Développement de systèmes d'information à l'aide de patrons. Application aux bases de données actives

Agnès Front

Thèse

présentée par

Agnès FRONT

pour obtenir le titre de Docteur
de l'Université Joseph Fourier - Grenoble 1
(arrêtés ministériels du 5 juillet 1984 et du 30 Mars 1992)
Spécialité : **Informatique**

Développement de systèmes d'information à l'aide de patrons
Application aux bases de données actives

Date de soutenance : 13 décembre 1997

Composition du jury :

Président :	Paul Jacquet
Rapporteurs :	Colette Rolland
	Michel Léonard
Examinateurs :	Christine Collet
	Daniel Pagonis
Directeurs :	Jean-Pierre Giraudin
	Mauricio Lopez

Thèse préparée au sein du
Laboratoire Logiciels, Systèmes, Réseaux – IMAG

à Papa...

Je ne t'oublierai jamais...

Remerciements

Mes premiers remerciements vont aux membres du jury :

M. Paul Jacquet, Professeur à l'Institut National Polytechnique de Grenoble et directeur du laboratoire LSR, pour l'honneur qu'il me fait de présider ce jury et pour m'avoir accueillie au sein de son laboratoire.

M^{me} Colette Rolland, Professeur à l'Université de Paris 1 Sorbonne que j'admire pour les nombreuses recherches novatrices qu'elle effectue dans le domaine des systèmes d'information et qui me fait le grand honneur d'apporter son jugement sur ce travail.

M. Michel Léonard, Professeur à l'Université de Genève, d'avoir bien voulu apporter son jugement sur cette thèse. Je tiens à lui exprimer toute ma gratitude pour l'intérêt qu'il porte à mon travail depuis notre première rencontre lors d'une journée Jeunes Chercheurs à Archamps.

M^{me} Christine Collet, Maître de Conférences à l'Université Joseph Fourier et co-responsable de l'équipe STORM, qui me fait l'honneur de participer à ce jury. Je la remercie également pour l'aide, les conseils et la confiance qu'elle m'a apportés.

M. Daniel Pagonis, Praticien hospitalier au Centre Hospitalier Universitaire de Grenoble, pour son aide précieuse lors de mes recherches d'une application réelle. Cette thèse n'aurait certainement pas été la même sans l'application médico-technique.

M. Mauricio Lopez, Ingénieur Bull, qui m'a encadrée lors de mon DEA et qui m'a permis de poursuivre en thèse. Je lui suis reconnaissante de m'avoir donné accès à l'ensemble des documents relatifs au projet Esprit III IDEA, tant du point de vue des applications que de celui de la méthodologie.

et surtout M. Jean-Pierre Giraudin, Professeur à l'Université Pierre-Mendès France, pour la disponibilité, les conseils, l'aide et la gentillesse dont il a fait preuve tout au long de ces quatre ans... Je ne saurai jamais lui exprimer toute ma gratitude pour avoir été non seulement mon directeur de thèse, mais aussi une personne sur laquelle j'ai toujours pu compter.

Messieurs Mauricio Lopez et Jean-Pierre Giraudin sont à l'origine de cette thèse. Je les remercie vivement de m'avoir initiée aux mondes de la conception de systèmes d'information et des bases de données et d'avoir su me transmettre un peu de leur savoir-faire.

Outre les membres du jury, mes remerciements vont à mes collègues de travail de ces quatre années :

M. Michel Adiba, professeur à l'Université Joseph Fourier et co-responsable de l'équipe STORM, qui m'a permis d'effectuer ma thèse dans de bonnes conditions à l'intérieur de l'équipe.

Les membres de l'axe "Conception de Bases de Données" de l'équipe STORM qui m'ont accompagnée lors de cette thèse. Je remercie plus particulièrement José pour son aide précieuse concernant les problèmes techniques GraphTalk en échange de quelques corrections de franco-brésilien, Dominique pour nos échanges passés et à venir sur les patrons de conception et Monique pour ses conseils et sa gentillesse.

Les membres de l'équipe STORM pour leur soutien, en particulier Claudia pour son aide sur le chapitre 6 de ce document, Thierry pour les questions techniques concernant NAOS et Pierre-Claude Scholl pour ses encouragements et ses remarques pertinentes qui m'ont beaucoup aidée à mieux clarifier les différents niveaux d'abstraction.

