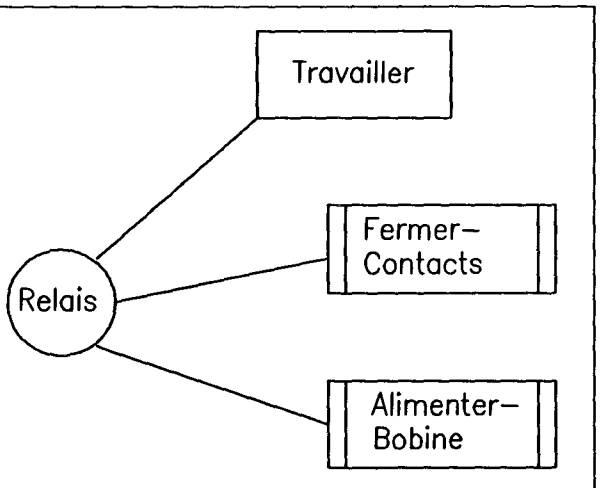


Figure 25 : Représentation des opérations

Nous avons vu qu'une opération peut être privée aux objets d'une classe, ou publique, c'est à dire déclenchable par les objets d'une autre classe. Nous proposons donc deux primitives graphiques pour la représentation des opérations. Dans la relation entre une opération et une classe, le rôle de chacune est identique quelle que soit l'opération et quelle que soit la classe ; de même nous savons qu'une opération ne peut exister qu'une fois sur chaque classe. La notion de pont de dénomination et de rôle qui apparaît dans l'attachement des attributs aux classes perd donc son intérêt.

Le graphisme que nous proposons pour les opérations publiques est constitué d'un rectangle simple, contenant le nom de l'opération. Ainsi, la figure 25 met en évidence, simplement par le graphisme, que l'opération TRAVAILLER est publique, donc peut être déclenchée par tout objet, même s'il n'est pas instance de RELAIS. Pour les opérations privées, nous proposons un rectangle dont deux côtés sont doublés : de la sorte, nous observons que les opérations FERMER_CONTACTS et ALIMENTER_BOBINE ne peuvent être déclenchées que depuis une opération qui est elle-même attachée à la classe RELAIS. Enfin, l'attachement d'une opération à une classe est matérialisé par un trait simple. Il n'est pas utile de choisir un graphisme plus élaboré, dans la mesure où l'on sait que :

- une opération est définie pour toutes les instances de la classe à laquelle elle est attachée

- chaque instance d'une classe connaît une et une seule définition de chacune des opérations définies sur ou héritées par cette classe : cette contrainte, étant systématique, n'est pas utile à la lecture d'un schéma.

B.II.2. Représentation de l'évolution

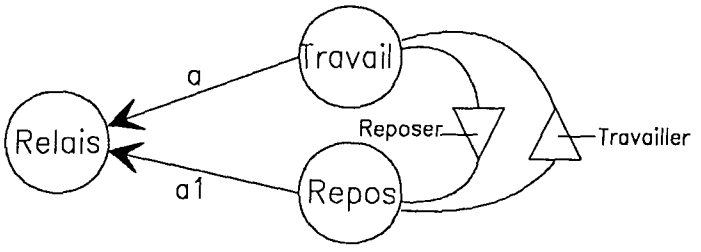


Figure 26 : Représentation de l'évolution d'un objet

Pour matérialiser graphiquement les relations qui existent entre classes sémantiques et classes alternatives (prédicat *a_pour_alt*), nous choisissons un arc simple qui rappelle les différents arcs d'héritage (figure 26). Cet arc est libellé par la lettre «a». Le modèle indique que chaque instance d'une telle classe est simultanément instance d'une et une seule des classes alternatives liées. Pour matérialiser, sur le schéma conceptuel, quelle est la classe alternative qui doit être considérée lors de l'instanciation de la classe évolutive (prédicat *alt_initiale*), nous choisissons de libeller l'arc entre la classe évolutive et la classe alternative par «a1». Cet arc dénote la classe alternative initiale (par défaut) de chaque instance de la classe évolutive.

L'évolution individuelle d'un objet doit être définie *in extenso* dans le schéma conceptuel. Cette évolution consiste en changements d'alternatives durant la vie de l'objet (prédicat *transition_possible*). Le concept de transition autorisée entre classes alternatives est représenté graphiquement, à l'aide de triangles dont la base est orientée vers l'alternative de départ et la pointe vers l'alternative d'arrivée. Ces triangles peuvent éventuellement porter le nom d'une opération qui devra être déclenchée pour réaliser cette transition. Cette opération est alors publique.

Chapitre C
La démarche SECOOSSE

La création d'un logiciel peut être considérée comme une opération réalisée en deux grandes étapes : la spécification et la conception précèdent le codage ; l'installation et les tests succèdent au codage. De façon évidente, une méthode de conception a pour rôle d'assister les intervenants de la première étape.

Le principe de base de la démarche de SECOOSSE relève de la constatation suivante : à partir d'une seule représentation ***objective*** de la réalité on peut réaliser plusieurs applications. En effet nous ne situons pas la réutilisabilité à l'échelle de la classe : l'héritage alourdit et contraint beaucoup trop cette réutilisabilité, en imposant le transport d'un grand nombre de super-classes, par spécialisation comme par factorisation. Au contraire, nous observons qu'une base de connaissance *complète* peut être réutilisée, à condition que les connaissances concernées soient objectives. Des connaissances objectives peuvent ne pas être complètes (dans le domaine du diagnostic, la composition des façades métalliques importe peu), mais elles doivent être indépendantes de l'application qui les utilise.

Dans le cas des SE portant sur des équipements électromécaniques, la représentation objective est en général la description de l'équipement lui-même. Quelles sont alors les connaissances supplémentaires qui doivent être modélisées pour réaliser un SEM ?

- le technicien peut décrire son raisonnement de maintenance, incluant une méthode de diagnostic et des heuristiques de réparation
- le technicien peut également exprimer sa vision de l'utilisation du SE. C'est lui, en effet, qui est destiné à utiliser le produit logiciel ! Or c'est également lui qui connaît le temps disponible entre deux opérations, l'aisance d'accès au matériel informatique depuis les endroits à mesurer, etc. En fonction des cas, l'interface du SE devra être très directive et conviviale car le technicien effectue les manipulations sans s'éloigner de l'ordinateur ; ou au contraire l'interface devra être très souple pour permettre au technicien de saisir

plusieurs informations qu'il a recueillies en évoluant dans l'équipement maintenu ...

- une autre forme de connaissance est fournie par les personnes qui sont destinées à saisir et à maintenir la base de connaissance : il s'agit alors de l'interface de saisie qui sera à nouveau variable selon, par exemple, que les composants traités sont plutôt électromécaniques, électroniques, voire micro-électroniques (circuits intégrés).

En résumé, quatre spécifications doivent être fournies :

- la structure
- la méthode de diagnostic
- l'interface de maintenance
- l'interface de gestion

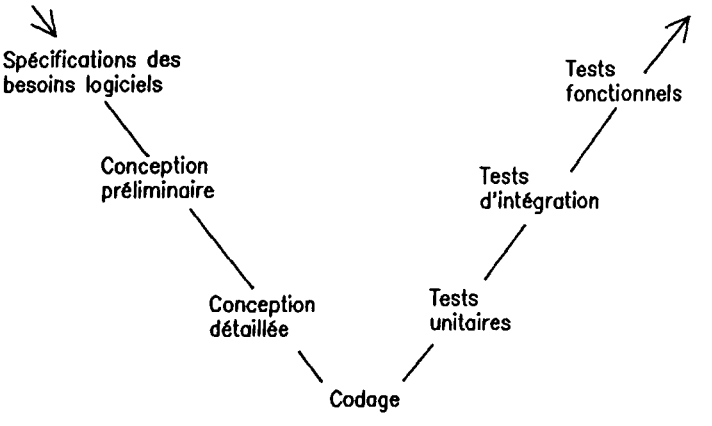


Figure 27 : Le cycle de vie logiciel

C'est à chacun de ces domaines de connaissance que nous appliquons la démarche de conception présentée en figure 27 : pour chaque domaine, une spécification des besoins est réalisée, qui aboutit à un ***cahier des charges*** ; la construction du cahier des charges est présentée en C.I.

L'étape de conception préliminaire permet ensuite, à partir du cahier des charges, d'identifier les concepts et de créer un schéma conceptuel faiblement structuré : cette étape est présentée en C.II.

Le paragraphe C.III présente un guide méthodologique pour obtenir des schémas conceptuels fortement structurés, qui amènent enfin à l'étape de conception détaillée, présentée en C.IV, qui permet d'unifier les différents domaines de connaissance et d'analyser la dynamique de l'application.

C.I. Création d'un cahier des charges

La spécification des besoins logiciels, dans le contexte des systèmes de maintenance, est réalisée en amont de l'utilisation de SECOOSSE. Elle fait intervenir un personnage particulier, parfois appelé *cogniticien*, dont le rôle est de recueillir l'expertise et de la mettre sous forme de textes bruts, reproductions intégrales des entretiens qui ont été menés. Un guide méthodologique peut être fourni pour cette étape, tel que la méthode OBA présentée dans [GIBSON 90] ou [PERNICI 90] ; cependant cette assistance méthodologique à la prise d'expertise n'est pas incluse dans SECOOSSE.
A partir des textes résultant des entretiens avec les experts, le cogniticien confronte les différentes spécifications obtenues. Un accord entre tous les experts est indispensable pour aboutir à un texte unique, linéaire, qui n'est plus sous forme d'entretiens mais au contraire structuré par thème et complet : il s'agit de ce qui est couramment nommé *cahier des charges.*

C.II. Mise en forme du cahier des charges

Le cahier des charges consiste alors en un texte en langage naturel au sujet duquel la seule certitude est que les experts en sont satisfaits. L'étape de conception préliminaire permet de transformer ce texte en un ensemble de phrases qui identifient les concepts et leurs relations. Elle offre également un support méthodologique pour la recherche de la complétude de la spécification obtenue.
La démarche de conception préliminaire est fondée sur une réécriture complète du cahier des charges sous une forme à peu près normalisée. Naturellement, le volume des informations manipulées peut sembler prohibitif à un utilisateur néophyte qui se lancerait dans la création d'un SEM2G (quoiqu'il existe probablement peu de personnes non averties qui se lancent dans une telle tâche). Notons cependant que les règles de transformation proposées sont simples et relèvent beaucoup de l'intuition : très rapidement, les concepteurs sont à même d'assumer mentalement la mise en forme du cahier des charges, en ne notant réellement que les questions qui permettront d'assurer la complétude de la spécification obtenue.

C.II.1. Identification des concepts

La reformulation du cahier des charges consiste avant tout en une mise sous forme binaire de toutes les phrases significatives. Chacune de ces phrases doit de plus être exprimée en fonction du caractère temporaire ou définitif des faits exprimés : nous suggérons d'adjoindre systématiquement les termes *parfois*, *toujours* et *jamais* dans chaque phrase binaire. Aussi souvent que possible, les phrases réciproques ou duales doivent être également exprimées. Si elles ne peuvent l'être, le cahier des charges doit être modifié après prise d'avis des experts.

Exemple

- *la phrase initiale est «Une liaison commence et se termine par une jonction qui est une borne d'une unité structurelle, qui peut par ailleurs en posséder d'autres.»*
- *cette phrase peut être reformulée :*
  - *«une liaison commence toujours par une jonction»*
  - *«une liaison se termine toujours par une jonction»*
  - *«une jonction est toujours une borne d'une unité structurelle»*
- *la formulation réciproque amène aux questions et phrases suivantes :*
  - *«une jonction commence toujours combien de liaisons ?»*
  - *«une jonction termine toujours combien de liaisons ?»*
  - *«une unité structurelle a toujours pour bornes des jonctions».*

Parmi toutes les phrases du cahier des charges, certaines peuvent difficilement être mises sous forme binaire, en particulier parce qu'elles font intervenir des expressions conditionnelles. Ces phrases sont alors laissées dans leur expression originale.

Exemple :

- *«Un relais est en panne si sa commande est passée et sa sortie est incorrecte»*
- *«lorsqu'une liaison est innocente on peut déduire que toutes ses liaisons amont sont également innocentes»*

Il est nécessaire de vérifier que toutes les phrases qui expriment une évolution sont soumises à des conditions. En effet, le modèle de SECOOSSE ne peut prendre en compte que des évolutions discrètes.

Exemple :

- *la phrase «la représentation du relais passe progressivement du bleu papillon à l'orange des mers du sud» ne représente pas une évolution discrète et ne pourra être traitée par SECOOSSE.*
- *la phrase «lorsque le relais devient innocent sa représentation est coloriée en vert» est conservée telle quelle.*

Notons cependant que ces phrases conditionnelles doivent malgré tout être remplacées au maximum par des phrases binaires.

Exemple :

La phrase «Un relais contient des contacts et une bobine, il est en panne si sa commande est passée et sa sortie est incorrecte» doit devenir :

. *«Un relais contient toujours des contacts»*
. *«Un contact est toujours contenu dans combien de relais ?»*
. *«Un relais contient toujours une bobine»*
. *«Une bobine est toujours contenue dans combien de relais ?»*
. *«un relais est en panne si sa commande est passée et sa sortie est incorrecte»*

Des phrases particulières font intervenir des entités qui, visiblement, existent en un exemplaire unique ; ces cas sont souvent marqués par l'absence d'article. Il est alors nécessaire de généraliser ces phrases, selon deux procédures distinctes :

- si la phrase nomme une entité comme un exemple représentatif, elle est reformulée à l'aide d'une dénomination plus générique.
- si l'entité possède réellement des propriétés qui la différencient de toutes les autres, le cahier des charges doit être revu. En effet SECOOSSE ne peut prendre en compte des propriétés spécifiques à des entités particulières.

Exemple :

«K37 est commandé par le signal qui sort de F13» doit devenir :

. *«Un relais est toujours commandé par un signal»*
. *«Un signal commande toujours combien de relais ?»*
. *«Un signal sort parfois d'un fusible»*
. *«Un fusible génère toujours combien de signaux ?»*

Enfin, après la transformation du cahier des charges en un ensemble de phrases binaires, il est nécessaire d'unifier tous les termes sémantiquement identiques.

Exemple :

les termes alimentation, alimentation électrique, power supply recouvrent les mêmes concepts.

Lorsque la totalité du cahier des charges a été traitée, le document final est soumis aux experts, qui doivent répondre à toutes les questions qui restent posées, puis valider la nouvelle formulation. Ensuite seulement vient l'étape de conceptualisation.

Exemple d'application :

- *le cahier des charges initial est le suivant : «Les relais, que l'on reconnaît par leur nom (par exemple K37, K15, ...) sont commandés, selon qu'il s'agit ou non de relais de puissance, par des signaux ayant des tensions de 12 volts ou 220 volts. Lorsqu'il est dans la position dite "de travail", le relais laisse passer un signal ; par contre lorsqu'il est "au repos" il coupe ce signal, dont la tension n'est cependant pas modifiée.*
- *la mise en forme de ce cahier des charges aboutit, après interrogation des experts sur les questions soulevées, aux phrases binaires suivantes :*
  - *«Un relais a toujours un nom»*
  - *«Un relais de puissance est toujours commandé par un signal de 220V»*
  - *«Un signal 220V commande toujours un nombre quelconque de relais de puissance»*
  - *«Un relais basse puissance est toujours commandé par un signal de 12V»*
  - *«Un signal de 12V commande toujours un nombre quelconque de relais basse puissance»*
  - *«Un relais laisse parfois passer un signal»*
  - *«Un signal passe parfois par un relais»*
  - *«Un relais coupe parfois un signal»*
  - *«Un signal est parfois coupé par un relais»*
  - *«Un signal a toujours une tension»*

C.II.2. Conceptualisation faible

Les phrases binaires obtenues par la démarche précédente sont de la forme *Sujet Verbe Complément*. La conceptualisation de ces phrases a pour but de catégoriser tous les concepts qui apparaissent dans ces phrases, en exploitant les définitions du modèle I.A., afin d'obtenir :

- une liste de NOLOTs
- une liste de LOTS
- une liste de rôles
- une liste de spécifications qui ne peuvent être intégrées au modèle des données et qui seront exploitées lors de l'analyse de la dynamique de l'application.

NOLOTs, LOTs et rôles sont extraits à l'aide de la définition donnée dans le chapitre B : les NOLOTs sont les termes qui recouvrent une connaissance non quantifiable, tandis que les LOTs peuvent être valorisés ; chaque verbe entre deux NOLOTs constitue le rôle du sujet dans une idée-type qui le relie au complément.

Exemple :

- *de la phrase «Un relais contient une bobine», on peut déduire :*
  - *le NOLOT* RELAIS
  - *le NOLOT* BOBINE
  - *le rôle de* RELAIS *est* CONTIENT *dans une idée entre* RELAIS *et* BOBINE.
- *de la phrase «Une bobine est contenue dans un relais» on peut déduire en plus :*
  - *le rôle de la bobine est* EST CONTENUE DANS *dans l'idée-type entre* BOBINE *et* RELAIS.

Conformément à ce que nous avons avancé dans le chapitre précédent, nous considérons désormais que les NOLOTs extraits correspondent à des classes, les LOTs à des attributs-valeur, les rôles à des attributs-relation.
Il est nécessaire d'établir une liste exhaustive des concepts avant d'établir le schéma conceptuel final, à l'aide de l'étape de structuration forte présentée dans la section suivante.

Exemple d'application :

Du cahier des charges pré-traité, exposé précédemment, nous pouvons extraire les NOLOTs :

. RELAIS
. RELAIS DE PUISSANCE
. RELAIS BASSE PUISSANCE
. SIGNAL
. SIGNAL 12V
. SIGNAL 220V

Nous observons également les LOTs :

. NOM *(d'un relais)*
. TENSION *(d'un signal)*

Et enfin les idées-types et leurs rôles

. COMMANDE *et* COMMANDE PAR *entre* SIGNAL 220V *et* RELAIS DE PUISSANCE
. COMMANDE *et* COMMANDE PAR *entre* SIGNAL 12V *et* RELAIS BASSE PUISSANCE
. PASSE PAR *et* LAISSE PASSER *entre* SIGNAL *et* RELAIS
. EST COUPE PAR *et* COUPE *entre* SIGNAL *et* RELAIS

C.III. Structuration forte du schéma conceptuel

[VERMEIR 82] présente une démarche automatisée de structuration forte des schémas conceptuels. Cette démarche est fondée sur la notion de classe homogène :

> Soit C une classe et P_C l'ensemble des propriétés associées à C. La classe C est dite homogène si et seulement si chaque instance de C PEUT POSSEDER une valeur pour chaque propriété de PC.

Exemple

- *la classe des* ENTIERS, *avec les rôles* EST_PLUS_PETIT_QUE *et* EST_PLUS_GRAND_QUE, *est homogène, puisque tout entier est plus petit qu'un autre et plus grand qu'un troisième.*
- *la classe des* ENTIERS_NATURELS, *avec les rôles* EST_PLUS_PETIT_QUE *et* EST_PLUS_GRAND_QUE, *n'est pas homogène, puisque 0 (zéro) ne peut posséder de valeur pour le rôle* EST_PLUS_GRAND_QUE.

Les auteurs définissent ensuite les matrices d'homogénéité :

> Soit C une classe et $P_C = \{P_1,...,P_n\}$ l'ensemble des propriétés associées à C. La matrice d'homogénéité H(C) est une matrice n x n dans laquelle la ligne i et la colonne i sont associées à la propriété P_i. La cellule disposée en colonne i, ligne j, notée $H_{ij}(C)$, contient 'Y' ou 'N' (oui ou non), en fonction de la réponse à la question : «chaque instance de C peut-elle posséder SIMULTANEMENT une valeur pour P_i et pour P_j ?»

Pour vérification, les auteurs signalent et justifient que la matrice d'homogénéité d'une classe doit être égale à sa fermeture transitive.

Exemple : [VERMEIR 82] illustre à l'aide d'exemples les différentes notions introduites.

Ensuite, les auteurs définissent un ordre sur les colonnes des matrices d'homogénéité :

> Soit C une classe, H(C) sa matrice d'homogénéité. Notons $H_i(C)$ la colonne i de la matrice H(C). Alors par définition
> $H_i(C) < H_j(C)$
> si et seulement si
> pour chaque ligne r, si $H_{ir}(C)$ contient 'Y' alors $H_{jr}(C)$ contient également 'Y'.

Cet ordre sur les colonnes, donc sur les propriétés, permet de calculer des classes d'équivalence de propriétés et d'ordonner ces classes d'équivalences. Dès lors, les auteurs fournissent la démarche suivante :

- chaque classe d'équivalence correspond à une nouvelle classe[1], dont le nom reste à trouver ; les propriétés de cette classe sont les propriétés qui appartiennent à la classe d'équivalence.
- chaque relation de supériorité entre deux classes d'équivalence indique une relation d'héritage entre les classes correspondantes, la sous-classe correspondant à la classe d'équivalence la plus «petite».
- Cette démarche fournit la fermeture transitive du graphe d'héritage, il est donc nécessaire de supprimer tous les arcs redondants.

Les auteurs donnent plusieurs exemples, mais leur conclusion est extrêmement intéressante dans le cadre de SECOOSSE. En effet, ils montrent que cette

1 *Admirons au passage l'élégance de la démarche, qui fait correspondre une classe à une classe (d'équivalence) !*

démarche de structuration est capable de transformer une classe unique, possédant la totalité des propriétés d'une application, en une hiérarchie de classes qu'il ne reste plus qu'à nommer. C'est dans ce contexte que nous décidons d'employer la démarche présentée, puisque les étapes précédentes de notre démarche ont abouti, entre autres, à une liste de propriétés indifférenciées. Cependant, nous devons prendre en compte l'un des concepts de notre modèle, qui n'est pas traité dans [VERMEIR 82] : il s'agit de l'évolution des objets. C'est pourquoi nous ne pouvons pas exploiter directement l'algorithme fourni, et nous nous attachons désormais à le transformer pour tenir compte des propriétés qui n'existent pas durant toute la vie d'un objet. De plus, la démarche proposée ne prend en compte que les rôles des objets, donc dans notre contexte aux attributs-relation. Nous en profitons pour l'étendre aux attributs-valeur.

Classe homogène

La définition de la classe homogène présentée dans [VERMEIR 82] peut sans difficulté être appliquée aux attributs-valeur aussi bien qu'aux attributs-relation.

Matrice d'homogénéité

Par contre, la prise en compte de l'évolution des objets nous amène à donner une nouvelle définition de la matrice d'homogénéité :

> Soit C une classe et $P_C = \{P_1,...,P_n\}$ l'ensemble des propriétés associées à C. La matrice d'homogénéité H(C) est une matrice n x n dans laquelle la ligne i et la colonne i sont associées à la propriété P_i. La cellule disposée en colonne i, ligne j, notée $H_{ij}(C)$, contient 'TJR', 'Tps' ou 'jam' (*toujours*, *de temps en temps* ou *jamais*), en fonction de la réponse à la question : «Quand une instance de C ***peut-elle*** posséder SIMULTANEMENT une valeur pour P_i et pour P_j ?»

Notons que cette question est uniquement destinée à mettre en évidence *l'incompatibilité conceptuelle* entre deux propriétés. Il n'est pas question, ici, de prendre en compte des indications sur le fait qu'une propriété doit être toujours ou de temps en temps valorisée : cette caractéristique de cardinalité minimale (ainsi d'ailleurs que la cardinalité maximale) de chaque propriété est définie durant la mise en forme du cahier des charges.

Exemple de matrice d'homogénéité :

Construisons la matrice d'homogénéité pour l'exemple d'application traité dans les sections précédentes. Nous ne considérons plus les NOLOTs extraits dans l'étape précédente, mais partons de l'hypothèse qu'une classe C, unique, porte tous les LOTs et tous les rôles. Notre but est donc de construire une hiérarchie de spécialisation et d'alternatives, dans laquelle, espérons-le, nous retrouverons les NOLOTs précédemment cités. La matrice d'homogénéité obtenue est présentée figure 28.

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1) LAISSE PASSER	TJR	jam	jam	jam	jam	jam	TJR	jam	TJR	TJR
2) PASSE PAR	jam	TJR	jam	jam	TJR	TJR	jam	TJR	jam	jam
3) COUPE	jam	jam	TJR	jam	jam	jam	TJR	jam	TJR	TJR
4) EST COUPE PAR	jam	jam	jam	TJR	TJR	TJR	jam	TJR	jam	jam
5) TENSION	jam	Tps	jam	Tps	TJR	TJR	jam	TJR	jam	jam
6) COMMANDE(12V)	jam	Tps	jam	Tps	TJR	TJR	jam	jam	jam	jam
7) COMMANDE PAR(12V)	Tps	jam	Tps	jam	jam	jam	TJR	jam	jam	TJR
8) COMMANDE(220V)	jam	Tps	jam	Tps	TJR	jam	jam	TJR	jam	jam
9) COMMANDE PAR(220V)	Tps	jam	Tps	jam	jam	jam	jam	jam	TJR	TJR
10) NOM	Tps	jam	Tps	jam	jam	jam	TJR	jam	TJR	TJR

Figure 28 : Une matrice d'homogénéité

Voici les commentaires sur la façon dont cette matrice d'homogénéité a été construite :

- *Il est évident que chaque $H_{ii}(C)$ contient TJR.*
- *$H_{7,1}(C)$ et $H_{9,1}(C)$ contiennent TJR car un (relais) qui laisse passer (un signal)* PEUT TOUJOURS *être commandé par un signal, de 12V ou de 220V.*
- *$H_{10,1}(C)$ contient TJR car un (relais) qui laisse passer (un signal) peut toujours posséder un nom.*
- *$H_{5,2}(C)$ contient TJR car un (signal) qui passe par (un relais) peut toujours posséder une tension.*

- *$H_{6,2}(C)$ et $H_{8,2}(C)$ contiennent TJR car un (signal) qui passe par (un relais) peut toujours commander, en 12V comme un 220V, (un relais).*
- *$H_{7,3}(C)$ et $H_{9,3}(C)$ contiennent TJR car un (relais) qui coupe (un signal) peut toujours être commandé, en 12V ou en 220V.*
- *$H_{10,3}(C)$ contient TJR car un (relais) qui coupe (un signal) peut toujours posséder un nom.*
- *$H_{5,4}(C)$ contient TJR car un (signal) qui est coupé par (un relais) peut toujours posséder une tension.*
- *$H_{6,4}(C)$ et $H_{8,4}(C)$ contiennent TJR car un (signal) qui est coupé par (un relais) peut toujours commander, en 12V comme un 220V, (un relais).*
- *$H_{2,5}(C)$ et $H_{4,5}(C)$ contient Tps car un (signal) qui a une tension passe par (un relais) de temps en temps et est coupé par (un relais) de temps en temps. Par contre, un (signal) qui a une tension peut toujours commander (un relais), en 12V comme en 220V ($H_{6,5}(C)$ et $H_{8,5}(C)$).*
- *de la même façon, $H_{2,6}(C)$ et $H_{4,6}(C)$, comme $H_{2,8}(C)$ et $H_{4,8}(C)$ contiennent Tps car (un signal) qui commande (un relais), en 12V ou en 220V, passe par (un relais) de temps en temps et est coupé par (un relais) de temps en temps. De plus ($H_{5,6}(C)$ et $H_{5,8}(C)$) (un signal) qui commande (un relais) en 12V ou en 220V a toujours une tension.*
- *$H_{1,7}(C)$, $H_{3,7}(C)$, $H_{1,9}(C)$ et $H_{3,9}(C)$ contiennent Tps car (un relais), qu'il soit commandé en 12V ou en 220V, peut de temps en temps couper et de temps en temps laisser passer (un signal). De plus ($H_{10,7}(C)$ et $H_{10,9}(C)$) un (relais), commandé en 12 ou en 220V, peut toujours avoir un nom.*
- *Enfin ($H_{1,10}(C)$, $H_{7,10}(C)$ et $H_{9,10}(C)$), un (relais) qui a un nom peut de temps en temps couper et de temps en temps laisser passer un signal, tandis qu'il peut toujours être commandé en 12V ou en 220V.*

Vérification de la cohérence

Pour vérifier la cohérence de la matrice d'homogénéité, nous nous proposons de comparer, pour i et j entre 1 et n, les cellules $H_{ij}(C)$ et $H_{ji}(C)$. Il est évident que si $H_{ij}(C)$ contient JAM, alors $H_{ji}(C)$ doit également contenir JAM. Les autres cas autorisés sont décrits dans le tableau suivant :

$H_{ij}(C) \setminus H_{ji}(C)$	**TPS**	**TJR**
TPS	①	②
TJR	③	④

La cellule ① spécifie des entités qui peuvent avoir une valeur soit pour la propriété P_i seule, soit pour la propriété P_j seule, soit pour les propriétés P_i et P_j simultanément.
La cellule ② spécifie des entités qui peuvent avoir à tout moment une valeur pour la propriété P_j et qui possèdent de temps en temps une valeur pour la propriété P_i.
La cellule ③ spécifie des entités qui peuvent avoir à tout moment une valeur pour la propriété P_i et qui possèdent de temps en temps une valeur pour la propriété P_j.
La cellule ④ spécifie des entités qui possèdent simultanément, indépendamment du temps, une valeur pour P_i et une valeur pour P_j.
Ces quatre cas sont les seuls autorisés dans une matrice cohérente.