L'équipe administrative du laboratoire LSR, l'équipe technique de l'ENSIMAG et le service Reprographie de l'institut IMAG, pour la disponibilité et l'efficacité dont ils font preuve tous les jours.

Jeff, Helena, Marie-Claude et tous les autres pour les pauses...

Et surtout Françoise et Rafael, mes "co-locataires de bureau", qui m'ont supportée ces dernières années et n'en sont pas moins devenus des amis.

Enfin, je n'oublierai pas mes amis et ma famille, en particulier :

La famille Conte-Marçot-Gatineau, ma belle-famille "de dans 6 mois", pour sa gentillesse et son soutien...

Mes frères, Michel et Joël, pour s'être occupé de moi et m'avoir soutenue depuis ces 16 années dans ma conquête vers le monde adulte et les soucis qui en résultent...

Maman, pour l'amour qu'elle me porte et la fierté qu'elle éprouve envers moi. N'oublie pas que je serai moi aussi toujours là pour toi...

Enfin, Rémy pour l'amour qu'il me témoigne et la patience dont il fait preuve dans la vie de tous les jours. Je crois qu'il n'est pas besoin d'en dire plus... J'espère que tous nos rêves se concrétiseront bientôt...

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte et objectifs de cette thèse

"Comment l'unité médico-technique d'un hôpital doit-elle réagir face à une demande d'acte de la part d'un patient ou d'un prescripteur?" "Que doit faire le système de commande et de contrôle d'un navire militaire lors de l'apparition d'une nouvelle menace pour le navire?" "Quelles sont les contraintes à vérifier lors de l'élaboration du planning du personnel d'une compagnie de trains et que faut-il faire lors de l'absence de l'un des agents?"...

Ces questions et surtout les réponses à ces questions sont essentielles pour une bonne définition du système d'information correspondant et une bonne conception d'un système informatique adapté. Elles représentent des besoins de réaction à des situations significatives apparaissant dans ou en dehors de l'application. La manière d'apporter des réponses à ces questions lors de l'implantation d'une application a évolué en permanence selon les progrès technologiques tant matériels (systèmes centralisés, distribués ou répartis) que logiciels (programmation batch, bases de données, objets, etc.), mais a toujours été en grande partie cachée dans du code, des données ou des programmes d'application. Aujourd'hui, de tels besoins sont mieux pris en compte par les nouvelles technologies. En particulier, la programmation événementielle ou les systèmes de gestion de bases de données actifs offrent des techniques pour implanter plus facilement des réactions à des événements significatifs détectés dans ou hors de l'application.

En parallèle, de tels besoins prennent de plus en plus d'importance dans les cahiers des charges fournis à des concepteurs d'applications. Une analyse plus approfondie de

tels besoins met en évidence que ceux-ci caractérisent essentiellement des besoins de réaction de l'application face à des événements et se situent au niveau de l'expression du comportement et de la dynamique de l'application. Nous appelons par la suite **applications réactives** ces applications dont une part importante des besoins se situe au niveau du comportement.

Apporter des éléments d'aide à l'analyse, à la conception et à l'implantation d'applications réactives est l'objectif global de cette thèse. Il s'agit en effet d'un enjeu économique important mais nécessaire, de disposer de modèles, techniques et méthodes permettant une maîtrise plus complète et efficace de l'ingénierie totale d'un système d'information. C'est d'une manière pragmatique que cette thèse tente d'apporter quelques éléments de solutions à ce problème complexe.

Cet objectif global ne peut être atteint sans une étude approfondie de la façon dont le comportement est actuellement pris en compte tout au long de l'activité de modélisation d'un système d'information, activité qui consiste à aboutir à une solution logicielle fiable et valide à partir du cahier des charges d'une application. Le cadre de cette thèse est par conséquent **la modélisation des besoins comportementaux d'applications réactives**. L'activité de modélisation utilise une représentation du monde réel appelée modèle. Le pouvoir d'expression d'un modèle, sa simplicité et son indépendance à l'égard d'une quelconque implantation en sont les caractéristiques les plus importantes, chaque modèle poursuivant ainsi trois objectifs [MPP96] :

- augmenter le pouvoir d'expression sémantique portée par le modèle,
- diminuer la distance entre la perception du monde réel et sa représentation exprimée dans le modèle,
- augmenter l'indépendance à l'égard des choix d'implantation.

En terme de modèle, la préoccupation de cette thèse correspond tout à fait à ces trois objectifs et s'oriente selon un cadre précis. D'une part, les besoins pris en compte se situent entre une spécification d'une situation réelle et la conception d'une solution technique. D'autre part, l'approche adoptée vise à proposer des techniques plus facilement appréhendables par un utilisateur. Enfin, l'approche a pour but de réutiliser dès le niveau de la représentation des besoins, des techniques couramment utilisées et éprouvées lors de l'analyse.

Ce dernier objectif s'insère dans un domaine actuellement primordial qui découle d'un enjeu économique devenu capital depuis quelques années : la recherche d'une

réutilisation maximale. Avec la notion d'objet, le problème de la réutilisation a bénéficié d'un atout important au niveau de l'implantation et de nos jours, aucun système orienté objet n'est vendu sans une ou plusieurs bibliothèques de classes ou de fonctions réutilisables lors du codage d'une application. Cependant, très peu de techniques sont actuellement proposées pour permettre la réutilisation aux niveaux de l'analyse et de la conception des applications. Une nouvelle approche a été proposée récemment dans ce but : l'approche à base de patrons a pour objectif de décrire avec succès des solutions récurrentes à des problèmes logiciels communs dans un certain contexte et d'aider les gens à réutiliser dès l'expression des besoins d'une application, des pratiques effectivement utilisées et éprouvées [ACM96].

L'enjeu de cette thèse consiste à **justifier, proposer et expérimenter l'utilisation des patrons dans le cadre de la conception d'applications réactives**, depuis l'expression des besoins de l'application jusqu'à une implantation dans un système technologique cible.

Nous adoptons une approche pratique pour mener à bien nos recherches et valider nos propositions. Ainsi, une synthèse des besoins comportementaux d'une dizaine d'applications réactives de domaines différents permet de confronter constamment nos propositions au monde réel. De plus, les méthodes existantes couramment utilisées professionnellement sont étudiées en respect de ces applications selon leur capacité à représenter la notion de comportement. Si l'ensemble de ces applications a toujours été considéré dans le cadre de nos recherches, c'est sur les besoins plus spécifiques d'une application cible (le système d'information de l'unité médico-technique d'un hôpital) que nos propositions d'une approche à base de patrons sont expérimentées. C'est finalement sur un système technologique cible existant, le système de gestion de bases de données actif NAOS que cette même application cible est implantée, dans le double but d'une part de montrer la faisabilité de notre approche, d'autre part de tester l'utilisation d'un système tel que NAOS.

1.2 Vocabulaire du domaine

En informatique, le comportement d'une application est généralement exprimé à l'aide d'un vocabulaire varié. Au niveau de l'analyse et de la conception, on parle de transitions d'**états** déclenchées par des **événements**. Au niveau de l'implantation, différents mécanismes sont utilisés pour représenter le comportement d'une application. En particulier, les systèmes de gestion de bases de données actifs utilisent des règles

actives pour représenter le comportement d'une application sous forme d'**événements** déclenchant des **actions** sous certaines **conditions**.

Tous ces termes explicités dans la thèse dans leur contexte propre sont utilisés dans des systèmes ou à des niveaux d'abstraction différents pour exprimer le comportement d'une application. Ces multiples utilisations impliquent parfois une perte du sens véritable de ces concepts. Dans notre volonté de nous rapprocher du niveau utilisateur et dans un souci de clarification, nous donnons une définition encyclopédique de chacun de ces concepts. Parfois, des définitions provenant d'autres disciplines que l'informatique permettent de regarder avec un œil critique les concepts utilisés dans notre domaine.

Nous mentionnons *en italique* les définitions qui servent à mieux cerner les concepts du domaine de cette thèse.