De plus, comme les auteurs de [VERMEIR 82], nous constatons que, de façon évidente, la matrice des TJR doit être transitive. Nous devons cependant étudier le problème entraîné par les TPS. Voici le tableau qui présente les valeurs possibles par transitivité : en ligne, nous avons placé les valeurs possibles pour $H_{ij}(C)$; en colonne, nous trouvons les valeurs possibles pour $H_{jk}(C)$; les cases contiennent les seules valeurs admises pour $H_{ik}(C)$, lors de la vérification de la transitivité.

$H_{ij}(C) \setminus H_{ji}(C)$	**TPS**	**TJR**
TPS	TPS,TJR	TPS,TJR
TJR	TPS,TJR	TJR

Exemple

Nous ne donnons pas le détail de la vérification, mais la matrice d'homogénéité précédente vérifie toutes ces contraintes.

Ordre sur les colonnes

Nous définissons ensuite non plus un, mais deux ordres sur les colonnes de la matrice d'homogénéité :

- L'ordre $<_g$ correspond à l'ordre présenté précédemment : $H_i(C) <_g H_j(C)$ signifie que toute entité ayant une valeur pour la propriété P_i peut toujours posséder une valeur pour la propriété P_j.

$$H_i(C) <_g H_j(C) \quad \Leftrightarrow \quad (\forall k)\ (H_{ik}(C) = TJR) \Rightarrow (H_{jk}(C) = TJR)$$

- L'ordre $<_a$ met en évidence les dépendances temporelles entre propriétés : $H_i(C) <_a H_j(C)$ signifie que $H_i(C) <_g H_j(C)$, mais également que toute entité pouvant posséder une valeur pour la propriété P_j peut également posséder une valeur pour P_i durant une partie de sa vie, mais ne possède aucune valeur pour P_i durant le reste de sa vie.

$$H_i(C) <_a H_j(C) \quad \Leftrightarrow \quad (C_i <_g C_j) \text{ et } (\forall k)\ (H_{ik}(C) = Tps) \Rightarrow (H_{jk} = TJR)$$

Nous définissons ensuite l'égalité des colonnes, $H_i(X) \equiv H_j(C)$ signifiant que toute entité qui peut posséder une valeur pour P_i peut également en posséder une pour P_j.

$$H_i(C) \equiv H_j(C) \quad \Leftrightarrow \quad (H_i(C) <_g H_j(C)) \text{ et } (H_j(C) <_g H_i(C))$$

Exemple

En appliquant ces définitions à la matrice précédente, nous pouvons constater par exemple que $H_1(C) <_a H_{10}(C)$. *En effet :*

$H_{1,1}(C) = TJR$ *et* $H_{10,1}(C) = TJR$, *donc* $H_1(C) <_g H_{10}(C)$;

de plus

$H_{1,7}(C) = Tps$ *et* $H_{10,7}(C) = TJR$,

$H_{1,9}(C) = Tps$ *et* $H_{10,9}(C) = TJR$,

$H_{1,10}(C) = Tps$ *et* $H_{10,10}(C) = TJR$,

CQFD.

Nous pouvons également constater que $H_7(C) <_g H_{10}(C)$. *En effet :*

$H_{7,1}(C) = TJR$ *et* $H_{10,1}(C) = TJR$,
$H_{7,3}(C) = TJR$ *et* $H_{10,3}(C) = TJR$,
$H_{7,7}(C) = TJR$ *et* $H_{10,7}(C) = TJR$,
$H_{7,10}(C) = TJR$ *et* $H_{10,10}(C) = TJR$,
CQFD.

Par contre il n'existe pas dans cette matrice de colonnes égales.

Démarche

La démarche de SECOOSSE est alors identique à celle proposée dans [VERMEIR 82] :

- Etablir les classes d'équivalence ξ_i des propriétés :

ξ_i *est définie par :* $(\forall P_i \in \xi_i)\ (\forall P_j \in \xi_i)\ (H_i(C) \equiv H_j(C))$

Exemple :

Les classes d'équivalence obtenues dans la matrice d'homogénéité précédente contiennent chacune une seule colonne, puisqu'il n'y a pas d'égalités entre colonnes dans cette matrice. Nous nommons donc ces classes ξ_1 *à* ξ_{10}.

- Etablir l'ordre entre les classes d'équivalence :

$$\xi_i \langle_g \xi_j \quad \Leftrightarrow \quad (\exists P_i \in \xi_i)\ (\exists P_j \in \xi_j)\ (H_i(C) <_g H_j(C))$$

$$\xi_i \langle_a \xi_j \quad \Leftrightarrow \quad (\exists P_i \in \xi_i)\ (\exists P_j \in \xi_j)\ (H_i(C) <_a H_j(C))$$

Exemple :

L'ordre sur les classes d'équivalence ξ_1 *à* ξ_{10} *est le suivant :*

$\xi_1 <_a \xi_{10}$, $\xi_2 <_a \xi_5$, $\xi_3 <_a \xi_{10}$, $\xi_4 <_a \xi_5$
$\xi_6 <_g \xi_5$, $\xi_7 <_g \xi_{10}$, $\xi_8 <_g \xi_5$, $\xi_9 <_g \xi_{10}$

- Le concept du modèle orienté objet correspondant à la classe d'équivalence ξ_i est une classe que nous noterons C_i. Les classes sont liées entre elles par des arcs de la façon suivante :

| *si* $\xi_i \langle_g \xi_j$, *un arc g lie la classe* C_i *à la classe* C_j.

Dans ce cas, la classe C_j est spécialisée par la classe C_i.

| *si* $\xi_i \langle_a \xi_j$, *un arc a lie la classe* C_i *à la classe* C_j.

Dans ce cas, la classe C_j a C_i pour alternative.

- Les relations d'inégalités entre les classes d'équivalence (les arcs entre les classes) ne sont pas optimisées ; au contraire toutes les transitivités sont exprimées :

| $(\xi_i \langle_g \xi_j)$ *et* $(\xi_j \langle_g \xi_k)$ *et* $(\xi_i \langle_g \xi_k)$

| $(\xi_i \langle_a \xi_j)$ *et* $(\xi_j \langle_a \xi_k)$ *et* $(\xi_i \langle_a \xi_k)$

Après élimination de ces arcs transitifs et utilisation des NOLOTs exprimés dans la section précédente pour nommer les classes obtenues, le schéma conceptuel fortement structuré correspondant aux spécifications initiales peut être dessiné.

Exemple :

- La classe C_1 correspondant à ξ_1 possède la propriété LAISSE_PASSER. Nous la nommons RELAIS_TRAVAIL.
- La classe C_2 correspondant à ξ_2 possède la propriété PASSE_PAR. Nous la nommons SIGNAL_NON_COUPE.
- La classe C_3 correspondant à ξ_3 possède la propriété COUPE. Nous la nommons RELAIS_REPOS.
- La classe C_4 correspondant à ξ_4 possède la propriété COUPE_PAR. Nous la nommons SIGNAL_COUPE.
- La classe C_5 correspondant à ξ_5 possède la propriété TENSION. Nous la nommons SIGNAL.
- La classe C_6 correspondant à ξ_6 possède la propriété COMMANDE(12V). Nous la nommons SIGNAL_12V.

- La classe C_7 correspondant à ξ_7 possède la propriété COMMANDE_PAR(12V). Nous la nommons RELAIS_BASSE_PUISSANCE.
- La classe C_8 correspondant à ξ_8 possède la propriété COMMANDE(220V). Nous la nommons SIGNAL_220V.
- La classe C_9 correspondant à ξ_9 possède la propriété COMMANDE_PAR(220V). Nous la nommons RELAIS_PUISSANCE.
- La classe C_{10} correspondant à ξ_{10} possède la propriété NOM. Nous la nommons RELAIS.

- les relations d'ordre établies entre les classes d'équivalence amènent au schéma conceptuel suivant :

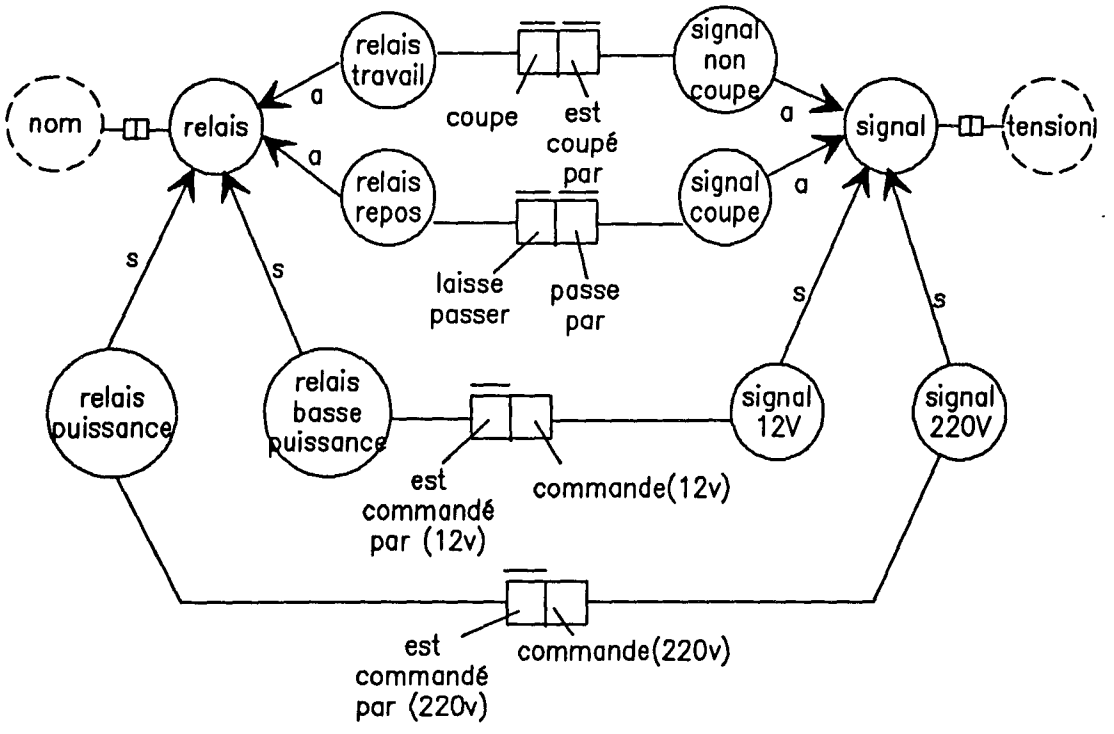


Figure 29 : Schéma conceptuel obtenu à l'aide de la démarche

Rappelons que les cardinalités minimum et maximum, portées sur les rôles, ne proviennent pas de la procédure proposée, mais de la mise en forme du cahier des charges.

C.IV. Conception détaillée de l'application

L'étape de conception détaillée permet d'exploiter les différents schémas conceptuels obtenus, dans les différents domaines de connaissance, pour créer un schéma conceptuel final, complet, fondé sur la modélisation de la réalité qui constitue l'un de ces domaines de connaissance. L'étape conceptuelle est alors achevée par la conception de la dynamique des traitements dont la démarche, nous le verrons, peut être rapprochée de celle qui est exploitée dans l'analyse structurée.

C.IV.1. Unification des différents domaines de connaissance

Les différents schémas conceptuels ont été conçus indépendamment les uns des autres, mais ils sont tous fondés sur la représentation structurelle de l'équipement à maintenir. Chaque concepteur doit donc établir les liens de spécialisation multiple entre ses classes et celles qui existent dans la représentation de l'équipement. Nous ne pouvons proposer de règles pour la réalisation de cette tâche. En effet, selon le domaine de connaissance qu'il traite, chaque concepteur obtient une hiérarchie arbitraire de classes. La seule constante réside dans le fait que tous les concepteurs ont oeuvré à partir des mêmes informations, c'est-à-dire à partir de la description de la structure de l'équipement à maintenir. Par conséquent, un concepteur est libre d'associer à une classe de la structure toute classe de son choix.
Aussi, plutôt que de proposer un exemple «jouet», nous suggérons une application réelle dans la partie III de ce rapport.

Lorsque chaque intervenant a réalisé cette tâche, le schéma conceptuel des données global obtenu est achevé et permet désormais à chacun d'établir les traitements nécessaire à l'activité de l'application.

C.IV.2. Conception des traitements et de la dynamique

Nous l'avons vu dans le paragraphe A.VI (page 103), le modèle de la dynamique de SECOOSSE assimile le couple <objet,opération> au couple <donnée,procédure> de la programmation structurée. Ce modèle minimal pour la dynamique d'un système orienté objets présente l'avantage de déléguer l'analyse de la dynamique à un domaine dans lequel les méthodes d'analyse sont aussi nombreuses que variées. Dès lors, nous proposons à chaque concepteur de sélectionner la méthode d'analyse structurée de son choix, correspondant au

mieux à ses aspirations et de l'appliquer à l'analyse de la dynamique de l'application en cours de création (Cf. par exemple [MEYER 84], [DUCRIN 84]).

Nous proposons alors quelques règles qui permettent d'unifier l'univers de la programmation structurée avec la dynamique des systèmes orientés objets.
Tout d'abord, un objet peut être considéré comme une donnée composite (correspondant par exemple au RECORD de Pascal ou au STRUCT de C) ; chaque attribut de l'objet correspond alors à un champ de la structure.
L'analyse structurée des traitements doit alors être réalisée avec une contrainte : toute procédure doit posséder au minimum un argument. Nous suggérons de nommer systématiquement ***Self*** cet argument dans la liste des paramètres de la procédure (il s'agit d'une dénomination habituelle, dans les systèmes orientés objets, pour marquer l'objet sur lequel porte la procédure en cours). Ce paramètre, Self, est toujours le premier paramètre dans le profil de la procédure.
Dès lors, nous proposons d'unifier une procédure P définie par :

P : $C_0 \times C_1 \times \ldots \times C_n \rightarrow C_r$
<Self,Argument$_1$,...,Argument$_n$> $\vdash\!\rightarrow$ Résultat

avec une opération P, attachée à la classe C_0, dont le profil est :

P : $C_1 \times \ldots \times C_n \rightarrow C_r$
<Argument$_1$,...,Argument$_n$> $\vdash\!\rightarrow$ Résultat

Certaines méthodes d'analyse structurée descendante, telles MEDEE [DUCRIN 84] permettent de différencier immédiatement les opérations publiques (celles à partir desquelles un morceau d'analyse débute) des opérations privées (celles qui sont créées par commodité). Dans le cas de certaines méthodes, cette différenciation doit faire appel au bon sens du concepteur.

C.V. Conclusion

Nous proposons dans ce chapitre une démarche complète, qui permet de transformer un cahier des charges en un schéma structurel complètement structuré. L'avantage de cette démarche réside principalement dans le fait que seuls les concepts maîtrisés par les experts, correspondant aux NOLOTs et aux LOTs, sont nécessaires ; les notions abstraites et propres au modèle de SECOOSSE, entendons l'héritage de spécialisation et les alternatives, ne sont jamais connus des experts : ces concepts sont générés automatiquement.

Des études complémentaires pourraient être appliquées à différentes étapes de la démarche proposée. Par exemple, des études sur le langage naturel, couplées à une application de règles systématiques de réécriture, permettraient de proposer une procédure automatisable pour la mise en forme du cahier des charges. Nous n'avons pas réalisé de telles études afin de limiter le cadre de notre travail aux méthodes de conception orientées objets.

Chapitre D
Validation : SECOOSSE par SECOOSSE

Nous avons présenté une définition succincte des modèles réflexifs dans le chapitre C de la première partie. Nous avons en particulier indiqué que la réflexivité d'un modèle garantit son homogénéité et favorise son extensibilité.

Le modèle présenté pour SECOOSSE n'est pas réflexif, en particulier parce que les concepts objets présentés sont exprimés non en fonction d'eux-mêmes, mais à l'aide d'un langage prédicatif. Nous nous proposons cependant d'exprimer le modèle de la méthode à l'aide de la méthode elle-même, aboutissant alors à une expression des concepts à l'aide de ces mêmes concepts. Cette expression exploitera également le langage et la démarche de la méthode. Cette (re)définition circulaire du modèle a deux buts :

- d'une part, le modèle est défini à l'aide de lui-même, donc est réflexif, avec les avantages sus-mentionnés
- d'autre part, cette première utilisation de la méthode de conception permet de valider le quadruplet qui compose la méthode : modèle, langage, démarche, outil.

Nous considérons que cette validation concerne également l'outil associé à la méthode. En effet, les schémas conceptuels présentés dans ce chapitre ont été réalisés avec l'assistance d'un outil informatique, nommé SEISME, qui est présenté dans le chapitre suivant. Nous observons ici un effet de bord de la méta-circularité désirée pour la méthode. En effet, l'outil doit naturellement prendre en compte, dans sa réalisation, les caractéristiques dues à la réflexivité du modèle : donc cette réflexivité doit avoir été décrite auparavant. Mais les schémas conceptuels permettant l'expression de la réflexivité sont réalisés à l'aide de l'outil, qui lui-même devrait être décrit dans un chapitre antérieur... Nous choisissons donc de présenter le modèle sous une forme réflexive dans ce chapitre, en utilisant des schémas conceptuels générés à l'aide de l'outil qui est décrit dans le chapitre suivant.

Nous exprimons maintenant les concepts du modèle de SECOOSSE selon trois thèmes : le paragraphe D.I présente les méta-classes, ainsi que la méta-description des relations d'héritage qui existent entre classes ; le paragraphe D.II présente la définition des classes de propriétés ; enfin le paragraphe D.III montre comment interviennent les concepts de l'utilisateur par rapport à ces méta-schémas.

D.I. Méta-classes et héritage

La figure 30 montre le schéma conceptuel des méta-classes de SECOOSSE. La hiérarchie correspondante, relativement simple, reflète fidèlement le modèle exprimé en langage prédicatif.

Le prédicat ***classe***(C) est formalisé à l'aide de la méta-classe la plus générale, CLASSE. Nous associons à chaque classe un nom, formalisé à l'aide de l'attribut NOM_CLASSE.

Les faits ***classe_factorisation***(C), ***classe_alternative***(C), ***classe_sémantique***(C) sont formalisés respectivement par les méta-classes CLASSE_FACTORISATION, CLASSE_ALTERNATIVE et CLASSE_SEM.

Les trois clauses qui définissent ***classe***(C), indiquées à l'équation 17, sont formalisées par les relations de spécialisation entre CLASSE et CLASSE_SEM, CLASSE_ALTERNATIVE, CLASSE_FACTORISATION.

Le modèle exprime les différentes relations d'héritage entre classes à l'aide des faits ***factorise***(C,C'), ***multi_spécialise***(C,C') et ***spécialise***(C,C'). Etant des faits, ces prédicats ne portent aucune contrainte supplémentaire. Au contraire, le modèle réflexif permet d'exprimer ces contraintes :

- le fait ***factorise***(C,C') est formalisé par un attribut de relation entre CLASSE_FACTORISATION et CLASSE. De la sorte, seule une classe de factorisation a la possibilité de factoriser ; par contre toute classe peut faire l'objet d'une factorisation.
- le fait ***multi_spécialise***(C,C') est formalisé par les attributs MSPECIALISE et MSPECIALISE_PAR entre CLASSE_SEM et elle-même. La contrainte exprimée est que seules des classes sémantiques peuvent être utilisées dans une relation de spécialisation multiple. Nous observons qu'il n'existe aucune contrainte sur la cardinalité des attributs : en effet, une classe n'est pas toujours aspect

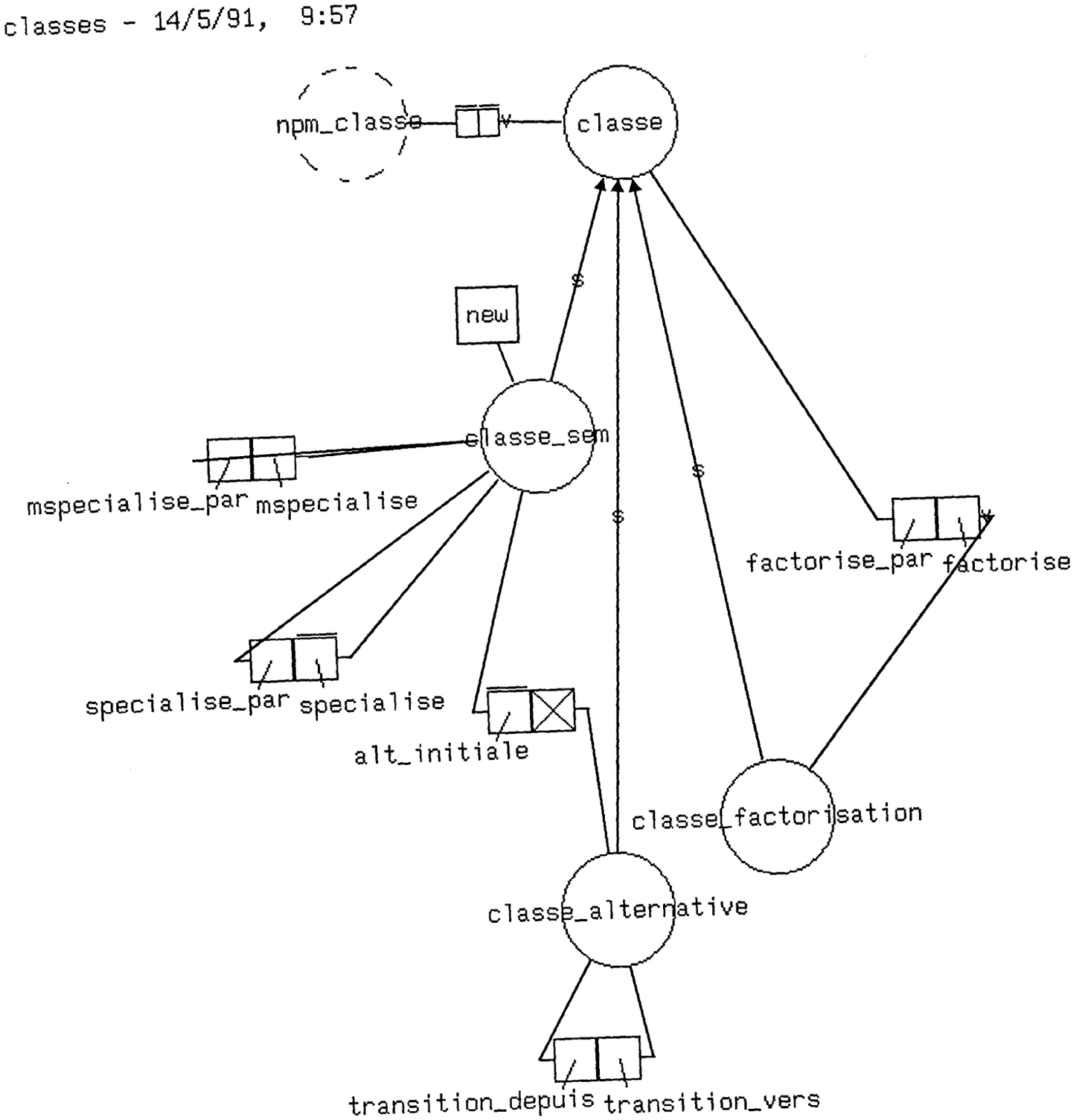
d'une autre classe et, réciproquement, une classe n'est pas toujours observée selon différents points de vue.

- de la même façon, le fait ***spécialise***(C,C') est formalisé par les attributs SPECIALISE et SPECIALISE_PAR entre CLASSE_SEM et elle-même. L'attribut spécialise possède une cardinalité maximale de 1 car une classe spécialise au maximum une super-classe.

Dans le modèle, toute classe peut se voir associer un ensemble de classes alternatives à l'aide des faits ***alt_initiale***(CS,CA) et ***transition_possible***(CA,CA') :

- le fait ***alt_initiale***(CS,CA) est formalisé par l'attribut de relation ALT_INITIALE entre CLASSE_SEM et CLASSE_ALTERNATIVE. Une classe sémantique possédant au maximum une classe alternative initiale, cet attribut de relation possède une cardinalité maximale de 1.
- le fait ***transition_possible***(CA,CA') est formalisé par les attributs TRANSITION_DEPUIS et TRANSITION_VERS, entre CLASSE_ALTERNATIVE et elle-même. Le nombre de transitions partant et arrivant à une classe alternative n'est pas contraint.

Figure 30 : Définition des classes



D.II. Classes de propriétés

La figure 31 montre la formalisation des propriétés telles qu'elles sont définies dans le modèle de SECOOSSE.

L'ensemble des propriétés est modélisé à l'aide de la classe PROPRIETE. Le fait ***nom_propriété***(P,N), qui associe l'identificateur N à la propriété P, est formalisé à l'aide de l'attribut NOM_PROPRIETE, dont la cardinalité est obligatoirement 1.

Le prédicat ***propriété***(C,P), tel que défini dans l'équation 19 (page 101), montre l'existence de la propriété P sur la classe C ; il est formalisé à l'aide des attributs de relation POSSEDE et ATTACHE_A, qui définissent qu'une classe peut posséder un nombre quelconque de propriétés mais qu'une propriété est attachée à une et une seule classe.

Le même prédicat ***propriété***(C,P) définit les propriétés comme appartenant à quatre catégories, les attributs, les opérations, les propriétés publiées et les propriétés héritées :

- le prédicat ***attribut***(C,A) est formalisé par la classe ATTRIBUT. La première clause de l'équation 19 entraîne la relation de spécialisation entre cette classe et la classe PROPRIETE.