1. **Evénement**

   D'après le Petit Robert [Rob93], un événement est un fait auquel aboutit une situation, un résultat ; c'est aussi ce qui arrive et qui a quelque importance pour l'homme. Une définition plus générale d'un événement est la suivante : *ce qui se produit, arrive ou apparaît ; un fait important, une circonstance marquante.*

   Selon les domaines scientifiques où ils sont utilisés, les événements ont des définitions diverses : en physique, un événement est toute spécification d'une position et d'un instant qui peut être représentée par un point dans le *système temps-espace* ; en mathématiques, un événement est un sous-ensemble mesurable d'un espace de probabilités ; en statistiques, un événement est un sous-ensemble de l'espace de tous les résultats possibles d'une expérience ; en chimie, un événement est le début ou la fin d'une activité spécifique dans un réseau PERT.

   Enfin, en informatique, un événement est soit une occurrence ou une arrivée significative d'un programme ou d'une tâche telle l'achèvement d'une opération asynchrone d'entrée-sortie, soit une transaction ou toute autre activité qui affecte l'enregistrement d'un fichier.

2. **Etat**

   Un état est *une manière d'être d'une personne ou d'une chose considérée dans ce qu'elle a de durable* [Rob93].

   Dans le domaine scientifique, le terme d'état est surtout utilisé en physique ; l'état de la matière est sa phase (solide, liquide ou gazeux), l'état d'un corps est

sa manière d'être relativement à sa cohésion, à l'arrangement ou à l'ionisation de ses atomes, etc.

Enfin en informatique, un état est la manière d'être d'un objet pendant une période de temps déterminée. Un état s'étend sur une période de temps et un changement d'état est dû à un événement.

3. **Condition**

Une condition est *un état, une situation ou un fait dont l'existence est indispensable pour qu'un autre état ou un autre fait existe* [Rob93]. Une définition plus générale d'une condition est *une circonstance à laquelle est subordonné l'accomplissement d'une action soumise à la production d'un phénomène.*

En mathématiques, une condition est une prémisse, un traitement ou une restriction dont dépend un résultat mathématique ou une conséquence. Une condition est nécessaire si sa vérité est impliquée par la vérité d'une autre proposition. Elle est suffisante si sa vérité implique la vérité d'une autre proposition. Elle est nécessaire et suffisante si sa vérité équivaut logiquement à celle d'une autre proposition.

En informatique, la différence entre condition nécessaire et condition suffisante n'est pas explicite. Cependant, les règles actives des systèmes de gestion de bases de données actifs sont une sorte de renforcement de l'approche "condition suffisante" alors qu'un formalisme de type "table de décision" est plus proche d'une "condition nécessaire" en exprimant pour chaque action à effectuer les conditions à remplir.

4. **Action**

Une action est une manière d'agir ou le fait de produire un effet sur quelqu'un ou quelque chose [Rob93]. Plus spécifiquement, une action est :

- un fait ou une faculté d'agir ou de manifester sa volonté en accomplissant quelque chose ;
- *un acte ou un effet produit par quelque chose ou quelqu'un agissant d'une manière déterminée.*

En mécanique, une action est une force externe appliquée à un corps et contrebalancée par une force égale dans la direction opposée appelée réaction (d'après la troisième loi du mouvement de Newton).

Enfin, en informatique, et plus particulièrement en intelligence artificielle, une action est un terme se référant à la clause d'une règle de production qui indique une ou plusieurs conclusions qui peuvent être tirées si les prémisses sont satisfaites.

Les termes utilisés en informatique pour exprimer le comportement d'une application sont divers. Parfois cependant, leurs limites ne sont pas très claires et conduisent à des abus de langages. Nous préférons à ces termes spécifiques des termes plus généraux et plus proches du niveau conceptuel, donc de la perception d'un utilisateur : nous parlons en effet de **règle** de comportement et de **réaction** à une **situation**.

1. **Règle**

   Une règle est ce qui est imposé ou adopté comme ligne directrice de conduite ou encore *une formule qui indique ce qui doit être fait dans un cas déterminé* [Rob93]. Des définitions plus spécifiques d'une règle sont les suivantes :

   - une prescription qui s'impose à quelqu'un dans un cas donné, un principe de conduite, une loi ;
   - un principe qui dirige l'enseignement d'une science ou d'une technique, une convention (par exemple une règle de grammaire) ;
   - *ce qui se produit ordinairement dans une situation donnée.*

   Des définitions scientifiques sont les suivantes :

   - *un traitement de conditions qui sont communément observées dans une situation donnée* ;
   - un traitement sur une partie prescrite d'une action désignée pour obtenir un résultat donné.