  De plus :
  - . les équations 1 et 2 sont formalisées par les classes ATTRIBUT_VALEUR et ATTRIBUT_RELATION.
  - . le fait ***domaine***(A,C) est représenté par l'attribut DOMAINE dont la cardinalité est 1.
  - . le fait ***dual***(A,A') est représenté par l'attribut DUAL dont la cardinalité est limitée à 1.
  - . les faits ***cardinalité***(A,N1,N2) et ***valeur_par_défaut***(A,V), communs aux attributs simples et aux attributs de relation, sont formalisés par les attributs VALEUR_PAR_DEFAUT, CARD_MIN et CARD_MAX de la classe ATTRIBUT.
  - . Enfin, l'équation 3 entraîne les relations de spécialisations entre ATTRIBUT et ATTRIBUT_VALEUR ainsi que entre ATTRIBUT et ATTRIBUT_RELATION.
- le prédicat ***opération***(C,O) est formalisé par la classe OPERATION. La seconde clause de l'équation 19 entraîne la relation de spécialisation entre cette classe et la classe PROPRIETE. Nous ajoutons l'attribut TEXTE_OPERATION

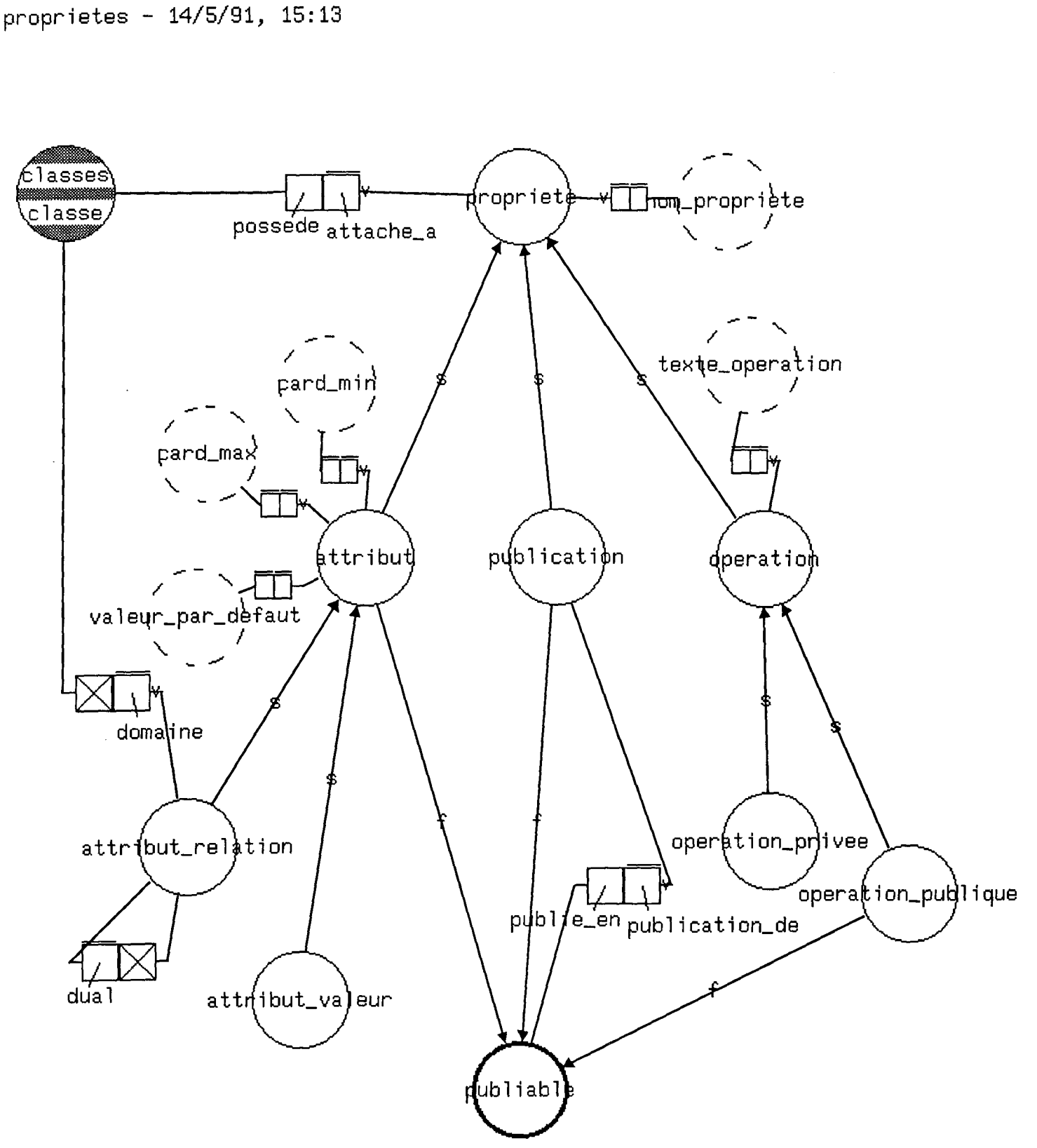
à cette classe, par rapport au modèle, afin de mémoriser qu'une opération possède une description détaillée. De plus :

. les faits ***opération_publique***(C,O) et ***opération_privée***(C,O) sont formalisés par les classes correspondantes, OPERATION_PUBLIQUE et OPERATION_PRIVEE.

. l'équation 8 entraîne les relations de spécialisation entre ces classes et la classe OPERATION.

- le fait ***propriété_publiée***(C,P) fait intervenir plusieurs concepts. D'une part, nous établissons une classe des propriétés publiées, nommée PUBLICATION. Une publication, outre qu'il s'agit d'une propriété, correspond à une propriété publiable : attribut, opération publique ou une autre publication. Cette relation entre propriété publiée et propriété publication est formalisée à l'aide des attributs PUBLIE_EN et PUBLICATION_DE.
  Dans l'absolu, l'attribut PUBLIE_EN devrait exister sur trois classes : ATTRIBUT, OPERATION_PUBLIQUE et PUBLICATION. Nous avons factorisé ces classes avec PUBLIABLE, sur laquelle est défini PUBLIE_EN.
- le prédicat ***propriété_héritée***(C,P) n'est pas formalisé dans ce schéma conceptuel. En effet, nous considérons l'héritage d'une propriété comme un *mécanisme* du modèle et non comme une caractéristique sémantique. Ce mécanisme n'est donc pas représenté parmi les concepts du modèle ; il doit au contraire être défini dans l'analyse de la dynamique du modèle réflexif, en utilisant les attributs MSPECIALISE et SPECIALISE sur CLASSE_SEM.

Figure 31 : Définition des propriétés



D.III. Objets

La figure 32 présente la relation entre le méta-schéma et les schémas établis par le concepteur. Cette mise en correspondance est établie à l'aide de la notion d'objet : le fait ***objet***(C,O) est formalisé par la classe OBJET.

Par définition de la réflexivité (Cf. section C.II.4 de la partie I, page 31), chaque concept du modèle doit être exprimé à l'aide du modèle. On considère alors que chaque classe et chaque propriété est un objet, matérialisé par une clause du prédicat ***objet***. Par conséquent, la «classe des classes» (CLASSE) ainsi que la «classe des propriétés» (PROPRIETE) définies dans les paragraphes précédents sont des sous-classes de OBJET : de la sorte, chacune de leurs instances -c'est-à-dire chaque classe et chaque propriété- est instance de OBJET.
Chacun des concepts du modèle a été formalisé dans les schémas précédents à l'aide de classes (par exemple, CLASSE, CLASSE_SEM, CLASSE_ALTERNATIVE, ...). Toutes ces classes, étant elles-mêmes des classes sémantiques, sont instances de CLASSE_SEM[1] : la méta-circularité du modèle de SECOOSSE se situe au niveau de CLASSE_SEM, qui est instance d'elle-même.

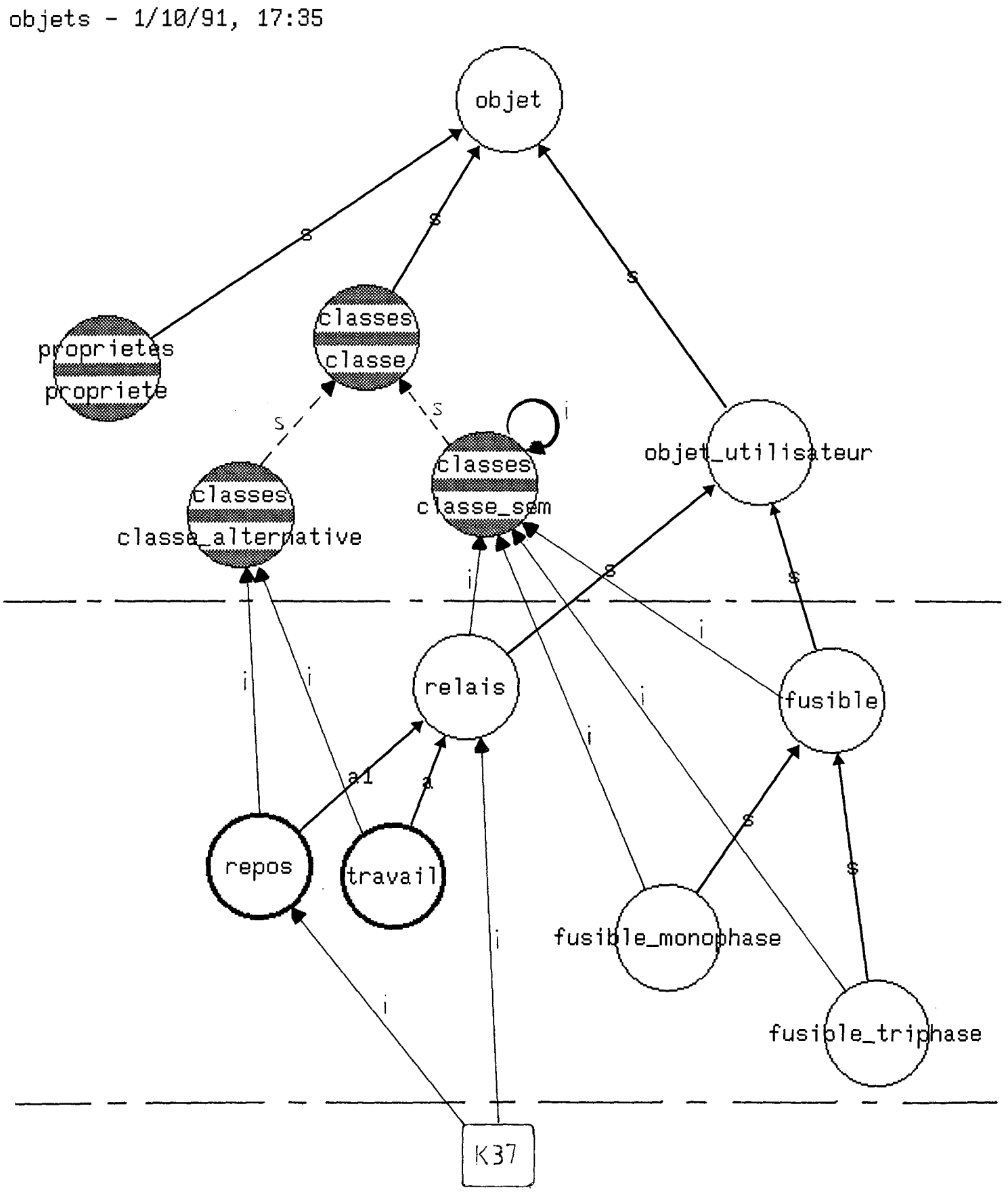
La classe OBJET_UTILISATEUR est destinée, quant à elle, à être instanciée, directement ou au travers de relations de spécialisation, par les objets que manipule l'utilisateur ; par conséquent, toute classe du concepteur doit être reliée, directement ou indirectement, à OBJET_UTILISATEUR.
Ainsi, les classes FUSIBLE et RELAIS, qui font partie des concepts de l'application et non du méta-schéma, sont directement en relation de spécialisation avec OBJET_UTILISATEUR ; par contre les classes FUSIBLE_MONOPHASE et FUSIBLE_TRIPHASE, spécialisant FUSIBLE, sont en relation indirecte avec OBJET_UTILISATEUR. Lors de la création d'une classe par le concepteur, donc lors de l'instanciation de CLASSE_SEM, cette classe est définie par défaut comme spécialisant OBJET_UTILISATEUR : la valeur de l'attribut VALEUR_PAR_DEFAUT associée à l'objet qui représente l'attribut SPECIALISE contient la valeur OBJET_UTILISATEUR. Si SECOOSSE était un langage, un mécanisme devrait s'assurer, lors de chaque modification des relations de spécialisation, que cet attribut SPECIALISE contient bien cette valeur par défaut.
Notons que FUSIBLE, RELAIS, FUSIBLE_MONOPHASE, FUSIBLE_TRIPHASE sont des classes sémantiques, donc qu'elles sont instances de CLASSE_SEM ; par contre, REPOS et TRAVAIL sont instances de CLASSE_ALTERNATIVE.

---

1 *L'arc i présenté figure 32, page 153, n'appartient pas au modèle de SECOOSSE. Il s'agit seulement d'une représentation de l'instanciation.*

Enfin, les objets créés par l'utilisateur dans ses classes sont pour leur part instances des classes de l'utilisateur : ainsi, K37 est instance de RELAIS et, parce qu'il existe des alternatives d'évolution pour cette classe, K37 est également (en cet instant) instance de REPOS.

Figure 32 : Définition des objets



D.IV. Conclusion

Cette expression sous une forme réflexive du modèle de SECOOSSE, outre le fait qu'elle permet de valider les concepts de la méthode, nous amène à envisager la création d'un outil informatique d'assistance à la conception fondé sur ces spécifications méta-circulaires. Le chapitre suivant propose l'analyse conceptuelle de SEISME, qui est un outil dédié au modèle de SECOOSSE.

Chapitre E
Les outils de SECOOSSE

La méthode de conception SECOOSSE a été conçue, rappelons-le, dans un contexte industriel. Dans cette situation, il s'est avéré indispensable que les concepts proposés puissent être mis en oeuvre rapidement et simplement par des intervenants qui ne sont pas nécessairement des spécialistes du modèle orienté objets. C'est pourquoi nous proposons un outil informatique d'assistance à la conception accompagnant le modèle et la démarche présentés. Cet outil doit par conséquent respecter les contraintes suivantes :

- tous les concepts proposés dans le modèle de SECOOSSE doivent être inclus dans le logiciel d'aide à la conception.
- un guide méthodologique doit permettre à l'utilisateur de respecter la démarche tout en le laissant libre de prendre lui-même des décisions.
- l'interface de l'outil doit être de forme graphique. En effet, comme nous l'avons présenté dans le chapitre consacré au langage de SECOOSSE, la représentation textuelle de concepts selon le modèle orienté objets fournit des spécifications peu lisibles et contraires à l'intuition.

C'est pourquoi nous proposons ici les schémas conceptuels d'un outil informatique nommé SEISME[1]. Cet outil a naturellement été conçu en fonction de l'expression méta-circulaire du modèle de SECOOSSE : par conséquent l'univers réel qui est modélisé dans SEISME est en fait composé du modèle de SECOOSSE et de l'interface avec l'utilisateur. Nous appliquons à la formalisation de SEISME la séparation des domaines de connaissance, ce qui nous amène à considérer :

- les concepts présentés dans le méta-schéma de SECOOSSE comme la réalité à modéliser (paragraphe E.I).
- l'interface graphique, utilisant la souris et un environnement graphique multi-fenêtré, comme un aspect de cette réalité (paragraphe E.II).
- une interface intermittente permettant l'acquisition et la gestion textuelles des informations telles que les noms, les cardinalités (...) constitue un

1 *Il ne s'agit pas d'un acronyme.*

second aspect selon lequel peut être considéré chaque concept. Elle est présentée dans le paragraphe E.III.

Les deux aspects d'interface sont connectés au modèle réel à l'aide de relations de spécialisation multiple.

E.I. Adaptation du modèle de SECOOSSE

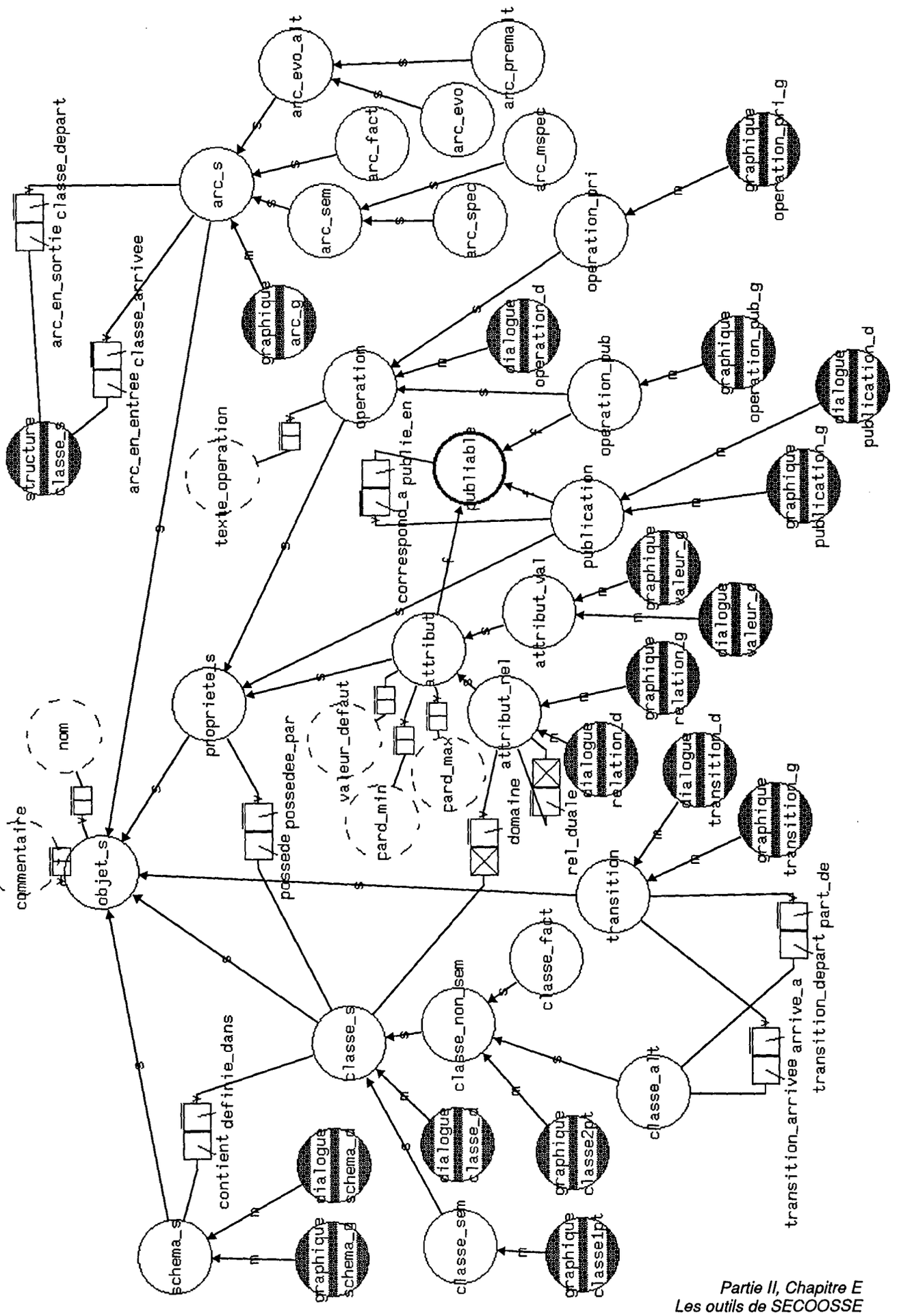
La figure 33 montre la modélisation de l'univers de l'outil.

Une classe très générale décrit les objets du modèle sous une forme générique : la classe OBJET_S indique que tout concept du modèle possède un NOM et qu'un COMMENTAIRE peut lui être associé.
La classe SCHEMA_S, à gauche, est la classe des schémas conceptuels. La conception d'une application importante, comme le sont les systèmes experts de seconde génération, peut en effet difficilement être réalisée sur un schéma unique. Cette classe des schémas est considérée selon deux aspects : d'une part, sa représentation graphique est formalisée dans l'aspect GRAPHIQUE ; d'autre part le dialogue qui renseigne les caractéristiques d'un schéma est formalisé dans l'aspect DIALOGUE.
Les attributs de relation CONTIENT et DEFINIE_DANS, vers et depuis la classe des classes CLASSE_S, modélisent le fait qu'un schéma contient un nombre quelconque de classes, alors qu'une classe est définie dans un schéma et un seul. Une classe peut être sémantique (CLASSE_SEM) ou non sémantique (CLASSE_NON_SEM), dans ce dernier cas il peut s'agir d'une classe alternative (CLASSE_ALT) ou d'une classe de factorisation (CLASSE_FACT) : on retrouve ici le méta-schéma de SECOOSSE présenté figure 30, à ceci près que la classe intermédiaire CLASSE_NON_SEM a été ajouté. Toutes les classes possèdent le même dialogue d'information, modélisé dans CLASSE_D. Par contre, les classes sémantiques sont représentées avec des bords de 1 point d'épaisseur dans les schémas conceptuels (classe CLASSE1PT) tandis que les classes non sémantiques sont représentées avec des bords de 2 points d'épaisseur (classe CLASSE2PT).
Les attributs de relation TRANSITION_DEPUIS et TRANSITION_VERS présentés sur la figure 30 sont modélisés ici à l'aide de la classe TRANSITION. En effet, les transitions sont des entités qui apparaissent sur les schémas conceptuels. Elles doivent donc avoir une existence intrinsèque pour pouvoir être dessinées ; cette existence est en particulier prise en compte dans les aspects correspondants, TRANSITION_G et TRANSITION_D.

Les attributs-relation POSSEDE et POSSEDEE_PAR entre CLASSE_S et PROPRIETE_S modélisent le fait qu'un nombre quelconque de propriétés peuvent être attachées à une classe, chacune de ces propriétés étant liée à une et une seule classe. Une propriété, conformément au méta-schéma de la figure 31, peut être un ATTRIBUT, de relation ou de valeur (classes ATTRIBUT_REL et ATTRIBUT_VAL), une OPERATION, publique ou privée (classes OPERATION_PUB et OPERATION_PRI), ou une propriété publiée (classe PUBLICATION). Conformément au méta-schéma, attributs, opérations publiques et publications peuvent être publiés, d'où la factorisation de la propriété PUBLIE_EN sur la classe PUBLIABLE. Chacun de ces concepts est représenté différemment dans le schéma conceptuel : les classes aspects RELATION_G, VALEUR_G, PUBLICATION_G, OPERATION_PUB_G et OPERATION_PRI_G modélisent leurs représentations graphiques respectives. De même, les dialogues de définition des attributs-relation, des attributs-valeur, des publications et des opérations sont différents, ce qui entraîne les classes aspects RELATION_D, VALEUR_D, PUBLICATION_D et OPERATION_D. Notons que les opérations, qu'elles soient publiques ou privées, possèdent le même dialogue de définition.

A droite du schéma apparaissent des classes qui ne sont pas mentionnées explicitement dans le méta-schéma de SECOOSSE. Ces classes modélisent les différentes relations, d'héritage ou d'alternative, qui peuvent lier deux classes de l'utilisateur : spécialisation et spécialisation multiple (classes ARC_SPEC et ARC_MSPEC), factorisation (ARC_FACT), alternative initiale et alternative (ARC_PREMALT et ARC_EVO). Pourquoi n'avoir pas conservé des attributs de relation entre classes ? Dans le contexte d'un outil de conception, un arc est une entité qui existe intrinsèquement, qui est visualisée sur l'interface graphique : il est donc indispensable que ces arcs soient modélisés à l'aide de classes. Dès lors, les attributs CLASSE_DEPART et CLASSE_ARRIVEE décrivent le lien entre les arcs et les classes. Les arcs apparaissent dans l'interface graphique (ARC_G) mais ne possèdent pas de dialogue de définition particulier.

Figure 33 : Les concepts gérés par SEISME



E.II. L'interface graphique

La figure 34 met en évidence les classes nécessaires à la gestion d'une interface graphique complète. Un objet graphique, sans préjuger de sa nature, modélisé à l'aide de OBJET_G, possède toujours un certain nombre de caractéristiques :

- une ICONE utilisée pour le dessiner
- un TEXTE dessiné sur l'icône
- une COULEUR
- des caractéristiques propres au gestionnaire d'interface utilisé (BITMAP, POPUP, REFERENCE_POINT).

Tous les concepts du modèle, présentés dans le paragraphe précédent, sont alors représentés à l'aide d'un aspect graphique. Les classes (CLASSE_G) peuvent être des classes "de définition" (CLASSE_DEF), qui représentent un concept de l'utilisateur, ou des classes "de rappel" (CLASSE_RAPPEL), qui ne sont qu'une duplication graphique des classes de définition : les attributs RAPPELEE_PAR et RAPPELLE mémorisent le lien graphique qui unit une classe de rappel à une classe de définition. Ce lien est en particulier utilisé pour mettre à jour les textes des classes de rappel lorsque le nom de la classe de définition est changé, ou pour supprimer les classes de rappel lorsqu'une classe de définition est supprimée. Les classes de rappel possèdent un texte supplémentaire, nommé TEXTE_SCHEMA, qui est dessiné en plus du nom de la classe. Une classe de définition n'est pas nécessairement dupliquée dans le schéma dans lequel elle est définie. C'est pourquoi il est nécessaire de rappeler, à l'aide des attributs DANS_SCHEMA et CONTIENT_RAPPEL, le schéma dans lequel est dessinée une classe de rappel.

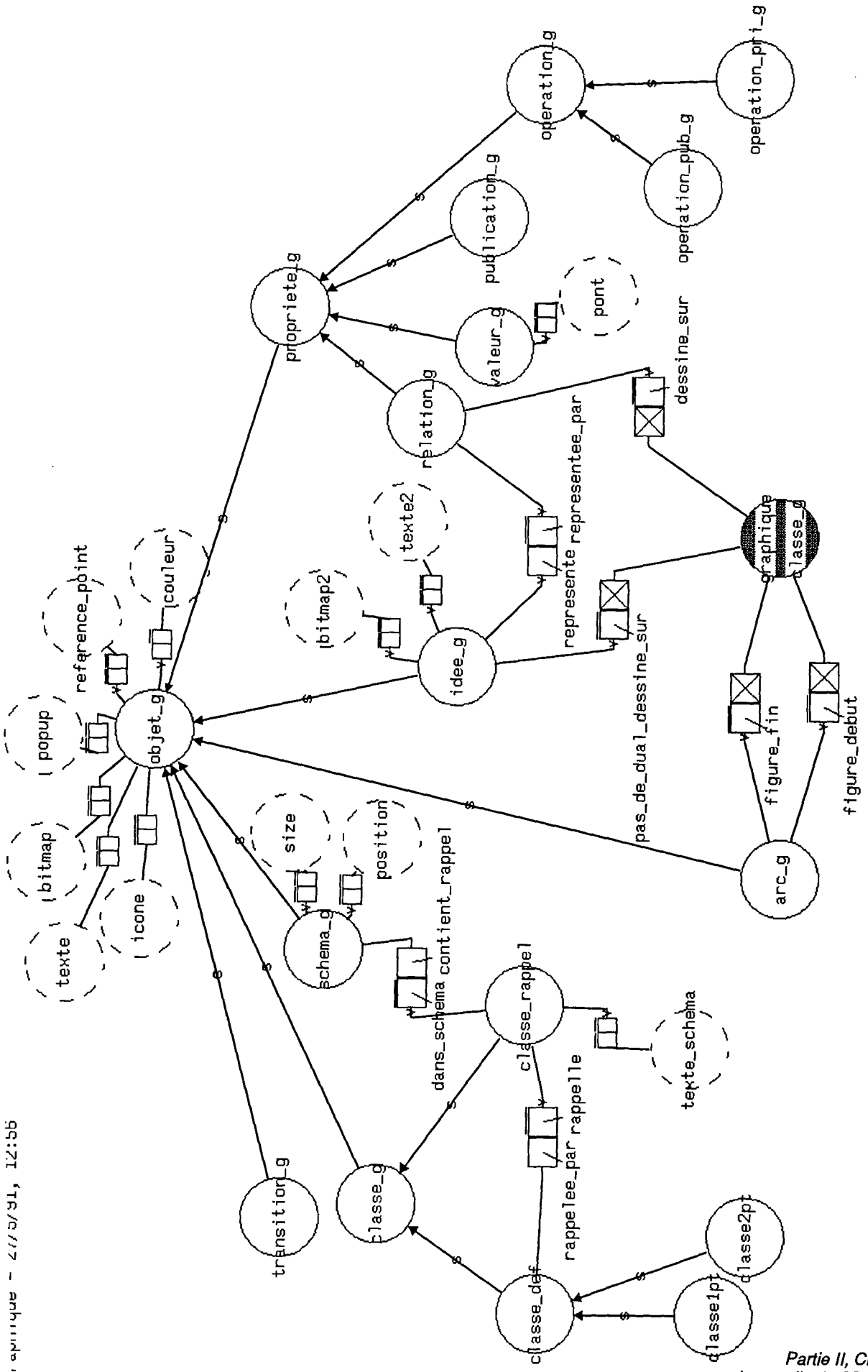
Les schémas, modélisés par SCHEMA_G, possèdent des caractéristiques graphiques particulières, puisqu'il ne s'agit pas de dessins mais de fenêtres : SIZE et POSITION permettent au gestionnaire graphique de placer la fenêtre de chaque schéma.

Les arcs, modélisés par la classe ARC_G, peuvent être dessinés entre des classes qui sont indifféremment des classes définies ou des classes de rappel. C'est pourquoi les attributs CLASSE_DEPART et CLASSE_ARRIVEE, définis dans le paragraphe précédent, ne sont pas suffisants pour dessiner un arc : les attributs FIGURE_DEBUT et FIGURE_FIN indiquent quels sont les objets graphiques qui constituent les extrémités de chaque arc.

Les attributs de relation ne sont jamais représentés seuls, puisque le modèle graphique que nous avons retenu ne contient que le concept d'idée, c'est-à-dire de couple d'attributs. C'est pourquoi un attribut de relation, dans l'aspect graphique (classe RELATION_G), n'est jamais directement dessiné ; par contre, REPRESENTEE_PAR une idée (idée_g), une relation sera DESSINEE_SUR une classe. Dans le cas où cet attribut de relation n'a pas de dual, il est nécessaire de

mémoriser, à l'aide de l'attribut PAS_DE_DUAL_DESSINE_SUR, la classe graphique sur laquelle sera dessinée la croix. Notons qu'une idée est dessinée à l'aide de deux boîtes accolées et contient deux noms d'attributs, ce qui justifie la présence des attributs BITMAP2 (la deuxième boîte) et TEXTE2.
Par comparaison, la représentation graphique des attributs-valeur (classe VALEUR_G) est très simple, puisqu'elle fait seulement intervenir le dessin d'un pont de dénomination. De même, les opérations publiques et privées ainsi que les publications, apparaissant sous forme de rectangles liés à une classe graphique, ne nécessitent pas d'informations supplémentaires.

Figure 34 : L'interface graphique de SEISME



E.III. L'interface par les dialogues

La figure 35 montre les classes nécessaires, organisées en une hiérarchie relativement simple, pour gérer les différents dialogues de SEISME. Un dialogue générique, décrit à l'aide de OBJET_D, contient trois items : NAME_ITEM permet de saisir le nom de l'objet décrit ; COMMENT_ITEM est un texte qui valorise l'attribut COMMENTAIRE de la classe OBJET_S ; QUIT_ITEM décrit un bouton "Quitter".
Le dialogue des opérations, décrit par la classe OPERATION_D, contient, en plus, un texte d'opération (attribut TEXTE_ITEM) ; les dialogues de définition des attributs (classe ATTRIBUT_D) permettent la saisie des cardinalités, minimum et maximum (MIN_CARD_ITEM et MAX_CARD_ITEM) et d'une valeur par défaut.

E.IV. L'application finale

La figure 36 montre la classe 'SEISME', qui représente l'application elle-même. Cette classe est destinée à être instanciée une seule fois, l'objet correspondant ayant la charge de la gestion du fonctionnement de l'application (supervision des actions de l'utilisateur, en particulier). Comme on le voit, cette classe possède plusieurs attributs qui mémorisent l'état interne de l'application : SAUVEGARDE indique si une sauvegarde du travail est nécessaire ; SELECTION mémorise les objets choisis par l'utilisateur, ces objets étant inversés sur les schémas ; PROCHAIN_DIALOGUE et PROCHAIN_SCHEMA mémorisent les coordonnées de la prochaine fenêtre de chaque catégorie à ouvrir ; FICHIER et REPERTOIRE indiquent la localisation de la sauvegarde sur le disque dur.
L'ensemble des schémas est également mémorisé par l'application à l'aide des attributs GERE et GERE_PAR.

Figure 35 : L'interface dialogue de SEISME

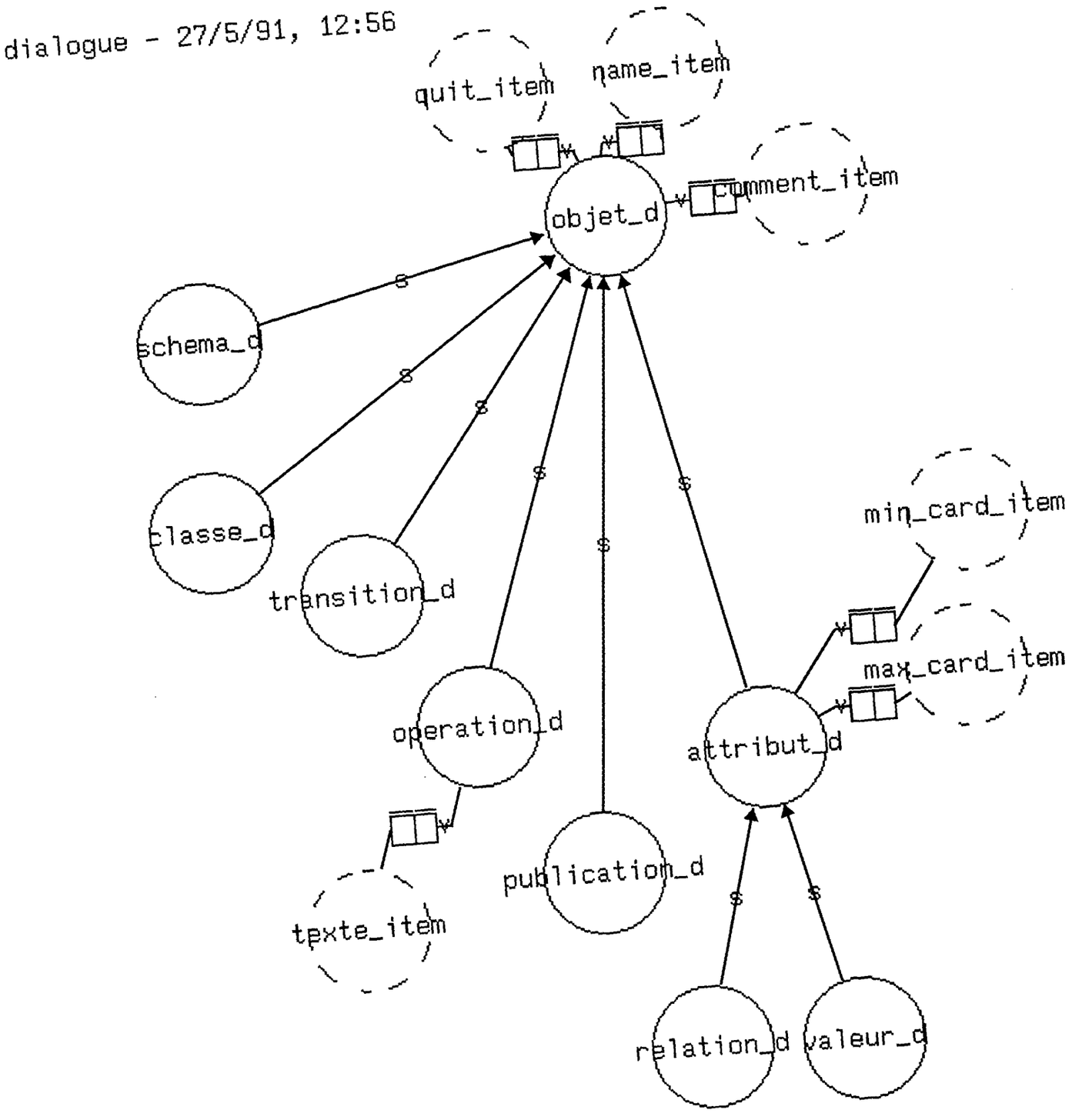
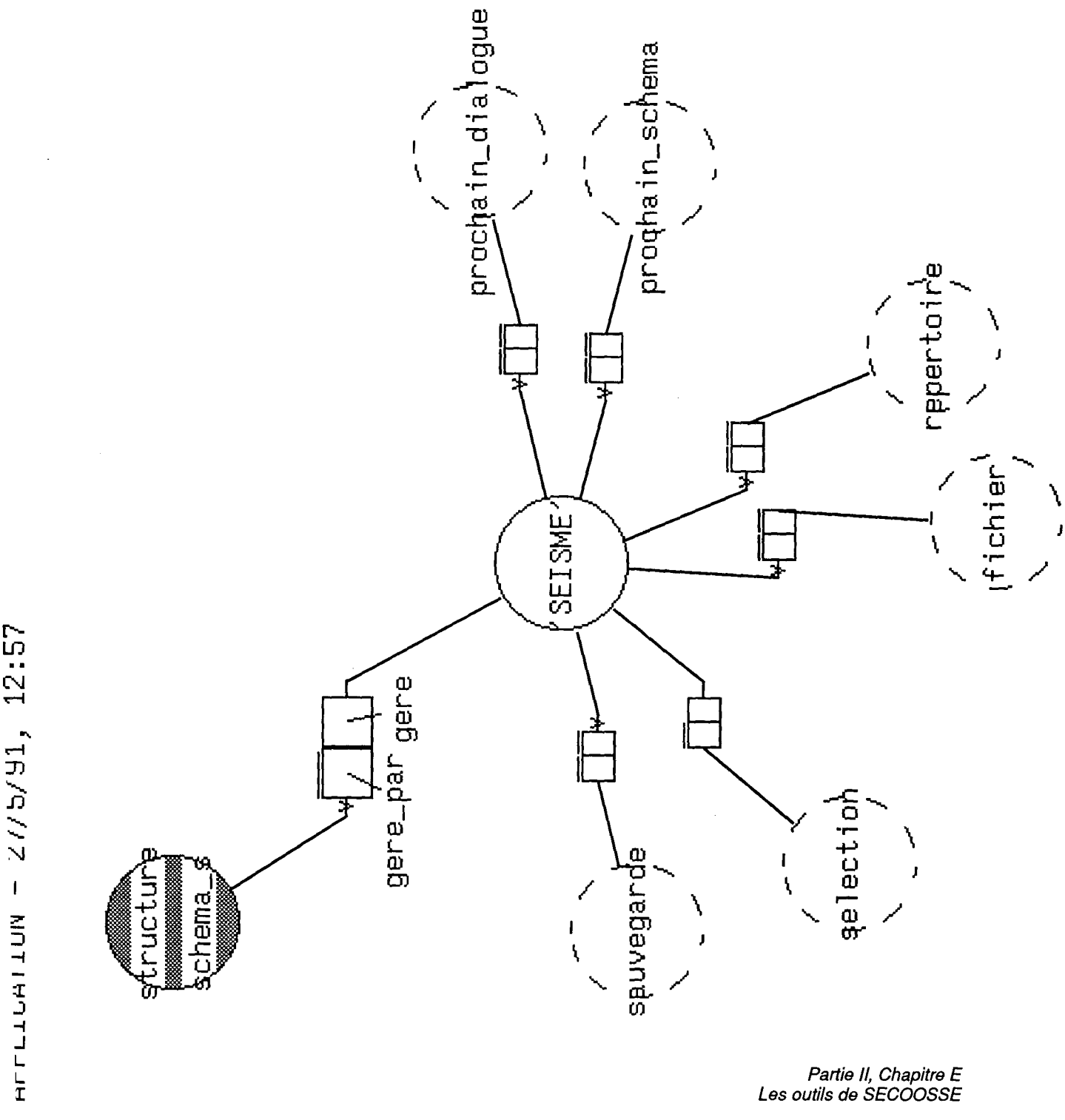


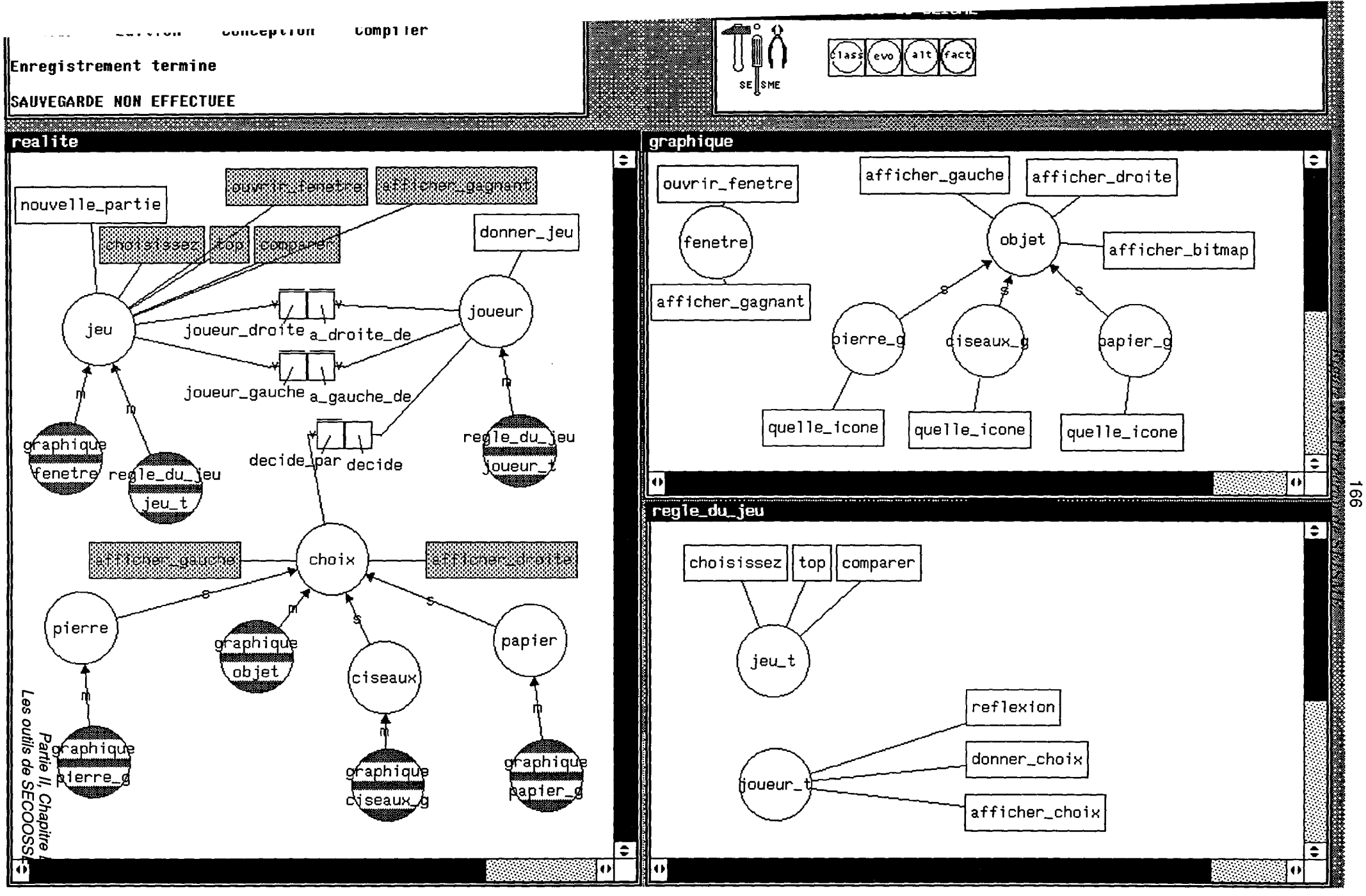
Figure 36 : L'application SEISME



E.V. Conclusion

Nous ne présentons pas ici l'analyse de la dynamique de SEISME, qui conduirait à un volume important d'informations portant très peu de sémantique. Pour les mêmes raisons, nous ne décrivons pas le détail de la réalisation de SEISME. Signalons seulement que nous n'avons pas automatisé la démarche proposée dans le chapitre C : SEISME peut dès lors constituer la base des futurs ateliers de conception utilisés pour le développement des outils d'assistance à la maintenance négociés actuellement au sein de TDF.

Les schémas présentés dans le chapitre sur la validation de SECOOSSE, ceux de ce chapitre, ainsi que la description de STEAMER dans la partie suivante ont tous été réalisés à l'aide de SEISME. Le grand nombre de schémas générés ainsi que leur clarté montre l'intérêt que présente l'outil informatique d'assistance à la conception.
Nous voyons ici que la validation qui a été appliquée à SECOOSSE (la réalisation d'un méta-schéma) a également été réalisée pour SEISME, puisque ce logiciel a été utilisé très tôt pour réaliser sa propre spécification.
En ce qui concerne l'interface graphique, nous présentons figure 37 une copie d'écran de SEISME, mettant en évidence les outils, la palette et trois schémas de conception d'un petit jeu.


Edition Conception Compiler
Enregistrement termine
SAUVEGARDE NON EFFECTUEE
SE SME
class evo alt fact
realite
nouvelle_partie
ouvrir_fenetre
afficher_gagnant
choisissez
top
comparer
donner_jeu
jeu
joueur
joueur_droite
a_droite_de
joueur_gauche
a_gauche_de
decide_par
decide
graphique fenetre
regle_du_jeu jeu_t
regle_du_jeu joueur_t
afficher_gauche
choix
afficher_droite
pierre
graphique objet
ciseaux
papier
graphique pierre_g
graphique ciseaux_g
graphique papier_g
graphique
ouvrir_fenetre
fenetre
afficher_gagnant
afficher_gauche
afficher_droite
objet
afficher_bitmap
pierre_g
ciseaux_g
papier_g
quelle_icone
quelle_icone
quelle_icone
regle_du_jeu
choisissez
top
comparer
jeu_t
reflexion
joueur_t
donner_choix
afficher_choix

Chapitre F
Conclusion

Nous avons proposé une méthode de conception complète, incluant un modèle de données, un modèle de langage, une démarche automatisable et un outil qui intègre ces trois composantes.

Le modèle des données proposé, fondé sur le modèle orienté objets, est étendu à l'aide de deux concepts : l'évolution des objets réels nous a amenés à introduire la notion d'alternative, puis l'intervention de plusieurs concepteurs dans la création des logiciels de maintenance nous a conduit à ajouter la relation de spécialisation multiple. Nous avons conscience qu'un modèle des données est d'autant plus aisé à manipuler qu'il comporte moins de concepts ; l'utilité ou le caractère indispensable des notions que nous avons ajoutées peuvent donc être discutés. Nous les justifions cependant en rappelant qu'un schéma conceptuel obtenu à l'aide d'une méthode de conception doit être aussi lisible que possible ; si tous les concepts étaient manipulés identiquement, par exemple grâce à l'héritage, la maintenabilité des schémas serait réduite dans la mesure où un même arc pourrait avoir plusieurs significations. Par contre, rien ne s'oppose à ce que l'implantation des schémas conceptuels dans un langage particulier fasse appel aux concepts standards.

Le modèle du langage, quant à lui, est fortement inspiré du modèle relationnel binaire, dont il est issu. Nous avons cependant mis en évidence l'excellente adéquation entre le modèle de Information Analysis et le modèle orienté objets que nous avons proposé.

La démarche, enfin, permet d'automatiser, à l'aide d'un algorithme relativement simple, le calcul de concepts qui ne sont pas toujours évidents à manipuler. Seules les relations de spécialisation-généralisation et d'alternative sont obtenues à l'aide des mécanismes exposés, mais il est envisageable d'étendre celui-ci au calcul systématique de toutes les relations, y compris la factorisation et la spécialisation multiple.

Que dire de l'outil, dont le seul intérêt est de permettre la saisie et la maintenance des schémas conceptuels ? Signalons simplement qu'il a été utilisé pour la conception d'un système expert complet, le résultat étant exposé dans la partie suivante.

Nous conclurons cette partie sur une constatation forgée tout au long de ce travail par la poursuite de l'étude bibliographique : alors qu'au début de l'étude menée ici les méthodes de conception orientées objets étaient rares et possédaient toutes des spécificités importantes, l'analyse des dernières publications montre que le nombre de modèles orientés objets et de méthodes associées a récemment considérablement augmenté. En comparant les modèles proposés, par exemple, dans [CASTELLANI 91], [NGUYEN 91] ou [CARRE 89-90a-90b], nous observons :

- d'une part, que notre modèle, à quelques détails près, est très proche de ceux proposés par ces auteurs
- qu'il était temps que ce travail aboutisse, sans quoi il aurait été dépassé avant même d'avoir été imprimé.

La dernière partie de ce mémoire est consacrée à l'application pratique de la méthode SECOOSSE aux systèmes experts de maintenance de seconde génération.

Partie III
Application aux Systèmes Experts de Maintenance et réalisation pratique

Le plus souvent, comme l'indique [VISSER 88], les observations et expériences de méthodes de conception portent sur la conception de petits programmes, les problèmes et remarques déduites de ces études relevant souvent de l'anecdote. La méthode de conception orientée objets SECOOSSE a été créée et étudiée dans le contexte des Systèmes Experts de Maintenance (SEM) et plus précisément dans le cadre du diagnostic et de la réparation des équipements de radiodiffusion. C'est par un constant dialogue entre les concepteurs et nous-mêmes que la méthode a été mise au point. La taille du jeu d'essai, ainsi que le contexte industriel qui impose des résultats pratiques évidents, montrent que les différentes remarques, loin de constituer des anecdotes, relevaient de problèmes réels. C'est pourquoi nous considérons que SECOOSSE est autant validée théoriquement, comme nous l'avons vu dans la partie précédente, à l'aide de son modèle réflexif, qu'elle l'est pratiquement à l'aide de la réalisation de STEAMER.

Cette réalisation a tout d'abord été fondée sur un grand nombre de réunions de cognitique, c'est-à-dire des entretiens avec les techniciens de différentes circonscriptions de TéléDiffusion de France. Ces entretiens, au nombre d'une quarantaine, se sont déroulés sur une période de 18 mois. Nous proposons ici, *in extenso*, un exemple de cahier des charges grossier, synthèse des seize premières réunions [MAFILLE 89] :

«

Les réunions cognitiques constituent la partie prise d'expertise lors de la construction d'un système expert. Dans notre cas, développement d'un générateur de systèmes experts, elles ont pour but d'établir la base de connaissance et de spécifier le formalisme de représentation de la connaissance.

Adopter un outil informatique utilisant uniquement un arbre de défaillance et une base de règles est apparu très difficile, compte tenu de la complexité du système et de la fréquence des réunions.

La méthode de diagnostic du technicien, s'appuyant largement sur une dizaine de schémas électriques, nous incite à essayer de reproduire son approche du problème sous la forme de déplacement au sein d'un graphe.

Les symptômes externes de panne se trouvant indiqués sur le bandeau de la chaîne de sécurité, sur les bandeaux annexes et sur les différents voltmètres,

ampèremètres, cet ensemble est le point d'entrée du système (nous nous plaçons dans le cas d'une panne franche).

1 LES STRATEGIES DE DIAGNOSTIC DU TECHNICIEN

Les problèmes auxquels le technicien doit répondre sont variés (électriques, électroniques, thermiques, mécaniques). Dans ce qui suit nous analysons deux cas typiques de diagnostic.

1.1 AUTOMATICITE DE L'EMETTEUR

Classiquement, à partir du bandeau de sécurité (tel qu'il apparaît sur les schémas de l'automaticité générale) le technicien :

1- *Prend en compte toutes les leds indiquant une fonction réalisée (ventil RF1, chauffage 382, etc.) Tous les modules nécessaires à la fonction sont déclarés corrects.*

2- *Part de la première led (led1) indiquant un défaut et répertorie tous les modules qui participent au fonctionnement de la led et ce jusqu'aux modules dont le bon fonctionnement est assuré par l'étape 1. Une liste [1] des modules suspects est donc obtenue.*

3- *Les leds étant regroupées par grandes fonctions (ventil, chauffage, etc.), si la led1 appartient à un groupe au sein duquel d'autres leds sont en défaut alors il refait l'étape 2 avec chacune des leds en défaut. Les listes obtenues sont unies et la liste [2] obtenue contient les modules suspects et le nombre d'occurrences de chaque module (le degré de suspicion de panne d'un module augmente lorsque la sortie de celui-ci est commune à deux -ou plus- branches déclarées défaillantes).*

4- *Recherche tous les témoins attachés aux modules de la liste [2] susceptibles de la réduire (leds sur carte, bandeaux annexes, vu-mètre, etc). On obtient la liste [3].*

5- *Ordonne la liste [3] des modules à contrôler en fonction des critères d'occurrence et d'accessibilité.*

6- *Teste l'état des module dans l'ordre de [3] jusqu'à trouver le module défaillant.*

7- *Si le module n'est pas terminal, à partir de la sortie déclarée défaillante, il y a recherche de tous les modules qui participent à la sortie, On crée de nouveau une liste [3] et les étapes 6, 7 sont relancées.*

DEFINITIONS

Dans le cas général, un module est une collection de modules terminaux. Un module est terminal lorsque l'entité qu'il représente est celle que l'on change au moment de la réparation.

Pour connaître l'état d'un module, le technicien peut le tester (prendre une mesure) en dynamique (le module étant alimenté) ou en statique (module déconnecté). Le technicien prend en compte plusieurs facteurs, dont :

- *la mesure est aisée ou non (par exemple, la commande chauffage est protégée par un capot)*
- *la mesure en dynamique est dangereuse ou non (par exemple,, par mesure de sécurité élémentaire, l'accès aux modules comportant de la haute tension est interdit tant que celle-ci est présente)*
- *la mesure en dynamique est mécaniquement possible ou non (par exemple, il est impossible de tester un module dont les connexions sont électriquement isolées)*

Le rôle du technicien étant de trouver l'origine de la panne en un minimum de temps, la stratégie de diagnostic doit tenir compte de tous ces impératifs au niveau 5.

1.2 TRAITEMENT DU SIGNAL

La méthode de diagnostic du technicien relative à cette partie de l'émetteur est simple. En effet la chaîne de traitement du signal étant fortement bouclée, le technicien isole la fonction en cause et change les composants suspects jusqu'à ce que le système fonctionne de nouveau. Il serait souhaitable de rencontrer le constructeur afin d'établir si une méthode plus fine n'existe pas. A priori seul le constructeur est en mesure d'apporter ce type de renseignement.

Pour cette partie et en l'état actuel de la prise d'expertise, nous ne pouvons proposer de représentation de la connaissance et de méthode de diagnostic qui soient satisfaisantes.

2 LA REPRESENTATION DE LA CONNAISSANCE

Il est possible de donner une représentation du système de la façon suivante :

Il consiste en un schéma (synoptique) de l'émetteur représentant les différents modules interconnectés entre eux.

Un module est soit :
- un châssis (ensemble de composants et/ou cartes électroniques)
- une carte électronique (ensemble de composants)
- un composant (module terminal)

Des fonctions de transfert attachées à chaque module permettent de retrouver tous les modules nécessaires au bon fonctionnement du module considéré (chaque sortie d'un module possède une fonction de transfert, c'est l'ensemble des entrées concourant à l'obtention d'un signal de sortie correct).

De plus à chaque module est associé l'ensemble des points de mesures et contrôles possibles (mesures en dynamique , type de contrôle: visuel, auditif) [point 4 de la stratégie de diagnostic].

Cette représentation constitue le niveau 1 de l'émetteur, c'est celui qui permet l'obtention d'une liste de modules suspects sans avoir fait une seule mesure.

Lorsque qu'un module est déclaré défaillant et qu'il s'agit d'un châssis ou d'une carte, la partie sous-jacente est montrée. Les mesures possibles sont indiquées (dynamiques ou non, etc). La démarche (points 5,6 et 7 du paragraphe 1.1) se poursuit jusqu'à l'élément défaillant.

Nous essayons ainsi d'avoir une représentation naturelle du système à diagnostiquer.

3 CONCLUSIONS

Si de grandes lignes se dégagent de cette synthèse, il n'en demeure pas moins l'obligation de valider bien des points. Par rapport à l'organisation actuelle, du transfert d'expertise (3 réunions/mois), un contact permanent avec un spécialiste en maintenance permettrait un retour d'information plus efficace. De même un dialogue soutenu avec le constructeur est souhaitable (le constructeur propose des stages sur les cartes de traitement du signal).

»

Ce cahier des charges initial, qui présente les grandes fonctionnalités tant du point de vue des données que dans l'aspect du raisonnement de diagnostic, a plus tard été affiné, en particulier en fonction de la démarche proposée dans SECOOSSE. Nous ne décrivons pas ici le détail des transformations de ce cahier des charges, de même que nous ne donnons pas dans cette partie la traduction en phrases binaires (étant donné le volume final d'informations, un rapport en plusieurs tomes serait à peine suffisant). Par contre, nous fournissons dans le chapitre A les spécifications conceptuelles telles qu'elles étaient exploitées en fin de cette étude. Le chapitre B discute de l'implantation de ces schémas dans un langage appelé LAP. Le chapitre C, en forme de conclusion sur la réalisation de STEAMER, présente les grandes lignes des travaux qui sont prévus dans le cadre des systèmes experts de maintenance.

Chapitre A
Spécification conceptuelle

Le projet de SEM mené par les laboratoires F.M.S. puis T.M.I., nommé STEAMER (***SysTème Expert d'Aide à la Maintenance d'Equipements de Radiodiffusion***), consiste en une étude complète des opérations de maintenance menées par les techniciens de TéléDiffusion de France, depuis la lecture des schémas électriques jusqu'aux réflexes implicites de réparation. Le but affirmé de STEAMER est de réunir, en une seule application, ce vaste spectre de connaissances pour proposer un outil complet et surtout d'emploi aisé.
Le nombre relativement important de domaines d'expertise (citons notamment la lecture des documentations papier, les mécanismes de diagnostic et les spécifications de réparation, pour ne considérer que les aspects techniques de STEAMER) a amené à scinder les caractéristiques en deux sous-applications : un outil logiciel, nommé GRACE (***Graphisme de Représentation Analytique de Composants Electromécaniques***) permet la saisie, sous une forme proche des documentations fournies par les constructeurs des équipements, de toutes les informations nécessaires au fonctionnement de STEAMER lui-même, c'est-à-dire du mécanisme de diagnostic et d'assistance à la réparation.

Les connaissances ont alors été divisées en plusieurs domaines :

- dans GRACE, les objets sont représentés en fonction :
  - . de la structure des documentations qui seront saisies à l'aide de cet outil
  - . de l'interface de cet outil
- dans STEAMER, les connaissances sont organisées en fonction du diagnostic et de l'assistance à la réparation :
  - . le fonctionnement électromécanique des équipements constitue un domaine d'expertise à part entière, comme le montrent les nombreuses études qui ont été menées sur exclusivement sur ce sujet
  - . le diagnostic nécessite une formalisation particulière des équipements ; cette formalisation a été obtenue à l'aide de très nombreux entretiens

avec les techniciens qui se chargent du diagnostic des émetteurs considérés

Ces différents domaines d'expertise, s'ils représentent chacun un ensemble de connaissance cohérent, ne sont pourtant pas indépendants puisque, par exemple, le fonctionnement d'un équipement est fondé sur la description de l'équipement, elle-même obtenue à l'aide des documentations. Nous sommes ici en présence de l'exacte définition du problème de multi-expertise qui nous a amené à introduire, dans la partie précédente de ce rapport, la notion de spécialisation multiple.
C'est pourquoi STEAMER et GRACE sont fondés sur une connaissance particulière qui consiste en une représentation objective de la réalité : comme le préconise la démarche de SECOOSSE, la conception du logiciel a débuté avec la conceptualisation des équipements sans *a priori* graphique ou technique, comme le décrit le paragraphe A.I ; chaque domaine d'expertise a ensuite donné lieu à un sous-schéma conceptuel particulier, les relations entre ces différents schémas et le schéma objectif étant réalisées à l'aide de la spécialisation multiple. Ainsi, les paragraphes A.II à A.IV décrivent les formalisations conceptuelles de ces différents domaines.

A.I. Représentation objective des équipements

Les figures 38 et 39 présentent le schéma conceptuel qui décrit la structure objective des équipements électromécaniques. La figure 38 décrit seulement les classes objectives et les propriétés publiées sur ces classes, tandis que la figure 39 décrit les classes objectives et leurs propriétés.

Dans le modèle de maintenance utilisé pour construire STEAMER, un équipement électromécanique est toujours composé de la chaîne suivante :

une ***unité structurelle*** - une ***jonction*** - une ***liaison*** - une ***jonction*** - une ***unité structurelle*** ...

Nous appelons UNITE STRUCTURELLE un morceau de l'équipement qui, admettant des signaux d'entrée, délivre des signaux de sortie, chacun composé à partir d'un sous-ensemble des signaux d'entrée.
Une JONCTION est un point de contact -soudure, borne, ...- qui permet à une unité structurelle d'acquérir un signal ou de délivrer un signal.
Une LIAISON est le véhicule d'un signal entre deux jonctions.

Les unités structurelles peuvent appartenir à deux catégories :

- un AGREGAT contient lui-même un ensemble d'unités structurelles ; la notion d'agrégat permet donc de considérer l'équipement selon une hiérarchie : l'agrégat le plus global, l'équipement lui-même, est divisé en plusieurs agrégats (dans le cas des émetteurs considérés, il s'agit de trois baies), qui eux-mêmes sont subdivisés jusqu'aux unités terminales, les ***feuilles***.
- une FEUILLE ne contient aucune autre unité structurelle : les feuilles constituent le plus bas niveau auquel l'équipement peut être considéré ; il existe plusieurs catégories de feuilles dans les équipements électromécaniques :
  - . les BLOCS, feuilles au sens le plus général
  - . les TEMOINS, qui peuvent être des voyants lumineux (LEDS ou Light Emitting Diodes) ou des appareils de mesure (VOLTMETRES, AMPEREMETRES, WATTMETRES, ...)
  - . les feuilles équipées d'INDICATEURS, comme les RELAIS et les RELAIS THERMIQUES (leur position peut être visualisée) ou les FUSIBLES, monophasés ou triphasés (dont l'état, grillé ou non, est indiqué par un néon)
  - . les BORNIERS, dont le rôle se limite à transférer de l'information d'une liaison vers une autre liaison
  - . certaines feuilles qui imposent l'ARRET DE L'EQUIPEMENT pour être observées.

Les jonctions sont soit au DEPART d'une liaison, soit en FIN d'une liaison ; toute jonction est associée à une feuille (attribut-relation BORNE_DE).

Dans le cas de certaines liaisons, la présence ou la correction du signal qu'elles véhiculent peut être observée directement à l'aide de témoins ; cette relation entre liaisons et témoins est mise en évidence par l'attribut-relation A_POUR_TEMOIN_DIRECT.

Les notions de BANDEAU et de GROUPE servent à formaliser l'interface que présente l'équipement au technicien. Un bandeau est un ensemble de témoins relativement aisément accessible ; les témoins d'un bandeau sont organisés en groupes qui dénotent certaine corrélation entre les fonctions testées. Par exemple, sur le bandeau principal des émetteurs, les leds qui concernent la ventilation des tubes appartiennent au même groupe. De plus, les groupes sont ordonnés entre eux (attributs-relation AVANT et APRES). En effet, les

fonctionnalités dont un groupe témoigne sont dépendantes : par exemple, si la ventilation ne fonctionne pas, l'alimentation des tubes d'amplification ne peut être réalisée (il s'agit de sécurités en chaîne).

Figure 38 : Représentation objective de l'équipement(hiérarchie et publications)

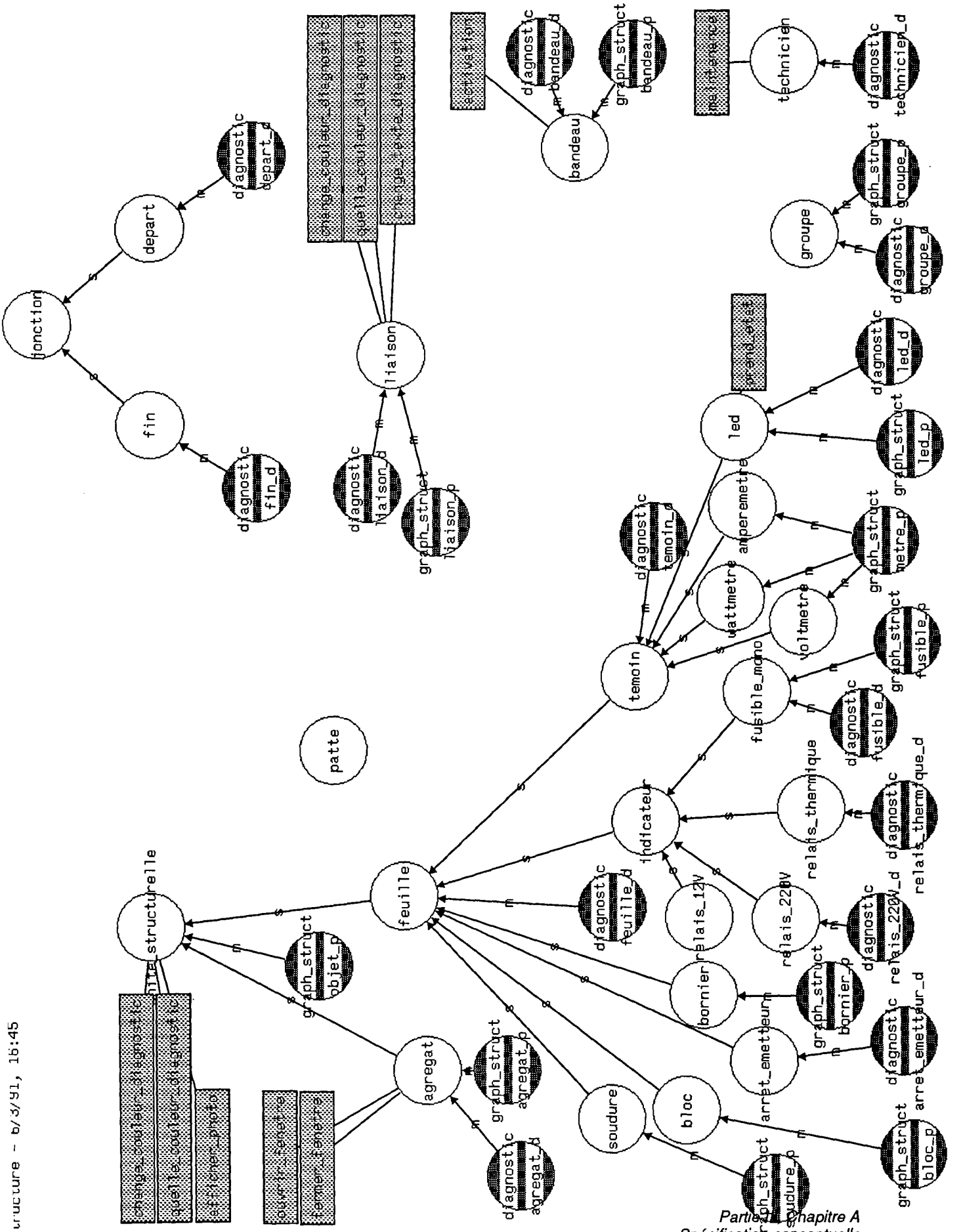
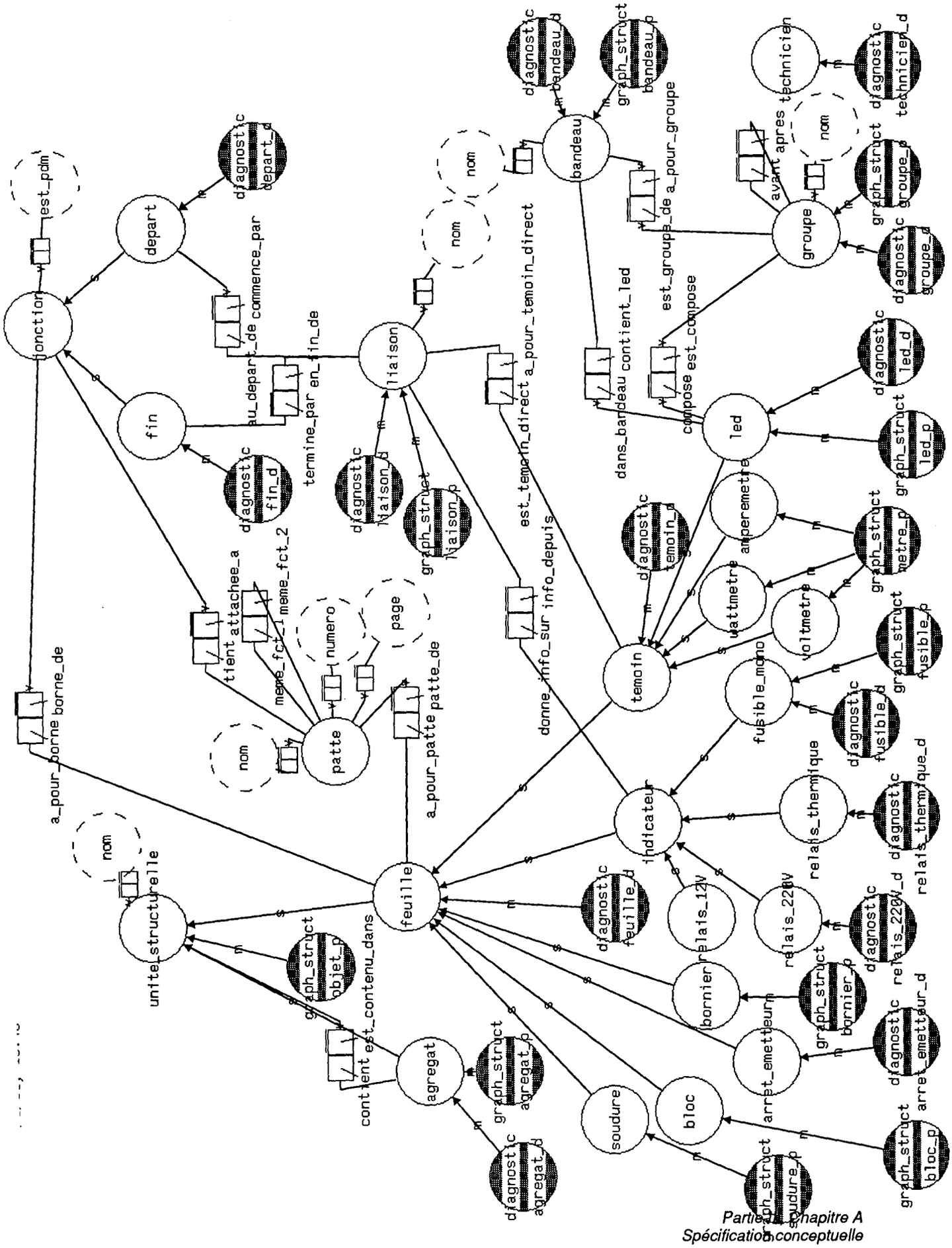


Figure 39 : Représentation objective du fonctionnement de l'équipement (hiérarchie et attributs)



A.II. Fonctionnement des équipements

La figure 40 illustre la conceptualisation du fonctionnement des équipements. Ce fonctionnement est fondé sur le concept de ***fonction de transfert***, et est lié à l'***état*** de chaque signal. En effet, nous avons vu que le rôle des unités structurelles est de transformer un ensemble de signaux d'entrée en un ensemble de signaux de sortie, éventuellement avec une transformation égale à l'identité. Une première hypothèse -de simple bon sens- consiste à associer à chaque liaison un et un seul signal : dès lors, l'étude fonctionnelle des signaux est strictement identique à l'étude fonctionnelle des liaisons.

Le principe d'une fonction de transfert est le suivant : étant données une unité structurelle et une liaison de sortie de cette unité, lorsque cette liaison est dans un certain état, alors l'état d'un sous-ensemble des entrées de l'unité peut être déduit, par l'inverse de la transformation réalisée par l'unité structurelle. Nous nous rapprochons ici des études sur la physique qualitative (Cf. [DEKLEER 83], [FORBUS 84], [KUIPERS 86]), mais en ne considérant qu'un ensemble discret d'états possibles pour chaque signal ; en effet, la formalisation d'une transformation continue est beaucoup plus complexe, notamment dans le cas de composants électroniques tels le transistor ou le circuit intégré.

Il existe donc une ou plusieurs FONCTIONS DE TRANSFERT pour chaque liaison ; lorsqu'une liaison de sortie d'une unité prend un ETAT donné (attributs POUR_LA_SORTIE et QUAND_LA_SORTIE_PREND), alors plusieurs AFFECTATIONS sont réalisées, chaque affectation consistant à donner, à une liaison d'entrée, un état particulier (attributs AFFECTE et DONNE). De la sorte, connaissant l'état d'une liaison, il est possible de déduire, par récurrence sur les fonctions de transfert, l'état de toutes les liaisons qui sont en amont : le fonctionnement de l'équipement est formalisé.

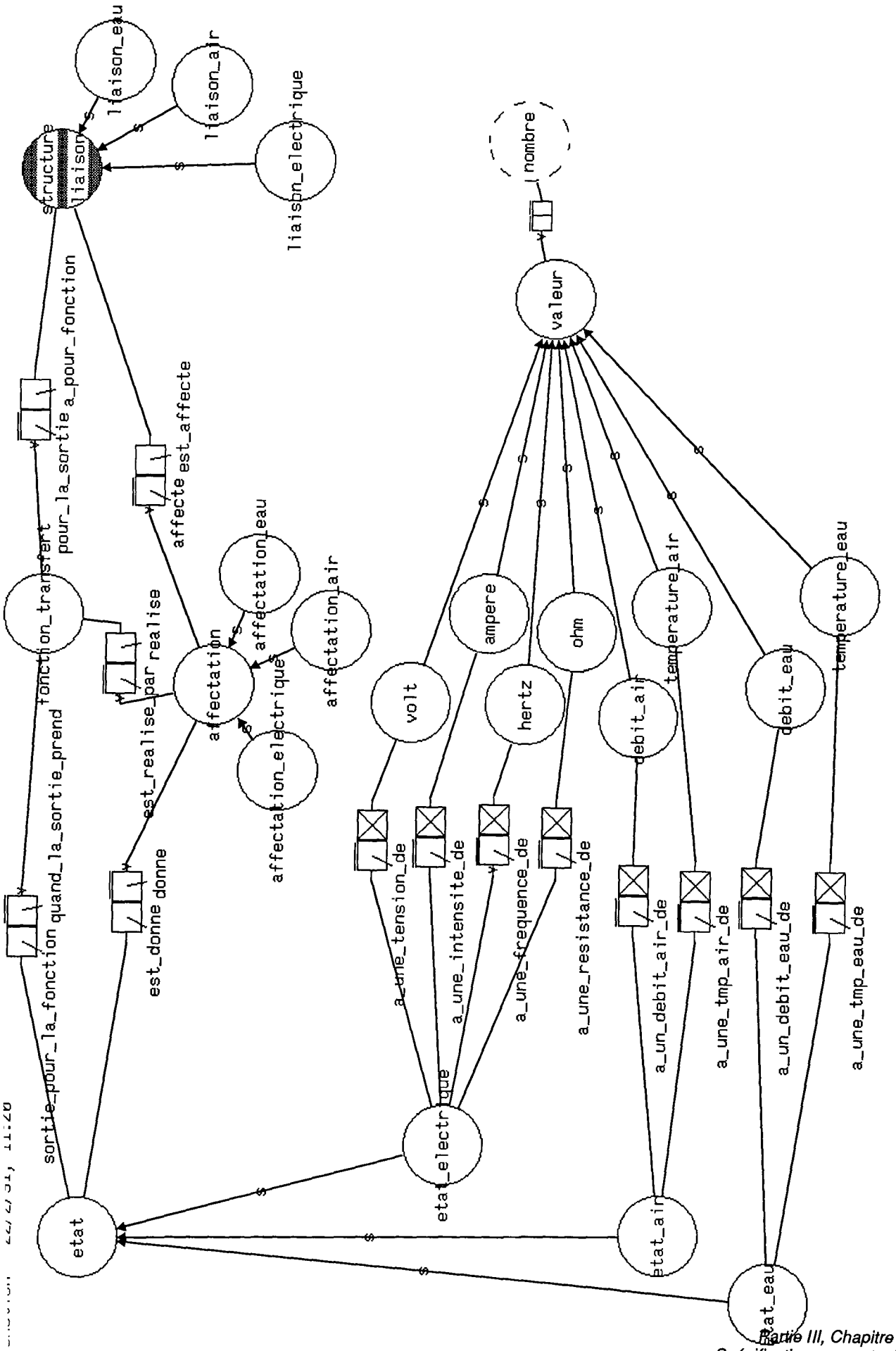
La figure 40 met en évidence une spécialisation des états et des liaisons ; en effet, les signaux véhiculés par les liaisons peuvent être de nature électrique, mais aussi fluide (notamment air ou eau dans les cas de refroidissement). Cette spécialisation a notamment pour but de détailler la description des valeurs qui peuvent être prises par les signaux :

- un signal électrique est décrit par une TENSION, une INTENSITE et une FREQUENCE, la liaison pouvant être RESISTANTE

- un flux d'air et un flux de liquide sont caractérisés par un DEBIT et une TEMPERATURE.

Toutes ces grandeurs sont matérialisées par des nombres ; ces nombres pourraient être formalisés à l'aide d'attributs-valeurs sur les classes ETAT_ELECTRIQUE, ETAT_AIR et ETAT_EAU. Cependant, la classe des VALEURS a été créée car les grandeurs mentionnées peuvent faire l'objet de traitements particuliers dans certains cas d'utilisation de la modélisation de l'équipement. Par exemple, dans une application destinée à la formation, une tension pourra faire l'objet d'une illustration représentant l'écran d'un oscilloscope ; ou bien l'ensemble de manipulations qui permet de vérifier une valeur peut être mis en évidence à l'aide d'une séquence, animée ou non, d'images ...

Figure 40 : Le fonctionnement de l'équipement



A.III. Interface STEAMER-utilisateur : GRACE

L'interface entre le système de maintenance et l'utilisateur utilise un double concept : d'une part, chaque objet existe dans une base d'objets d'interface ; ces objets d'interface, dont la description, relativement complexe, est donnée figure 41, sont eux-mêmes représentés à l'écran un nombre quelconque de fois sous une forme graphique : la figure 42 montre le schéma conceptuel qui formalise les objets graphiques.

Les objets d'interface sont donc représentés figure 41. Nous allons décrire les grandes lignes de ce schéma conceptuel sans nous attarder sur les détails. Parmi les classes principales de ce schéma, nous observons :

- A_DIALOG, qui est la classe de tous les objets dont les caractéristiques sont définissables dans une fenêtre particulière (nommée dialogue).
- AGREGAT_P et FEUILLE_P, qui font irrésistiblement penser aux classes AGREGAT et FEUILLE définies dans la représentation de l'équipement.
- parmi les sous-classes de FEUILLE_P, certaines sont dessinées à l'aide de rectangles : elles sont factorisées par RECTANGLE_P. D'autres possèdent des caractéristiques graphiques particulières, telles que la limitation en nombre d'entrées ou de sorties (classes de factorisation UNE_SEULE_ENTREE et UNE_SEULE_SORTIE).

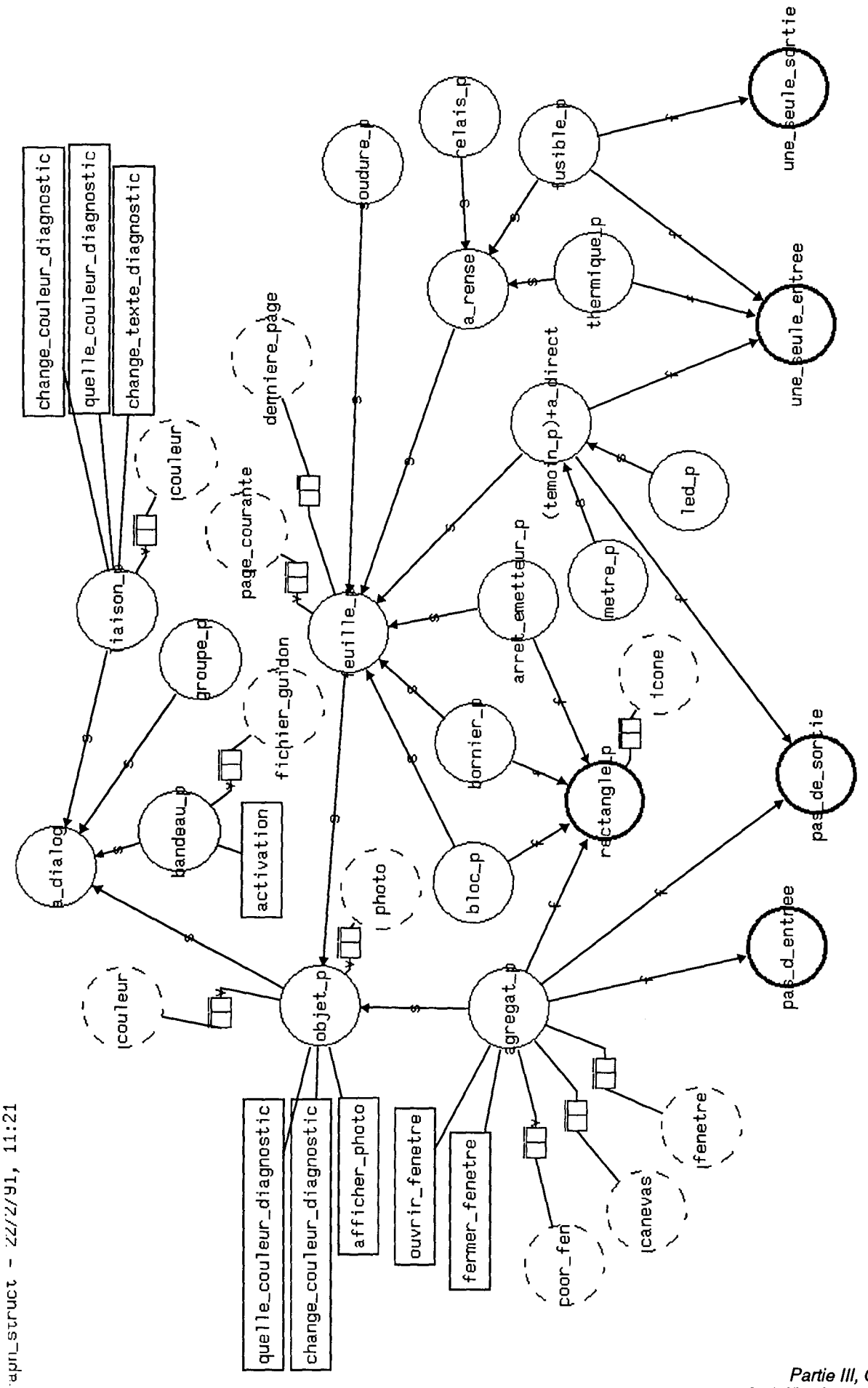
Toutes ces classes possèdent de plus des attributs nécessaires à la gestion de l'interface graphique (par exemple PHOTO, FENETRE, ...)

Les objets graphiques, qui sont directement affichés à destination de l'utilisateur, sont quant à eux organisés en une hiérarchie plus simple. Une différence fondamentale est faite entre les objets qui sont BORNES et ceux qui ne le sont pas (NON_BORNE) : les NON_BORNEs sont les représentations graphiques des entités manipulées ; les bornes en constituent des rappels graphiques, mentionnant leur nom, dans les autres schémas électriques.
Parmi les non_bornes, certains objets sont dessinés à l'aide de rectangles tandis que d'autres font appel à des images prédéfinies (classe A_BITMAP). Parmi ces derniers, certains peuvent posséder une image qui tourne de 90° (classe TOURNABLE, avec l'attribut POSITION qui mémorise l'angle courant de chaque objet tournable).

Ces deux schémas, les objets d'interface et les objets graphiques, sont liés par quatre attributs de relation et leurs duaux : DESSIN_DE, REPRESENTE, BORNE_DE_G et BORNE_DANS.

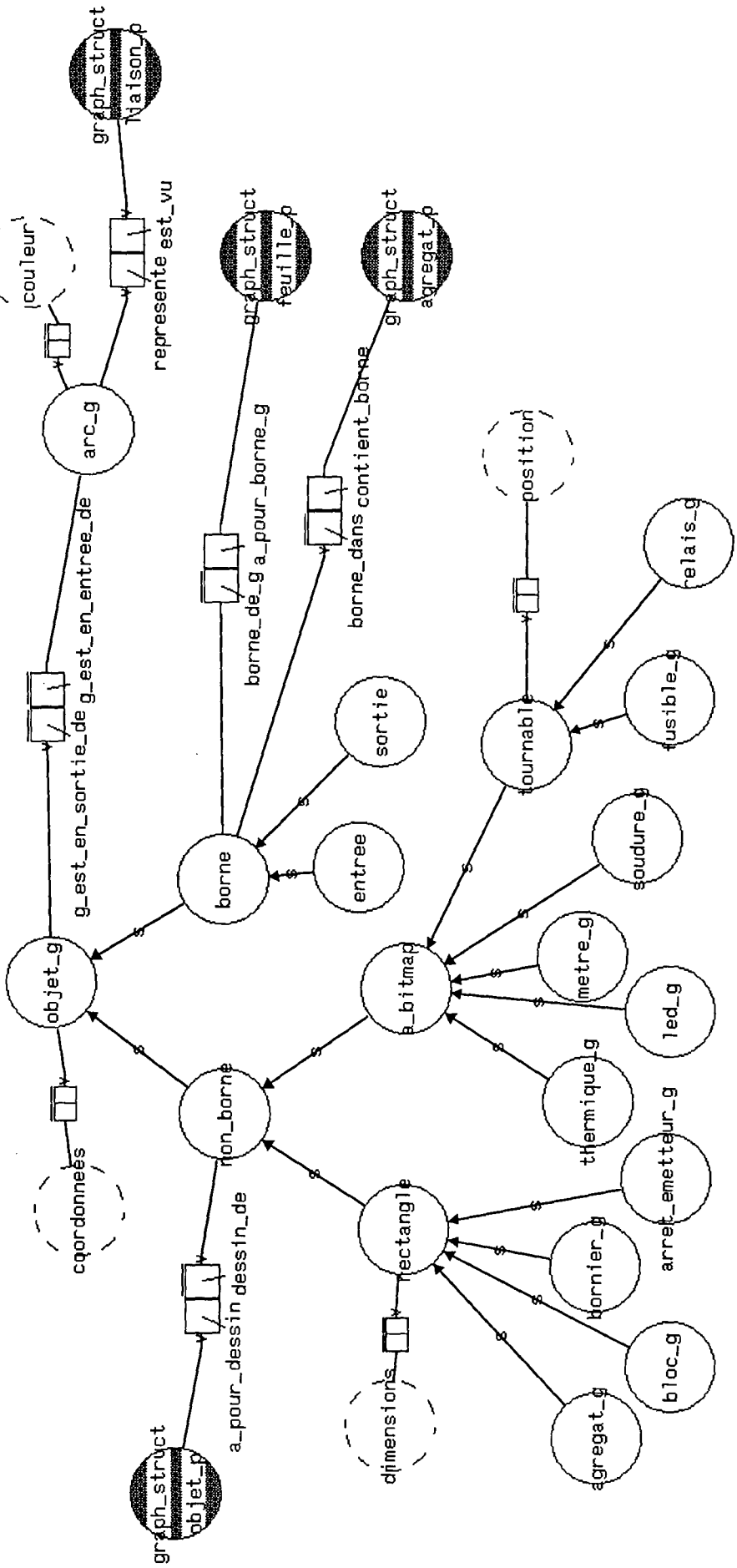
La complexité de cette spécification de l'interface graphique de STEAMER illustre clairement, beaucoup mieux que tout exemple, l'intérêt de la spécialisation multiple. En effet, sans ce concept, chaque concepteur aurait été obligé de manipuler les objets en tenant compte de leurs caractéristiques graphiques qui, comme on le voit, sont très loin de la simplicité. Au contraire, l'utilisation de la spécialisation multiple permet de ne faire apparaître, sur le schéma structurel utilisé par tous les concepteurs, que certaines caractéristiques présentes dans l'interface : c'est ainsi que la figure 38 ne mentionne que la publication des propriétés QUELLE_COULEUR_DIAGNOSTIC, CHANGE_COULEUR_DIAGNOSTIC, AFFICHER_PHOTO (...) parmi toutes celles qui sont définies figure 41.

Figure 41 : GRACE, l'interface de STEAMER



graph_struct - 22/2/91, 11:21

Figure 42 : GRACE, les objets purement graphiques



A.IV. Diagnostic

Le diagnostic, illustré en figure 43, porte essentiellement sur les liaisons elles-mêmes ; en effet, les mesures de test effectuées par les technicien sont, le plus souvent, réalisées sur les jonctions qui lient ces liaisons aux unités structurelles.
La règle fondamentale de diagnostic est :
«Si un module est en panne alors cette panne se traduit par une valeur incorrecte en sortie du module lorsque les entrées sont correctement appliquées.»

On notera qu'un module présentant des défauts peut ne pas avoir de manifestations de pannes extérieures !

Le diagnostic consiste à considérer toutes les feuilles de l'équipement comme suspectes, puis, à l'aide des mesures pertinentes, à réduire l'ensemble des feuilles suspectes jusqu'à ce qu'il n'en reste plus qu'une, qui est alors en panne. En réalité, cette stratégie générale est souvent modifiée par des heuristiques locales qui soit en améliorent la rapidité, soit permettent de détecter plusieurs cas de pannes réciproques.
Le fonctionnement du système de diagnostic est de façon générale le suivant :

- toutes les feuilles sont INCONNUES (aucune mesure n'a été effectuée)
- le bandeau principal de l'équipement contient, en cas de panne, des témoins "incorrects" (allumés au lieu d'éteints, par exemple) ou "corrects". Sous l'hypothèse que les défaillances des témoins sont détectées manuellement, chaque témoin correct certifie que le signal qui l'alimente est correct : en utilisant les fonctions de transfert, il est possible d'INNOCENTER toutes les liaisons en amont de ce signal. A cette étape, aucun ensemble de suspects n'a encore été construit ; par contre, on sait d'ores et déjà quelles liaisons sont hors de cause.
- Nous avons vu que les témoins d'un bandeau sont organisés en groupes ; les groupes sont utilisés ici pour ne pas considérer comme suspects tous les témoins incorrects, mais seulement ceux du groupe le plus en amont ; en effet il est normal que les témoins des groupes en aval soient incorrects puisque la fonction qui les alimente n'a pas été déclenchée du fait des sécurités.

  C'est pourquoi seuls les témoins incorrects du groupe incorrect le plus en amont sont considérés. Pour chacun de ces témoins, on connaît l'état que l'on attendait pour le signal qui l'alimente ; ce signal, donc l'unité qui le génère, est suspect, et l'état attendu est propagé vers l'amont : toutes les liaisons en amont d'un témoin incorrect deviennent suspectes ; cependant,

cette propagation des suspicions cesse naturellement dès qu'une liaison innocentée est rencontrée. Notons que certaines liaisons sont en amont de plusieurs témoins incorrects : la probabilité que le signal qu'elles portent soit incorrect est très forte. Ce cas de figure est mémorisé (il est nommé OCCURRENCE d'une liaison).

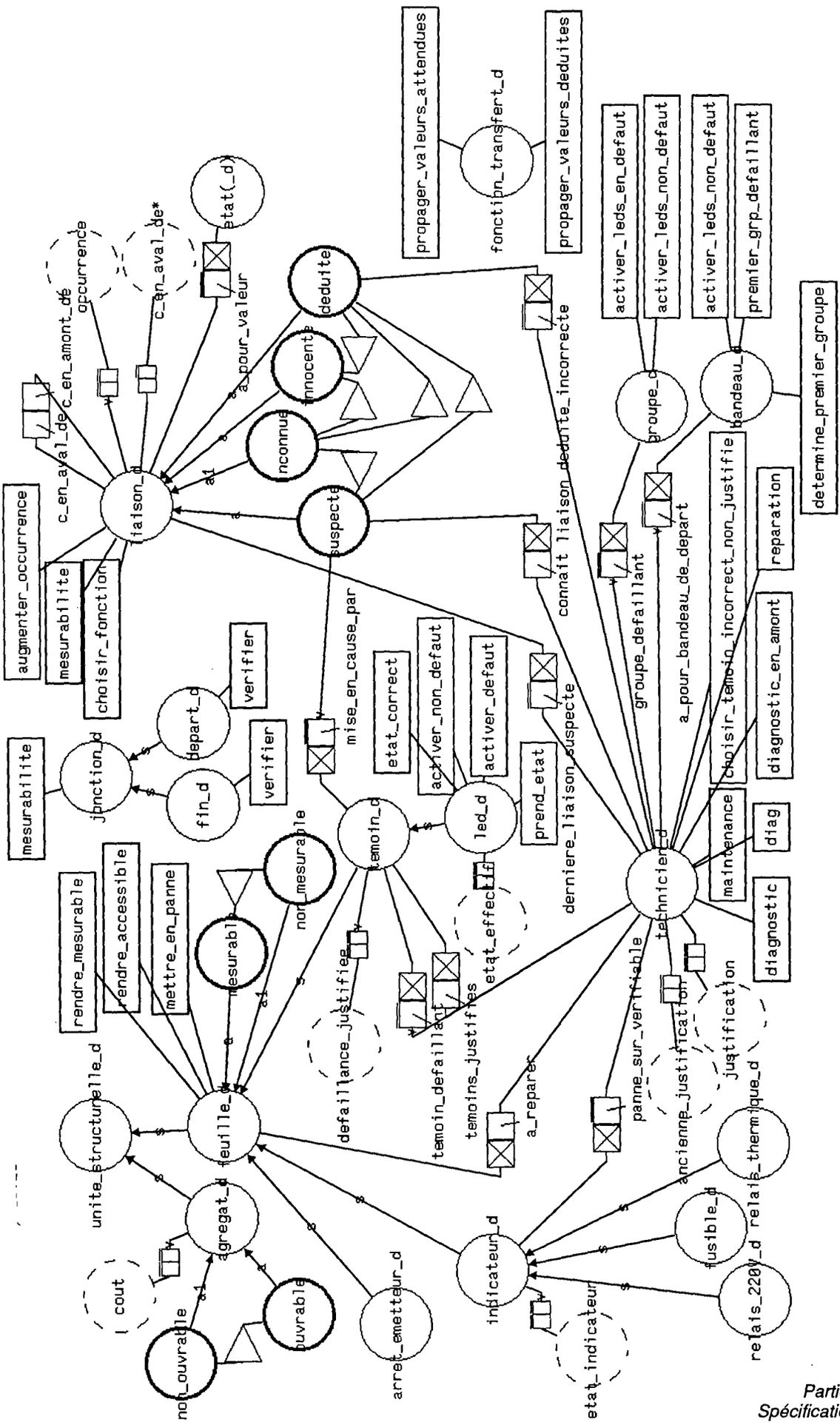
- Le diagnostic lui-même consiste à établir un ordre sur les liaisons suspectes obtenues ; cet ordre fait intervenir des notions -heuristiques- de fiabilité, de sécurité et d'accessibilité. Pour chaque liaison, selon cet ordre, une mesure est demandée. Si le résultat de la mesure correspond à la valeur attendue, la liaison est innocentée et par conséquent toutes ses liaisons en aval sont également innocentées : la panne se situe en aval. Si par contre la mesure ne retourne pas la valeur attendue, alors la défaillance est en aval, et toutes les liaisons en amont sont innocentées, donc retirées de l'ensemble des suspects. Si il n'existe plus de liaisons, en amont, qui soient suspectes, alors l'unité à l'origine de la dernière liaison testée est défaillante et doit être réparée.

Nous avons notamment vu intervenir les notions de liaisons INCONNUES, SUSPECTES et INNOCENTES ; ces concepts sont formalisés sous forme d'alternatives de la classe des liaisons. La figure 43 indique qu'il existe une quatrième alternative, les liaisons DEDUITES. Cette alternative permet de différencier les liaisons qui ont été innocentées "théoriquement" par le système de diagnostic, de celles qui l'ont été "pratiquement" par des mesures. En effet, une liaison dont l'innocence a été déduite peut être mesurée et éventuellement redevenir suspecte, voire coupable : il s'agit là de l'une des heuristiques qui résultent de nombreux entretiens avec les techniciens, et qui rapproche le fonctionnement du système de diagnostic des méthodes utilisées par ceux-ci (remise en cause).

L'ensemble des liaisons suspectes est mémorisée à l'aide de l'attribut-relation CONNAIT.
Les alternatives OUVRABLE et NON_OUVRABLE d'une part, MESURABLE et NON_MESURABLE d'autre part, permettent de formaliser l'état physique de l'équipement durant le diagnostic, notamment pour tenir compte de l'accessibilité des composants à mesurer.

Les autres classes qui apparaissent dans la figure 43 font partie des spécialisations multiples des classes représentant la structure objective de l'équipement.

Figure 43 : Le diagnostic



Chapitre B
Réalisation

Depuis sa création, le CERLOR a pris part à des développements informatiques dans le contexte de la maintenance. STEAMER n'est pas, loin de là, la première étude dans ce domaine : deux prototypes de systèmes de maintenance, fondés sur des règles et un moteur d'inférence, avaient été précédemment conçus. Petit à petit, un ensemble d'outils ont été créés, favorisant la recherche dans le domaine de l'application de l'informatique aux problèmes de maintenance. En particulier, la plupart des réalisations antérieures à ce travail ont été réalisées en Prolog. Lorsque l'intérêt du modèle orienté objets pour les études en cours s'est fait sentir, la décision a été prise de ne pas perdre les connaissances acquises en matière de programmation logique : c'est pourquoi un langage orienté objets entièrement développé et programmable selon Prolog, nommé LAP, a été acquis (Cf. [LAP 89], [BURNEAU 89] pour une étude de ce langage). C'est donc en LAP que STEAMER a été réalisé.

Après avoir comparé brièvement les concepts du langage LAP à ceux du modèle orienté objets standards, nous montrerons comment le méta-schéma du langage peut être modifié pour qu'y soient intégrés les concepts propres à SECOOSSE.

B.I. LAP et le modèle orienté objets

Le concept de classe du modèle orienté objet est présent en LAP sous le nom de *model*. Les attributs sont présents dans LAP sous le nom de *slot*. LAP possédant un modèle méta-circulaire, ces slots sont instances d'une méta-classe, donc possèdent eux-mêmes des slots (les facettes), qui sont entre autres : le type (atome Prolog ou un model), les cardinalités, la valeur par défaut, la possibilité d'inférer une valeur, ... LAP gère également des *links*, qui correspondent très exactement à la fois à l'attribut-relation présenté dans SECOOSSE et à l'idée-type de IA, c'est-à-dire qu'un link est composé de deux slots duaux typés réciproquement. Les opérations du modèle standard sont présentes en LAP sous le nom de *method* ; LAP n'inclut pas, ni explicitement ni implicitement, de concept de message. L'héritage est présent en LAP, qu'il s'agisse d'héritage

multiple ou non. Dans le cas de l'héritage multiple, le choix de la propriété à utiliser parmi plusieurs propriétés homonymes définies sur plusieurs super-classes est réalisé ... en fonction de la date comparée de la création des super-classes !
LAP possède également des concepts particuliers : tout événement dans la base d'objets, qu'il s'agisse d'une création d'instance, d'un déclenchement d'opération, d'une valorisation d'attribut (...) peut être précédé et suivi de l'exécution de réflexes, sortes d'opérations qui n'attendent pas d'être explicitement déclenchées pour s'exécuter.
LAP possède une notion très floue de l'encapsulation. Dans la mesure où ce produit est fondé sur Prolog, les atomes et les prédicats sont toujours visibles. Par conséquent, toute propriété, de tout objet, peut être manipulée quel que soit l'objet initial.

B.II. Implantation du méta-schéma de SECOOSSE en LAP

Pour que les schémas conceptuels générés à l'aide de SECOOSSE puissent de façon simple être utilisés pour une réalisation en LAP, nous nous sommes proposé d'implanter les concepts originaux de SECOOSSE dans le méta-schéma de ce langage. Un accord de collaboration avec les auteurs de LAP nous a permis de réaliser un début d'implantation. Malheureusement, pour des raisons commerciales, cette collaboration a été rapidement rompue. En effet, le méta-schéma d'un langage est ce qui en constitue à la fois la richesse et la puissance. Mettre à la disposition d'un client la totalité de ce méta-schéma et, en particulier, ses primitives d'implantation est donc, pour une société commerciale, un risque important. C'est pourquoi nous nous limitons à exposer le méta-schéma fondamental de LAP, puis nous en indiquons les modifications possibles. Ces modifications ont pour but de permettre aux concepts de spécialisation multiple et d'objets évolutifs d'exister en tant que tels dans le langage.

La description d'un méta-schéma passe obligatoirement par la description des instanciations. C'est pourquoi nous utilisons un composant graphique qui n'est pas défini dans le langage conceptuel de SECOOSSE : il s'agit de l'arc d'instanciation, qui indique, pour un objet, quelle est sa classe. Cet arc d'instanciation est labellé à l'aide d'un «i» (nous l'avons déjà rencontré dans le paragraphe D.III (page 151), durant la description du modèle réflexif de SECOOSSE).

B.II.1. Méta-schéma de LAP

La figure 44 met en évidence l'organisation de base, relativement simple, du méta-schéma de LAP. Outre le méta-schéma, représenté en partie haute, nous avons indiqué une classe de l'utilisateur, RELAIS, ainsi qu'une instance de cette classe, K37.
Le méta-schéma de LAP est basé sur la classe META_MODEL, classe générique de toutes les classes de LAP. Instance de META_MODEL, le modèle META_OBJECT définit de façon générale tous les objets de LAP. Notons que META_MODEL est lui-même un modèle (instance de lui-même), mais également un objet (lien de spécialisation entre META_MODEL et META_OBJECT). Le méta-schéma à proprement parler s'arrête là. Les auteurs du langage ont cependant spécialisé META_MODEL en SYSTEM_MODELS, qui est instancié par un grand nombre de modèles nécessaires au langage lui-même. De même, ils ont créé USER_MODELS, destiné, quant à lui, à être instancié par l'utilisateur (de nombreuses contraintes sont vérifiées sur USER_MODELS, alors qu'elles ne le sont pas dans la hiérarchie supérieure).
Lorsque l'utilisateur instancie USER_MODELS, c'est à dire qu'il crée, par exemple, le modèle RELAIS, ce modèle est automatiquement factorisé par USER_MODEL, qui elle-même spécialise META_OBJECT. Cette factorisation automatique des modèles de plus haut niveau de l'utilisateur permet d'exprimer le fait que toute instance-utilisateur (ici un relais, K37) possède les propriétés d'un objet, définies sur META_OBJECT. Nous pouvons considérer USER_MODELS et USER_MODEL comme une interface entre le méta-schéma de LAP et les modèles de l'utilisateur.

La figure 45 montre la structure du méta-schéma de LAP pour ce qui est de la gestion des propriétés. Deux modèles, META_SLOTS et META_METHODS, décrivent les modèles qui seront instanciés avec des attributs et des opérations. Les modèles SLOTS et METHODS décrivent de façon générique les propriétés, attributs et opérations. A ce méta-niveau, les auteurs du langage ont ajouté les modèles SYSTEM_SLOTS et SYSTEM_METHODS, destinés à être instanciés avec les propriétés nécessaires au langage lui-même. Enfin, les modèles USER_SLOTS et USER_METHODS seront instanciés avec les propriétés de l'utilisateur.

Figure 44 : Méta-schéma des modèles de LAP

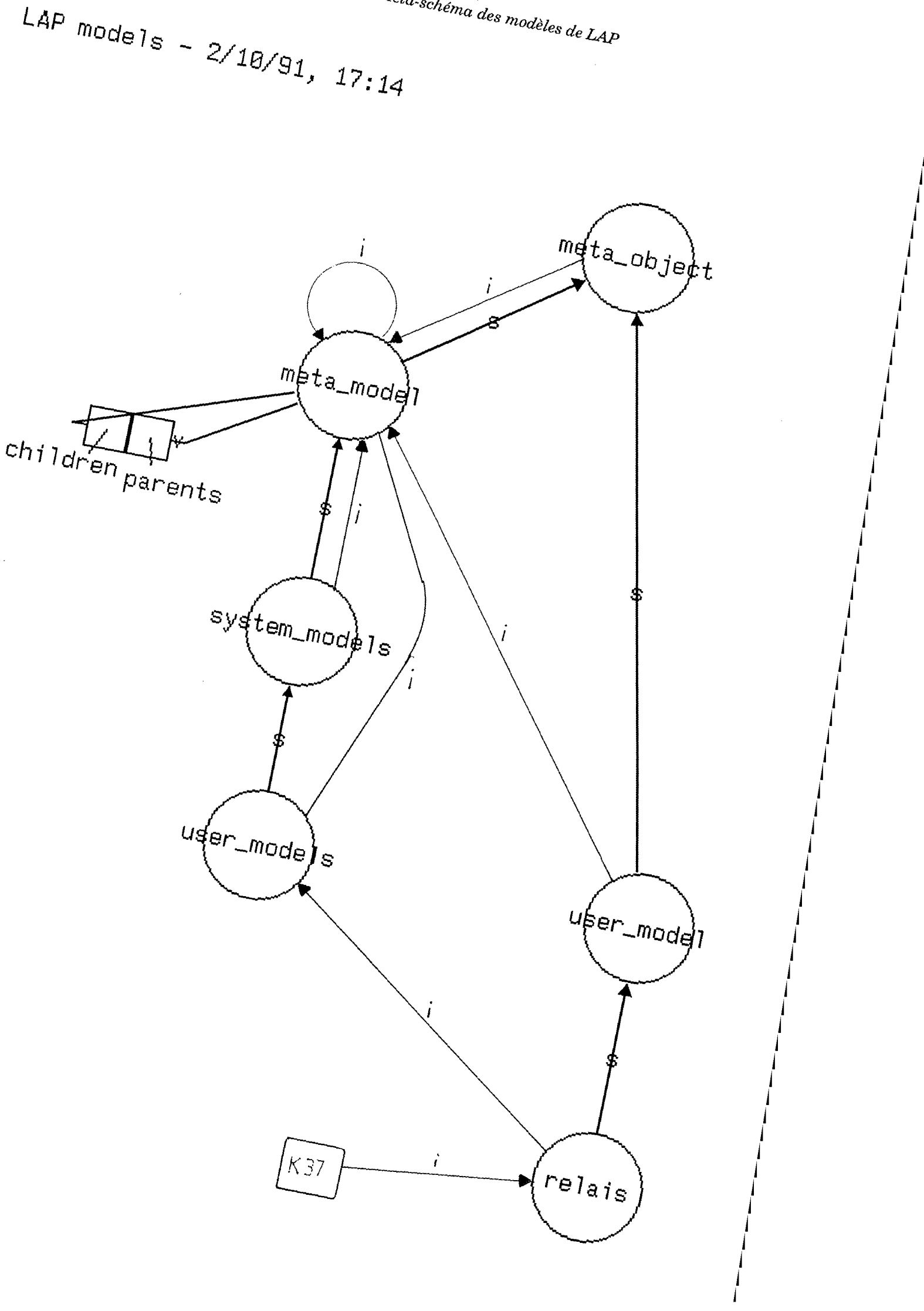
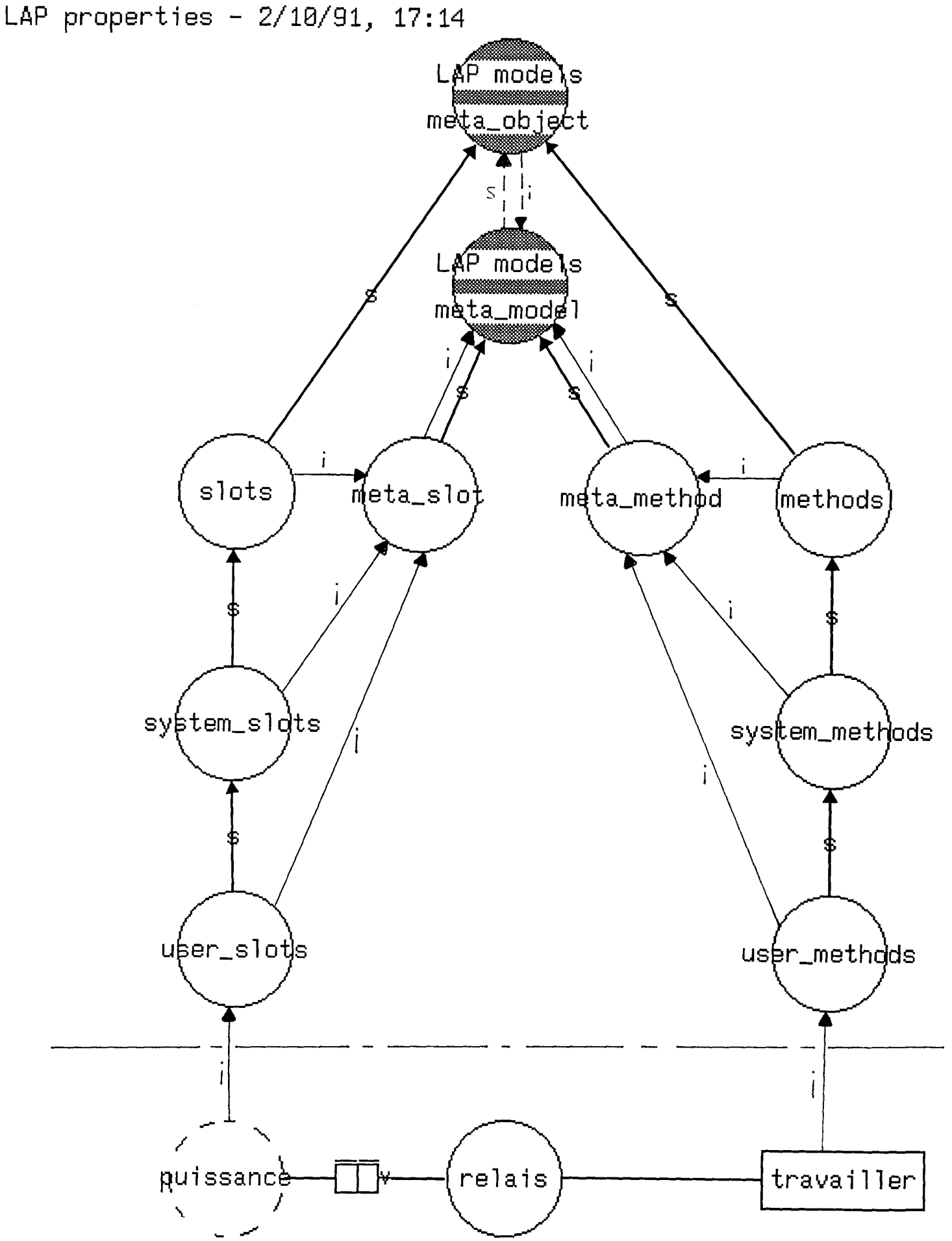


Figure 45 : Méta-schéma des propriétés de LAP



B.II.2 Enrichissement

Le méta-schéma de LAP autorise un objet à être instance d'une classe et d'une seule. Malheureusement, et c'est là qu'une intervention au niveau des primitives d'implantation aurait été nécessaire, cette contrainte est câblée à un niveau très bas et ne peut être modifiée par l'utilisateur, en particulier parce qu'une bonne partie des optimisations du fonctionnement de LAP est fondée sur cette restriction. L'enrichissement du méta-schéma de LAP pour implanter les concepts de spécialisation multiple et d'objet évolutif réside donc dans la possibilité de considérer un objet en tant qu'instance de plusieurs classes (prédicat *objet(O,C)* du modèle de SECOOSSE).
Une modification du méta-schéma porterait alors à trois niveaux :

- d'une part, au niveau de la définition des classes, il est nécessaire de mémoriser les relations d'alternative et de spécialisation multiple.
- deuxièmement, au niveau des objets, nous choisissons d'instancier séparément toutes les classes connexes, puis de signaler aux instances ainsi créées qu'elles représentent la même entité réelle.
- troisièmement, il faut permettre aux propriétés définies sur l'une des classes connexes d'être accessibles par les instances des autres classes connexes.

B.II.2.a) Définition des classes

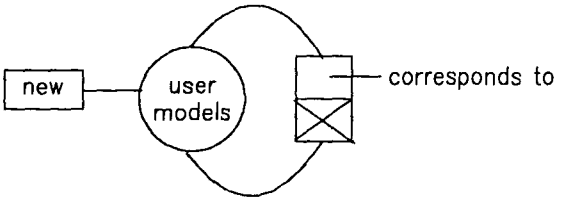


Figure 46 : Modification de la définition des classes de LAP

Pour définir les classes connexes dans le méta-schéma, nous modifions légèrement la classe des classes de l'utilisateur, USER_MODELS (Cf. fig. 46). En effet, nous ajoutons un attribut-relation, CORRESPONDS_TO, qui indique, pour chaque classe, quelles sont les classes «équivalentes» (c'est-à-dire les classes alternatives pour une classe évolutive, la classe évolutive correspondant à une classe alternative, la classe objective pour une classe aspect). Des règles simples permettent de valoriser cet attribut pour chaque classe à partir du schéma conceptuel.
L'opération NEW d'instanciation d'une classe est redéfinie de sorte que lorsqu'une classe est instanciée, toutes les classes correspondantes le sont également. Les instances créées dans toutes ces classes sont alors liées entre elles comme l'explique le paragraphe suivant.

B.II.2.b) Définition des objets

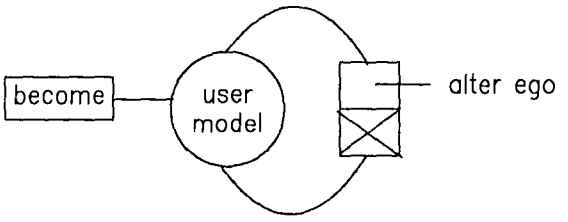


Figure 47 : Modification de la définition des objets de LAP

Nous avons choisi de représenter une entité réelle par plusieurs instances. Il est cependant nécessaire que chaque instance connaisse ses *alter_ego*, c'est-à-dire les autres instances qui participent à la représentation de la même entité. C'est pourquoi nous modifions la classe générique des objets de l'utilisateur, USER_MODEL, en lui ajoutant l'attribut ALTER_EGO. Lors de l'instanciation d'un ensemble de classes par l'opération NEW sur USER_MODELS, toutes les instances créées, dans les différentes classes, sont valorisées pour cet attribut.
Pour gérer les objets évolutifs, nous ajoutons également sur USER_MODEL l'opération BECOME, dont le rôle est de supprimer l'instance alternative parmi celles qui représentent l'entité réelle concernée, avant de créer une nouvelle instance alternative dans une autre classe alternative. Au cours de cette action, l'opération BECOME met à jour, naturellement, l'attribut ALTER_EGO de toutes les instances concernées par cette modification.

B.II.2.c) Définition des propriétés

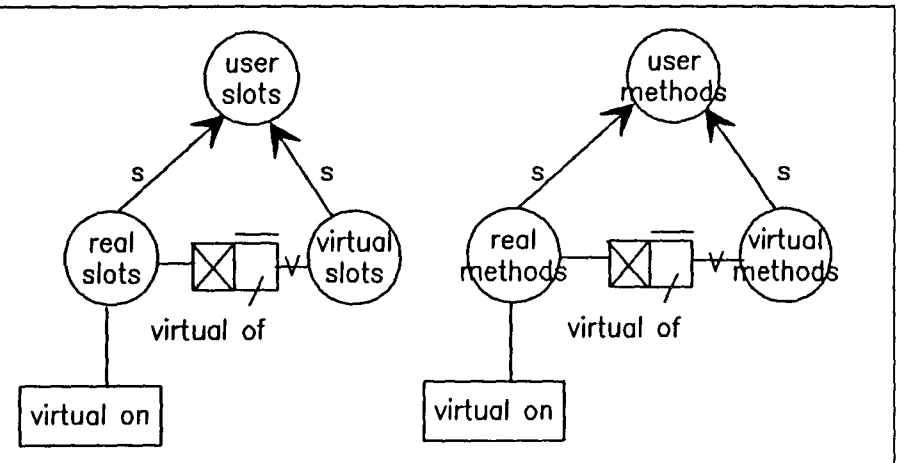


Figure 48 : Modification de la définition des propriétés de LAP

L'intérêt de l'appartenance d'un objet à plusieurs classes réside, comme nous l'avons vu dans le modèle de SECOOSSE, dans la possibilité qu'a cet objet d'accéder aux propriétés de toutes ces classes. Dans notre cas, chacune des instances créées ne voit que les propriétés définies sur sa classe propre. C'est pourquoi nous devons introduire la notion de *propriété virtuelle* qui, définie sur une classe, offre aux instances de cette classe la possibilité d'accéder à une propriété définie sur une classe connexe. Ainsi, la figure 48 montre comment le méta-schéma des propriétés est modifié. La classe USER_SLOTS est spécialisée par :

- la classe des REAL_SLOTS, qui contient désormais, à la place de USER_SLOTS, les attributs réellement définis sur une classe
- la classe des VIRTUAL_SLOTS, un slot virtuel ayant pour fonction de décrire quel est l'attribut réel -et la classe connexe correspondante- qui doit en fait être déclenché. L'attribut VIRTUAL_OF mémorise, pour chaque slot virtuel, le slot réel qui doit être manipulé. La possibilité qu'offre LAP d'inférer une valeur pour un slot permet alors de réaliser, très simplement, l'accès au slot réel.

La classe des slots réels possède quant à elle une opération VIRTUAL_ON, dont le rôle est de créer un slot virtuel associé à un slot réel.
Le méta-schéma des opérations, étant exactement symétrique, est modifié de façon identique.

B.II.2.d) Exemple d'instanciation

La figure 49 montre un exemple de schéma conceptuel à implanter. La figure 50 indique comment est réalisée cette implantation à l'aide des nouveaux concepts. Nous avons matérialisé sur cette figure les relations entre objets à l'aide de l'icône des idées-types, de telle sorte que chaque relation puisse être nommée.

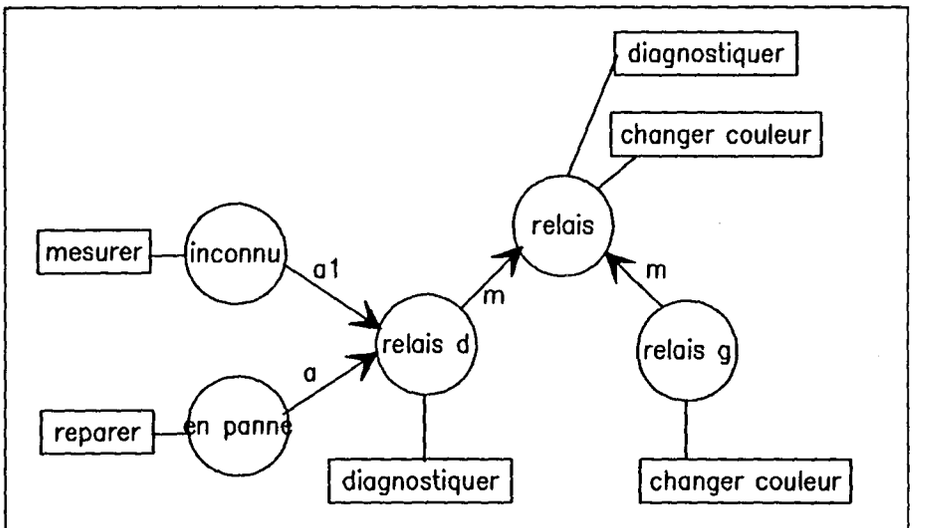


Figure 49 : Exemple de schéma conceptuel à implanter

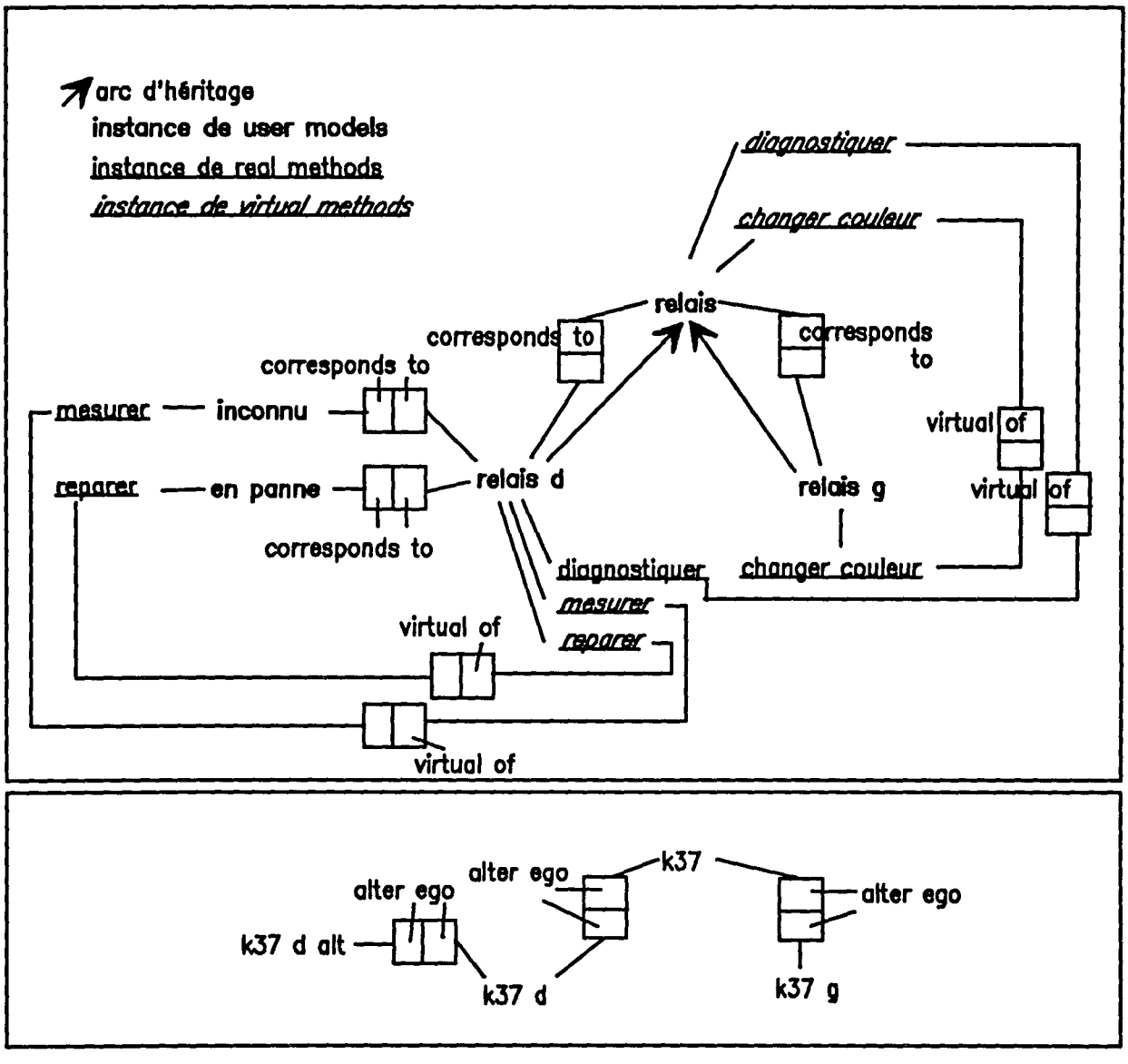


Figure 50 : Implantation du schéma conceptuel précédent

B.III. Réalisation réelle

Les premières réalisations de STEAMER prenaient en compte certaines notions de l'enrichissement du méta-schéma de LAP présenté précédemment. Cependant, trois arguments ont amené à une modification de cette solution :

- d'une part, les modifications n'ont jamais pu être implantées complètement, pour les raisons citées précédemment.
- d'autre part, le volume de la base de connaissance est tel que la multiplication du nombre des instances impliquée par la solution retenue

alourdit considérablement le système, à la fois en place mémoire et disque et en temps de réponse (la figure 51 montre un exemple de schéma structurel qui ne représente qu'une petite partie de la représentation de l'équipement).

- enfin, la réactivité du système de maintenance, tel qu'il est prototypé en LAP, est insuffisante : en théorie, moins d'une seconde devrait s'écouler entre une action de l'utilisateur et la réponse du système. En pratique, STEAMER a parfois besoin de plus d'une minute pour certaines actions !

C'est pourquoi STEAMER a tout d'abord été réécrit en LAP «de base», mais en exploitant l'héritage multiple comme une solution pour implanter la spécialisation multiple ; les objets évolutifs, eux, sont restés implantés à l'aide d'une solution proche des suggestions de la section précédente.
Cette solution a naturellement imposé aux concepteurs une attention particulière concernant les noms des propriétés et les techniques de parcours du graphe d'héritage utilisées par LAP. La réactivité du système obtenu a été améliorée par cette transformation, mais la tâche des concepteurs a été considérablement alourdie.

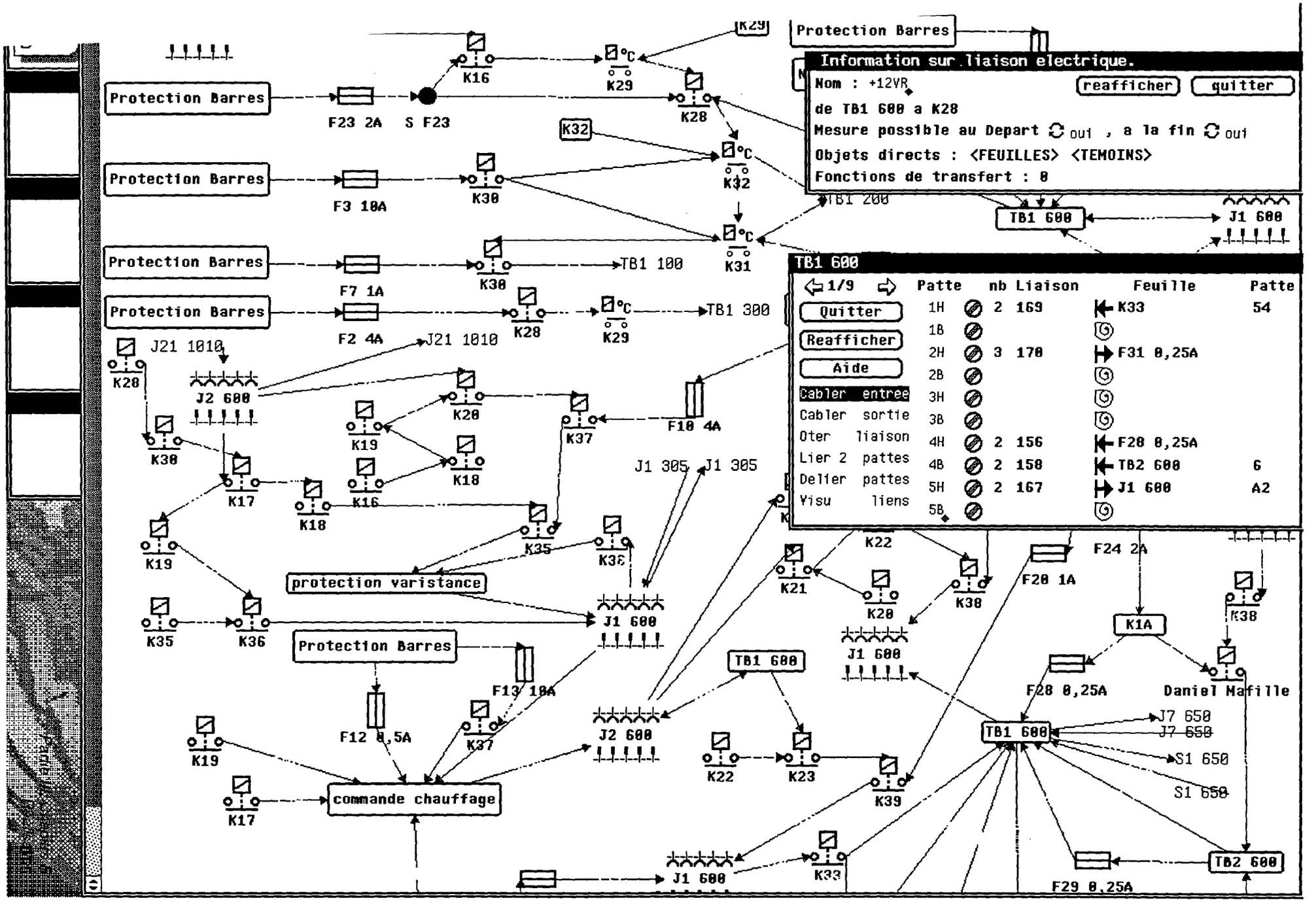


Figure 51 : Un écran de Grace

Chapitre C
Conclusion sur STEAMER

STEAMER a été intégralement conçu à l'aide du modèle de SECOOSSE, appliqué par le biais de l'outil SEISME. Son implantation à l'aide du langage LAP, sur stations de travail Sun, a permis de mettre en oeuvre et de valider un grand nombre de théories sur le raisonnement de diagnostic. Le prototype réalisé aurait dû être validé par une implantation à l'intérieur d'un centre d'émission. Cependant, la conception et la réalisation de STEAMER à l'aide de SECOOSSE ont permis au laboratoire T.M.I. de prouver son savoir-faire dans le domaine des «machines intelligentes», ainsi que l'intérêt et la faisabilité de produits de Maintenance Assistée par Ordinateur, dans le contexte des équipements de radiodiffusion. C'est pourquoi le thème d'étude portant sur l'implantation d'une méthodologie de diagnostic a été relayé par des recherches connexes sur l'assistance à la maintenance fondée sur l'aspect visuel autant que sur un raisonnement d'expert.

Conclusion

Sur la thèse qu'un domaine de connaissance spécifique -le diagnostic d'équipements électromécaniques- nécessite un environnement spécifique, ce travail présente une méthode de conception spécifiquement destinée aux systèmes experts de maintenance de seconde génération.

I. Nature des propositions

Le modèle de la méthode repose sur le modèle orienté objets, auquel ont été ajoutés des concepts d'abstraction nouveaux, permettant d'exprimer davantage de sémantique : ainsi, les notions de factorisation et de spécialisation sont issues du concept d'héritage mais sont formellement définies et exploitées dans la méthode ; de même, le concept de spécialisation multiple autorise plusieurs concepteurs à oeuvrer simultanément et indépendamment sur une représentation unique de la réalité ; enfin, la notion d'objet évolutif simplifie considérablement le travail de conception en permettant de formaliser les objets dans les différentes étapes de leur vie. Le modèle proposé respecte toujours le principe d'encapsulation, tant à l'échelle de l'objet qu'à l'échelle du schéma conceptuel, dans la mesure où les schémas conceptuels établis par plusieurs concepteurs sont indépendants et ne communiquent que grâce à des propriétés explicitement publiées par les concepteurs.

L'utilisation du langage IA, issu de la méthode NIAM dédiée au modèle Entité-Relation, montre que l'approche relationnelle et l'approche orientée objets nécessitent toutes deux un langage graphique clair ; de plus, l'application de ce langage graphique au modèle orienté objets met en évidence des similitudes conceptuelles entre NIAM et SECOOSSE.

La démarche proposée permet de systématiser l'extraction des concepts à partir d'entretiens avec les intervenants experts et fournit, par l'intermédiaire d'une mise en forme en langue naturelle, puis d'un algorithme automatisable, un ensemble de classes, leurs possibilités d'évolution et les relations de spécialisation qui les lient.

Enfin, un outil dédié à la méthode permet de créer et de maintenir les schémas conceptuels.

La méthode de conception proposée à partir de ces quatre composantes a été validée dans un cadre industriel complexe : il s'agit de la formalisation d'un système de maintenance de seconde génération (c'est-à-dire ne faisant pas

intervenir de moteur d'inférence ni de base de règles), appliqué à un émetteur de télévision. Ce système expert a été implanté et fournit des résultats significatifs.

II. Intérêt du travail

En introduisant un nouveau modèle de systèmes informatiques -le modèle orienté objets- au CERLOR, nous avons l'espoir, à terme, d'offrir à tous les chercheurs qui utilisent l'informatique comme un outil, et non comme une fin, la possibilité de concevoir et de réaliser plus simplement leurs logiciels spécifiques. Son application spécifique aux systèmes de maintenance permet désormais d'envisager une rationalisation des études futures sur les sujets connexes vers lesquels se tourne le laboratoire T.M.I.

Ce travail constitue également un point d'entrée vers des thèmes de recherche récents, portant principalement sur la réalisation et l'exploitation de systèmes de gestion de bases de données orientées objets.

III. Perspectives d'avenir

Il existe deux grands axes de recherche en relation avec ce travail.

III.1. Méthodes de conception

De nombreux sujets de recherche connexes peuvent cependant être étudiés, tant dans le domaine de la Maintenance Assistée par Ordinateur qu'en ce qui concerne les méthodes de conception. Ainsi, en ce qui concerne la méthode de conception, une étude pourrait porter sur une automatisation de la mise en forme du cahier des charges par l'application de règles sur le langage naturel et de règles de réécriture. De plus, la démarche peut être étendue pour prendre en compte l'évolution des schémas conceptuels. Enfin, une étude particulière pourrait porter sur la généralisation de la méthode à la conception de logiciels orientés objets dans un contexte plus général que celui des systèmes de maintenance.

III.2. Maintenance Assistée par Ordinateur

Après la conception et la réalisation de STEAMER, le laboratoire T.M.I. engage de nouvelles études sur le thème de la MAO. Ainsi, la faisabilité d'un terminal d'atelier fondé sur une assistance visuelle, faisant intervenir systèmes multimédia et navigation hypertexte, doit désormais être étudiée. En outre, une collaboration semble s'ouvrir entre TéléDiffusion de France et un grand équipementier pour équiper les émetteurs de télévision des prochaines

générations de systèmes de diagnostic intégrés. Parallèlement, une étude fondamentale sur les méthodologies de diagnostic employées par les experts a pris sa source dans STEAMER et va être développée.

Annexes

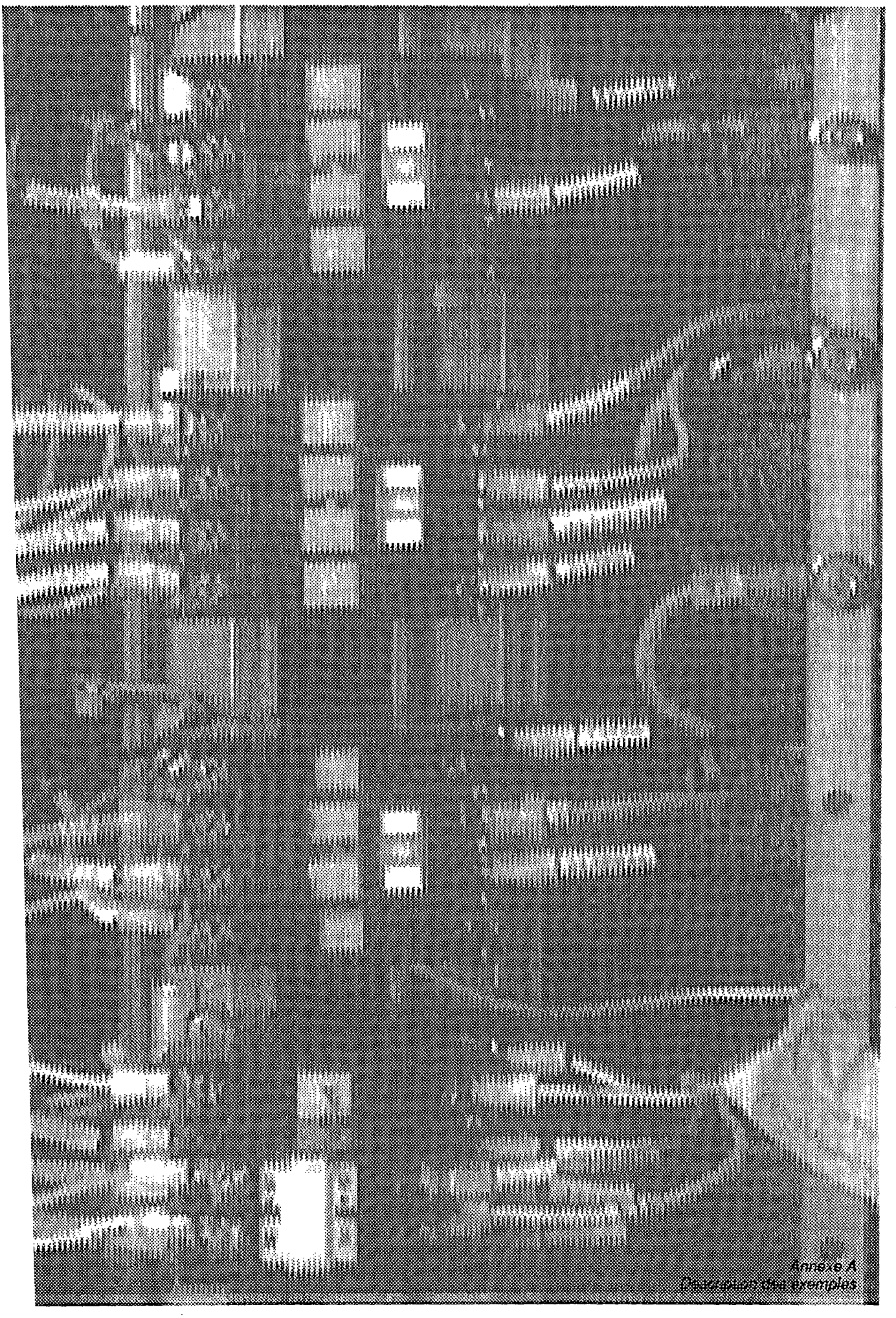
Annexe A
Description des exemples

Un relais est un composant électromécanique. Son rôle est d'établir ou d'interrompre la continuité d'un circuit électrique. Le signal électrique véhiculé par ce circuit peut être de n'importe quelle forme, intensité et tensions, les limites étant imposées par les caractéristiques du relais.
Le circuit est interrompu par des ***contacts***, qui sont en quelque sorte des interrupteurs à l'intérieur du relais. La position de ces contacts est commandée par un axe métallique. Cet axe est actionné par le champ magnétique issu d'un ***bobinage*** : lorsque la bobine est alimentée, le champ magnétique attire l'axe vers la bobine ; par un principe de levier, les contacts sont alors fermés. Réciproquement, lorsque le champ magnétique cesse, un système de ressorts rappelle l'axe vers sa position initiale, donc les contacts s'ouvrent.
L'alimentation de la bobine est indépendante du circuit coupé ; notamment, une bobine ne nécessite pas des courants importants. Par contre, le circuit coupé peut véhiculer de fortes intensités.
L'intérêt d'un relais est donc de commander un signal, quelconque, à l'aide d'un second signal indépendant.

Evoquons également des types particuliers de relais, tels ceux dont les contacts sont *fermés* au repos et ouverts en travail, ainsi que les *bistables*, possédant deux bobines qui doivent être alimentées alternativement pour que les positions des contacts soient modifiées.

Mentionnons enfin les différentes formes que peut prendre un relais, allant de la présentation "circuit intégré" (micro-relais) à l'entité cubique de dix centimètres de côté.

Figure 52 : Un panneau de quatre relais



Annexe B
Références bibliographiques

[ABITEBOUL 90]
Virtuality in Object-Oriented Databases
Serge ABITEBOUL
Actes des 6èmes journées du PRC «Bases de Données Avancées», Montpellier, 1990, p 17-32

[AFNOR 88]
fiabilité, maintenabilité, disponibilité. Recueil des normes françaises
AFNOR-UTE
Livre, éditeur AFNOR, 1988

[ALAGIC 89]
Object oriented versus classical database methods and tools : a case study
Suad ALAGIC, Miroslav KANDIC, Nadira KRDZALIC
Actes des 3èmes journées «Pratique des méthodes et outils logiciels d'aide à la conception de systèmes d'information» (LIANA), Nantes, 1989, p 255-269

[AUBERT 89]
SPOKE
Jean-Pascal AUBERT, Patrick DIXNEUF
Article de Génie Logiciel & Systèmes Experts N° 17, Décembre 1989

[AULD 88]
Programming plus points
William AULD
Article de Systems International Vol. 16 N° 2, Décembre 1988

[AULD 89]
That object of design
William AULD
Article de Systems International Vol. 17 N° 6, Juin 1989

[BARBIER 89]
Glossaire des systèmes à objets : application à l'informatisation de la fonction production d'une entreprise manufacturière
Franck BARBIER, Alain HAURAT
Article de Génie Logiciel & Systèmes Experts N° 17, Décembre 1989

[BARTHES 90]
Objets et Intelligence Artificielle
Jean-Paul BARTHES, Jacques FERBER, Paul-Yves GLOESS, Anne NICOLLE, Philippe VOLLE
Actes des 3èmes journées du PRC «Intelligence Artificielle», Paris, 1990 (Ed. HERMES)

[BAZEX 89]
L'inférence dans les bases de données relationnelles : approche Prolog/approche relationnelle ?
P. BAZEX, A. HAMEURLAIN, W. BENNANI, R. JOACHIM
Actes de la 1ère conférence européenne «techniques et applications de l'intelligence artificielle en milieu industriel et de service» (CONVENTION IA), Paris, 1989, p 11-25 (Ed. HERMES)

[BIRAMA 89]
Analyse comparative de quelques méthodes de conception de systèmes d'informations. Définition d'un cadre de comparaison.
Ndong BIRAMA
Mémoire de recherches doctorales, CRIN (Université de Nancy I), Janvier 1989

[BLAIR 89]
Genericity vs Inheritance vs Delegation vs Conformance vs ...
Gordon S. BLAIR, John J. GALLAGHER, Javad MALIK
Article de JOOP Vol 2 No 3, Septembre/Octobre 1989

[BODART 85]
IDA (Interactive Design Approach) : une méthode de conception assistée de SI.
F. BODART
Actes du 7ème séminaire Tuniso-Français d'informatique, Tunis, 1985

[BOIS 89]
Spécification d'une méthode fondée sur le concept d'objets
Huguette BOIS
Actes des 3èmes journées «Pratique des méthodes et outils logiciels d'aide à la conception de systèmes d'information» (LIANA), Nantes, 1989, p 402-422

[BOOCH 86]
Object-Oriented Development
Grady BOOCH
Article de IEEE Transactions on Software Engineering, Vol. SE-12, N° 2, Février 86

[BOULANGER 87]
SEIVE : système expert en integration de vues externes
Louis BOULANGER, Yvon MARIE-SAINTE
Actes de Convention informatique, Barcelone, 1987

[BOUZEGHOUB 86]
SECSI : un outil interactif de modélisation conceptuelle de bases de données
M. BOUZEGHOUB, G. GARDARIN, E. METAIS
Actes Nouvelles perspectives des BD, 1986 (Ed. EYROLLES)

[BRODIE 81]
Final report of the ANSI/X3/SPARC DBS - SG (relational database task group)
Michael L. BRODIE, Joachim W. SCHMIDT
Doc. No. SPARC-81-690

[BRODIE 84]
On conceptual modelling - Perspectives from Artificial Intelligence, Databases, and Programming Languages
Michael L. BRODIE, John MYLOPOULOS, Joachim W. SCHMIDT
Livre, éditeur Springer-Verlag, 1984

[BURNEAU 89]
Etude de la Modélisation Orientée Objets dans le contexte des Systèmes Experts en Maintenance
Jean-Christophe BURNEAU
Rapport de DEA, CRIN (Université de Nancy I), Juin 1989

[BURNEAU 91a]
Une méthode de conception orientée objets dans le contexte des systèmes experts de maintenance
Jean-Christophe BURNEAU, Odile THIERY
Actes du congrès «Informatique des organisations et systèmes d'information et de décision» (INFORSID), Paris, Juin 1991.

[BURNEAU 91b]
Object oriented databases for Maintenance Expert Systems
Jean-Christophe BURNEAU, Odile THIERY
Actes du congrès «Databases and expert systems» (DEXA), Berlin, Août 1991.

[BURNEAU 91c]
A procedure to classify evolving objects in object oriented knowledge bases
Jean-Christophe BURNEAU, Odile THIERY
Article de Computer Systems Science and Engineering (special issue on object oriented systems). To be published.

[BYTE 89]
Object-Oriented Programming
Articles de BYTE, Vol. 14 N°3, Mars 1989

[CARDELLI 84]
A semantic of multiple inheritance
Luca CARDELLI
Lecture Notes in Computer Science, Vol. 173, p 51-67, Springer Verlag, 1984

[CARRE 89]
Méthodologie orientée objet pour la représentation des connaissances. Concepts de point de vue, de représentation multiple et évolutivité d'objet
Bernard CARRE
Thèse, Université des Sciences et Techniques de LILLE-FLANDRES-ARTOIS, Janvier 1989

[CARRE 90a]
Multiple and Evolutive Representation in the ROME language ; towards an integrated Corporate Information System
Bernard CARRE, Lenneke DEKKER, Jean-Marc GEIB
Actes du congrès «Technology of Object-Oriented Languages and Systems» (TOOLS), Paris, 1990, p 101-109

[CARRE 90b]
The Point of View notion for Multiple Inheritance
Bernard CARRE, Jean-Marc GEIB
Actes de «European Conference on Object Oriented Programing» (ECOOP), 1990

[CASTELLANI 91]
Le modèle de la méthode MCO d'analyse et de conception des systèmes d'objets
Xavier CASTELLANI
Actes du congrès «Informatique des organisations et systèmes d'information et de décision» (INFORSID), Paris, Juin 1991, p 395-429

[CAUVET 87]
Une base de règles pour la modélisation des aspects dynamiques des systèmes d'information
Corinne CAUVET, Christophe PROIX, Dominique TRECOURT
Actes du congrès «Des bases de données aux bases de connaissance», Nice, 1987

[CAUVET 89a]
La conception de Bases de Données orientées objets : modèle et méthode (O*)
Corinne CAUVET, Christophe PROIX
Actes des 5èmes journées du PRC «Bases de Données Avancées», Genève, 1989, p 23-47

[CAUVET 89b]
O* : un modèle pour la conception de Bases de Données orientées objets
Corinne CAUVET, Colette ROLLAND
Actes du congrès «Informatique des organisations et systèmes d'information et de décision» (INFORSID), Nancy, 1989, p 265-284

[CAUVET 90]
Représentation des connaissances pour la conception des systèmes d'information
C. CAUVET, G. GROSZ, C. PROIX
in Nouvelles perspectives des Systèmes d'Informations, Eyrolles, 1990 (textes recueillis par André FLORY et Colette ROLLAND)

[CHEN 76]
The entity-relationship model : toward a unified view of data
Peter P. CHEN
Article de ACM Transactions on Database Systems Vol. 1, N° 1, 1976

[CHEVAL 89]
Une extension aux modèles orientés objets : règles d'évolution
J.L. CHEVAL
Actes du congrès «Informatique des organisations et systèmes d'information et de décision» (INFORSID), Nancy, 1989, p 339-358

[CODD 70]
A relational Model of Data for Large Shared Data Banks
E.F. CODD
Article de Communication of the ACM Vol. 13 N° 6, 1970

[CODD 79]
Extending the Data Base Relational Model to capture more meaning
E.F. CODD
Article de ACM Transactions on Database Systems Vol. 4, N°4, 1979

[COLNET 89]
Prototypage, programmation objet : application à l'édition structurelle
Dominique COLNET
Thèse, CRIN (Université de Nancy I), Février 1989

[COMYN-WATTIAU 89]
La conception incrémentale : une application pratique de l'intégration de vues
Isabelle COMYN-WATTIAU, Elisabeth METAIS
Actes des 5èmes journées du PRC «Bases de Données Avancées», Genève, 1989

[COURTIN 88]
Failure diagnosing expert system for the TDF-1 satellite's power supply system and Expert-Draw graphic interface
J.-P. COURTIN, A. THUAIRE, J.-M. LEMOINE, A. MONASTEROLO
Actes du colloque international «Interaction Homme-Machine et IA dans les domaines de l'aéronautique et de l'espace», Toulouse, 1988, p 229-239

[COX 86]
Object oriented programming : an evolutionary approach
Brad J. COX
Livre, éditeur Addison-Wesley Publishing Company, 1986,

[DEKLEER 83]
The origin, form and logic of qualitative physical laws
J. de KLEER, J.S. BROWN
Actes de la 8ème «International Joint Conference on Artificial Intelligence», Karlsruhe (RFA), 1983

[DESICY 89]
Mariage système expert et base de données : naissance d'une nouvelle représentation de la connaissance
A.-N. DESICY
Actes de la 1ère conférence européenne «techniques et applications de l'intelligence artificielle en milieu industriel et de service» (CONVENTION IA), Paris, 1989, p 89-93

[DUCOURNAU 89]
La multiplicité de l'héritage dans les langages à objets
Roland DUCOURNAU, Michel HABIB
Article de TSI n° 1, 1989

[DUCRIN 84]
Programmation
Amédée DUCRIN
Livre, Ed. DUNOD, 1984

[DUGERDIL 88]
Les mécanismes d'héritage d'OBJLOG
Philippe DUGERDIL
Actes des journées «Mardis Objets», CRIN (Nancy), 10 Mai 1988, p 185-202

[FILLOQUE 89]
La méthode Jackson (J.S.D)
Jean-Marie FILLOQUE
Actes des 3èmes journées «Pratique des méthodes et outils logiciels d'aide à la conception de systèmes d'information» (LIANA), Nantes, 1989, p 35-73

[FINANCE 79]
Etude de la construction de programmes : méthodes et langages de spécification et de résolution de problèmes
Jean-Pierre FINANCE
Thèse d'état, CRIN (Université de Nancy I), 1979

[FLORY 89]
Toward a new generation of design tools : some basic principles proposal
André FLORY, José MOREJON, Arnold ROCHFELD
Actes de la 1ère conférence «Technology of Object-Oriented Languages and Systems» (TOOLS), Paris, 1989, p 437-448

[FORBUS 84]
Qualitative Process Theory
Kenneth D. FORBUS
Article de Artificial Intelligence, Vol. 24, 1984, p 85-168

[FOUCAUT 82]
Modèle et outil pour la conception des systèmes d'information dans les organisations - Projet REMORA
Odile FOUCAUT
Thèse d'Etat, CRIN (Université de Nancy I), Juin 82

[GALLAIRE 85]
La représentation des connaissances
Hervé GALLAIRE
Article de La Recherche N° 170, Octobre 1985

[GLASSEY 89]
Conceptual Design of a Software Object Library for Simulation of Semiconductor Manufacturing System
C. Roger GLASSEY, Sadashiv ADIGA
Article de JOOP, Novembre/Décembre 1989

[GIBSON 90]
Objects - Born and Bred
Elizabeth GIBSON
Article de Byte, Octobre 1990

[HABRIAS 88]
Le modèle relationnel binaire - Modèle I.A. (NIAM)
Henri HABRIAS
Livre, éditeur Eyrolles, 1988

[HABRIAS 89b]
Le schéma conceptuel ? Non ! Les schémas conceptuels ou vers l'approche orientée objets
Henri HABRIAS
Actes des 3èmes journées «Pratique des méthodes et outils logiciels d'aide à la conception de systèmes d'information» (LIANA), Nantes, 1989, p 423-437

[HABRIAS 90]
Des ronds et des flèches, petit traité de télépathie à l'usage des concepteurs
Henri HABRIAS
Article de TRACE (Ed. CERTA-CRDP Dijon) n° 1, p 11-20

[HABRIAS 91a]
Attention au discours ! NIAM, l'informatique et les mathématiques
Henri HABRIAS
in Introduction à la Spécification, IUT de Nantes, 1991

[HABRIAS 91b]
La méthode MOON : Méthode Orientée Objets Normalisés
Henri HABRIAS
Actes de CIL, Barcelone, 1991

[HEITZ 87]
HOOD, une méthode de conception hiérarchisée orientée objets pour le développement de gros logiciels techniques et temps réel
Maurice HEITZ
Article de BIGRE N° 57, p 42-61, Décembre 87

[HULL 87]
Semantic DataBase Modeling : survey, applications and research issues
Richard HULL, Roger KING
Article de ACM Computing Surveys Vol. 19 N° 3, septembre 1987

[IGL 88]
S.A.D.T.
Document IGL
Actes des 2èmes journées «Pratique des méthodes et outils logiciels d'aide à la conception de systèmes d'information» (LIANA), Nantes, 1988, p 42-91

[INFORSID 76]
Colloque sur la recherche en informatique des organisations
Publication IRIA, Caen, 1976

[JACKSON 89]
JSD and the object oriented approach
Michael JACKSON
Actes des 3èmes journées «Pratique des méthodes et outils logiciels d'aide à la conception de systèmes d'information» (LIANA), Nantes, 1989, p 1-11

[JAULENT 90]
Génie Logiciel : les méthodes
Patrick JAULENT
Livre, éditeur Armand Colin, 1990

[JOCHEM 89]
An Object-Oriented Analysis and Design Methodology for Computer Integrated Manufacturing Systems
Roland JOCHEM, Markus RABE, Wolfram SÜSSENGUTH, Peter BALS
Actes de la 1ère conférence «Technology of Object-Oriented Languages and Systems» (TOOLS), Paris, 1989, p 75-84

[KUIPERS 86]
Qualitative Simulation
Benjamin KUIPERS
Article de Artificial Intelligence Vol. 29, 1986, p 289-338

[LAI 89]
HyperHood ++ : an object oriented design tool for developments in object oriented programming languages
Michel LAI
Actes de la 1ère conférence «Technology of Object-Oriented Languages and Systems» (TOOLS), Paris, 1989, p 295-308

[LALANDA 89]
Intérêt de l'abstraction d'alternatives pour l'évolutivité des bases de données orientées objets
Philippe LALANDA
Rapport de DEA, CRIN (Université de Nancy I), Juin 1989

[LAP 89]
LAP users'manual (preliminary version) v3.0
LAP reference manual
Document ELSA Software

[LE 82]
La méthode MERISE
Frédéric LE, Yves TABOURIER, Hubert TARDIEU
Actes de la 3ème conférence «Comparative Review of Information System design methodologies - International federation for Information Processing» (CRIS-IFIP, Working Group 8.1), Amsterdam (Hollande), 1986

[LE 85]
La méthode MERISE
F. LE, O. de BELER
Actes du 7ème séminaire Tuniso-Français d'informatique, Tunis, 1985

[LEONARD 89]
Conception et représentation des objets à comportement complexe
Daniel LEONARD
Actes du congrès «Informatique des organisations et systèmes d'information et de décision» (INFORSID), Nancy, 1989, p 285-298

[LISSANDRE 90]
Architectes et promoteurs : panorama des méthodes de conception
Michel LISSANDRE
Article de Minis & Micros N°333, Janvier 1990

[LIVERCY 78]
Théorie des programmes ; schémas, preuves, sémantique
C. LIVERCY
Livre, Ed. DUNOD Informatique, 1978

[LOTT 90]
Conception orientée par les objets : des méthodes encore adolescentes
Michel LOTT
Article du Monde Informatique, Juin 1990

[MAFILLE 89]
Rapport interne de cognitique
Daniel MAFILLE
Rapport interne TéléDiffusion de France CL/FMS/DM/036/89

[MASINI 89]
Les langages à objets : langages de classes, langages de frames, langages d'acteurs
Gérald MASINI, Amedeo NAPOLI, Dominique COLNET, Daniel LEONARD, Karl TOMBRE
Livre, éditeur InterEditions, 1989

[MEYER 84]
Méthodes de programmation
B. MEYER, C. BAUDOIN
Livre, éditeur EYROLLES, 1984

[MUENIER 90]
Enquête Génie Logiciel
Michel MUENIER
Article de AFCET/INTERFACES N°96, octobre 1990

[NGUYEN 89]
Schema evolution in object-oriented database systems
Gia Toan NGUYEN, Dominique RIEU
Article de Data & Knowledge Engineering, Vol. 4 N°1, Juillet 1989

[NGUYEN 91]
Databases issues in object-oriented design
Gia Toan NGUYEN, Dominique RIEU
Actes de la 4ère conférence «Technology of Object-Oriented Languages and Systems» (TOOLS), Paris, 1991

[NIJSSEN 77]
Architecture and models in database management systems
G.M. NIJSSEN
Actes de la conférence «Comparative Review of Information System design methodologies - International federation for Information Processing» (CRIS-IFIP, TC2), Amsterdam (Hollande), 1977

[OLLE 90]
Méthodologies pour les systèmes d'information
T.W. OLLE, J. HAGELSTEIN, I.G. MACDONALD, C. ROLLAND, H.G. SOL, F.J.M. VAN ASSCHE, A.A. VERRIJN-STUART
Livre, éditeur DUNOD Informatique, 1990

[PAROT 88]
La méthode J.S.D. et les outils associés
J.M. PAROT
Actes des 3èmes journées «Pratique des méthodes et outils logiciels d'aide à la conception de systèmes d'information» (LIANA), Nantes, 1988, p 260-273

[PELLAUMAIL 85]
La méthode AXIAL
Philippe PELLAUMAIL
Actes du 7ème séminaire Tuniso-Français d'informatique, Tunis, 1985

[PELLAUMAIL 86]
La méthode AXIAL
Philippe PELLAUMAIL
Livre, éditeur Les éditions d'organisation, 1986

[PENNEY 87]
Class modification in GemStone object-oriented DBMS
D.J. PENNEY, J. STEIN
Actes de la conférence «Object Oriented Programing Systems, Languages and Applications (OOPSLA), Orlando (Floride), 1987

[PERNICI 90]
Requirements specifications for object-oriented systems
B. PERNICI
in Nouvelles perspectives des Systèmes d'Information, Eyrolles, 1990 (textes présentés par André FLORY et Colette ROLLAND)

[RIEU 91]
Classification et représentation d'objets
Dominique RIEU, A. CULET
Actes du congrès «Informatique des organisations et systèmes d'information et de décision» (INFORSID), Paris, 1991, p 85-107

[ROCHE 88]
Un logiciel d'aide à la conception des systèmes d'information : LACSI
Alain ROCHE
Actes du congrès «Informatique des organisations et systèmes d'information et de décision» (INFORSID), La Rochelle, 1988

[ROCHE 89]
LACSI : un logiciel d'aide à la conception de systèmes d'informations
Alain ROCHE
Actes du congrès «Informatique des organisations et systèmes d'information et de décision» (INFORSID), Nancy, 1989, p 617-623

[ROCHFELD 89a]
L'univers des méthodes de conception
Arnold ROCHFELD
Actes du congrès «Informatique des organisations et systèmes d'information et de décision» (INFORSID), Nancy, 1989, p 477-514

[ROCHFELD 89b]
Vers une nouvelle génération d'outils de conception. Quelques principes de base.
Arnold ROCHFELD, José MOREJON, André FLORY
Actes du congrès «Informatique des organisations et systèmes d'information et de décision» (INFORSID), Nancy, 1989, p 515-533

[ROGERS SAXON 87]
AI-based diagnostic reasoning : an overview of WHAT, HOW and WHY
ROGERS SAXON C.
Actes de la 2ème conférence «AI & Advanced Computer Technology», Long Beach (Californie), 1987

[ROLLAND 82]
The REMORA methodology for information systems design and management
Colette ROLLAND, Christian RICHARD
Actes de la 1ère conférence «Comparative Review of Information System design methodologies - International federation for Information Processing» (CRIS-IFIP, Working Group 8.1), Amsterdam (Hollande), 1982, p 369-426

[ROLLAND 83]
Database Dynamics
Colette ROLLAND
Article de DATA BASE, printemps 1983

[ROLLAND 85]
REMORA : une méthode de conception des systèmes d'informations
Colette ROLLAND
Actes du 7ème séminaire Tuniso-Français d'informatique, Tunis, 1985

[ROLLAND 87]
Conception de Systèmes d'informations ; la méthode REMORA
Colette ROLLAND, Odile FOUCAUT, G. BENCI
Livre, éditeur Eyrolles, 1987

[ROSEN 90]
Pour une définition méthodologique de la notion d'«orienté objet»
Jean-Pierre ROSEN
Article de AFCET/Interfaces N°96, Octobre 1990

[SMITH 77a]
Database Abstractions : Aggregation
John Miles SMITH, Diane C.P. SMITH
Article de Communications of the ACM, Juin 1977, Vol. 20 N°6

[SMITH 77b]
Database Abstractions : Aggregation and Generalization
John Miles SMITH, Diane C.P. SMITH
Article de ACM Transactions on Database Systems, Vol. 2 N°2, Juin 1977

[SPACCAPIETRA 90]
Intégration de vues et relativisme sémantique
Stefano SPACCAPIETRA, Christine PARENT
Actes des 6èmes journées du PRC «Bases de Données Avancées», Montpellier, 1990

[TABOURIER 86]
De l'autre côté de MERISE : systèmes d'information et modèles d'entreprise
Yves TABOURIER
Livre, éditeur Les Editions d'Organisation, 1986

[TEOREY 82]
Design of data base structures
T.J. TEOREY, J.P. FRY
Livre, éditeur Prentice Hall, 1982

[THIERY 85]
LASSIF : LAngage de Spécification des Systèmes d'Informations et Logiciel d'Aide à la Spécification des Systèmes d'Informations
Odile THIERY
Thèse d'Etat, CRIN (Université de Nancy I), Juin 85

[THIERY 86]
Un environnement de méthodes et d'outils pour la conception de Systèmes d'Information en informatique de gestion
Odile THIERY, Jacques THEVENOT
Actes du 1er congrès «Génie Industriel», Paris, 1986

[TROYER 89]
RIDL* : un outil de génie logiciel pour la conception de bases de données en présence de contraintes d'intégrité
O. de TROYER, R. MEERSMMANN
Actes du congrès «Informatique des organisations et systèmes d'information et de décision» (INFORSID), Nancy, 1989, p 634-654

[VERMEIR 82]
A procedure to define the object type structure of a conceptual schema
D. VERMEIR, G.M. NIJSSEN
Article de Information Systems Vol. 7 N°4 p 329-336, 1982

[VERHEIJEN 82]
NIAM : an information analysis method
G.M.A. VERHEIJEN, J. van BEKKUM
Actes de la 1ère conférence «Comparative Review of Information System design methodologies - International federation for Information Processing» (CRIS-IFIP, Working Group 8.1), Amsterdam (Hollande), 1982

[VISSER 88]
Expert software design strategies
Willemien VISSER, Jean-Michel HOC
Rapport INRIA, version finale d'un chapitre de "Psychology of programming", London:Academic Press, J.M. Hoc, T. Green, R. Samurçay (Eds.)

[WALLER 91]
Schema Updates an Consistency
Emmanuel WALLER
Actes des 7èmes journées du PRC «Bases de Données Avancées», Lyon, 1991

[WASSERMAN 89]
Concepts of Object-Oriented Structured Design
Anthony I. WASSERMAN, Peter A. PIRCHER, Robert J. MULLER
Actes de la 1ère conférence «Technology of Object-Oriented Languages and Systems» (TOOLS), Paris, 1989

[WASSERMAN 90]
The Object-Oriented Structured Design Notation for Software Design Representation
Anthony I. WASSERMAN, Peter A. PIRCHER, Robert J. MULLER
Article de COMPUTER, p 50-63, Mars 1990

[WEGNER]
The object classification paradigm
Peter WEGNER
in Perspectives on Object-Oriented programming

[WEGNER 87]
Dimensions of Object-Based Language Design
Peter WEGNER
Actes de la conférence «Object Oriented Programing Systems, Languages and Applications (OOPSLA), Orlando (Floride), 1987, p 168-182

[WEGNER 88]
Inheritance as an incremental classification mechanism or what like is and isn't like
Peter WEGNER, Stanley B. ZDONIK
Actes de «European Conference on Object Oriented Programing) (ECOOP), Oslo (Norvège), 1988

[WODA 90]
STEAMER : SysTème Expert d'Aide à la Maintenance des Equipements de Radiodiffusion
Pierre WODA, Jean-Christophe BURNEAU
Actes des 10èmes journées «Les systèmes experts et leurs applications», Avignon, 1990, p 251-263

UNIVERSITE DE NANCY I

NOM DE L'ETUDIANT : Monsieur BURNEAU Jean-Christophe

NATURE DE LA THESE : DOCTORAT DE L'UNIVERSITE DE NANCY I
en INFORMATIQUE

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le 22 NOV. 1991 n° 526

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

UNIVERSITE NANCY I
LE PRESIDENT

M. BOULANGÉ

Une méthode de conception orientée objets appliquée aux systèmes de maintenance d'équipements de radiodiffusion

Résumé

Cette thèse propose une méthode de conception destinée à la création de systèmes de maintenance d'équipements de radiodiffusion. Le modèle de la méthode consiste en une extension du modèle orienté objets, prenant en compte les entités qui évoluent au cours de leur vie et formalisant la notion de points de vue différents sur un même objet. Les schémas conceptuels sont exprimés à l'aide d'un langage graphique qui permet l'expression de tous les concepts et des contraintes associées. Une démarche permet d'automatiser le calcul des classes à partir d'un énoncé en langage naturel. Un outil supporte la méthode, autorisant la gestion de schémas conceptuels nombreux et importants.

La première partie présente les contraintes auxquelles doit répondre une méthode de conception dans le contexte de la maintenance, puis analyse six méthodes existantes pour aboutir sur la nécessité de proposer une méthode spécifique.

La seconde partie décrit chaque élément de la méthode proposée (modèle, langage, démarche, outil) et propose une validation de la méthode par elle-même.

La troisième partie de ce rapport présente une application de maintenance d'équipements de radiodiffusion entièrement conçue à l'aide de la méthode proposée.

Mots-clés

Modèle orienté objets - Méthode de conception orientée objets - Logiciel d'aide à la conception - Maintenance Assistée par Ordinateur - Equipements de radiodiffusion