   En informatique, ce terme est surtout utilisé en intelligence artificielle où une règle de production est un moyen formalisé pour exprimer de l'information dans un format cause-à-effet, généralement sous la forme "si... alors...".

2. **Situation**

   Une situation est *l'ensemble des circonstances* dans lesquelles une personne *se trouve* [Rob93]. Des définitions plus spécifiques sont les suivantes :

   - l'état ou la fonction de quelqu'un ou de quelque chose dans un groupe, une place ou un rang ;

– *l'état de quelque chose*, d'un groupe, d'une nation, *par rapport à une conjecture donnée, dans un domaine déterminé.*

En littérature, une situation est un état caractéristique issu d'une action ou d'un événement et que traduisent un ou plusieurs personnages d'un récit.

Enfin dans le domaine scientifique, le terme de situation est surtout utilisé en géographie : une situation est la localisation d'un concept géographique en relation avec son environnement, une localisation relative à des concepts par opposition à sa localisation absolue (site).

En informatique, le concept de situation n'est généralement pas utilisé.

3. **Réaction**

Enfin, une réaction est une réponse à une action par une action contraire tendant à l'annuler ou *la manière dont une personne ou un groupe réagit face à un événement ou à l'action de quelqu'un d'autre* [Rob93].

Le terme de réaction prend surtout de l'importance dans les domaines scientifiques : en mécanique, une réaction est une force qu'exerce en retour un corps soumis à l'action d'un autre corps (d'après la 3ème loi du mouvement de Newton) ; en chimie, une réaction est l'action réciproque de deux ou plusieurs substances qui entraîne des transformations chimiques ; en physiologie, une réaction est toute réponse à une stimulation physique, par exemple une réaction de réflexe ; en psychologie, une réaction est la réponse mentale et émotionnelle élicitée en réponse à une situation donnée.

En informatique, le concept de réaction n'est généralement pas utilisé.

Le tableau 1.1 synthétise les définitions que nous retenons de tous les termes introduits pour modéliser le comportement d'une application.

1.3 Démarche et contributions de ce travail

L'étude du cahier des charges d'une dizaine d'applications réactives du monde réel est à l'origine de ce travail. En effet, elle permet de mettre en évidence que les cahiers des charges d'applications réactives intégrent différents types de comportements communs à la plupart des applications dans des domaines divers. Nous avons établi une

Terme	Définition type
Evénement	Tout fait important ou toute circonstance marquante, à un certain moment et dans un certain contexte.
Etat	La manière d'être d'une chose considérée dans ce qu'elle a de durable.
Condition	Une circonstance à laquelle est subordonné l'accomplissement d'une action soumise à la production d'un phénomène.
Action	Un acte ou un effet produit par quelque chose ou quelqu'un agissant d'une manière déterminée.
Règle	Ce qui se produit ordinairement dans une situation donnée.
Situation	L'ensemble des circonstances dans lesquelles une chose se trouve par rapport à une conjecture donnée, dans un domaine déterminé.
Réaction	La manière dont une personne ou un groupe réagit face à un événement ou à l'action de quelqu'un d'autre.

TAB. 1.1 – *Synthèse des définitions des concepts utilisés*

classification de ces types de comportement que nous appelons **règles de comportement** ou **situations comportementales** et qui globalement, s'expriment sous la forme **Situation-Réaction**. Cette classification des types de besoins comportementaux communs à plusieurs domaines d'applications constitue le premier apport de cette thèse.

A partir de cette classification, nous souhaitons alors déterminer comment de tels besoins comportementaux sont pris en compte, tant lors de la représentation et de l'analyse des besoins d'une application que lors de l'implantation de ces besoins dans un système technologique. Une étude des techniques proposées dans des méthodes reconnues dans le monde professionnel ou à l'étude dans le monde de la recherche est ainsi menée dans le but de répondre à la première préoccupation. Des limites sont constatées dans ces méthodes concernant leur prise en compte des situations comportementales des applications dès le niveau de la représentation des besoins.

Suite aux limites constatées dans les méthodes existantes pour représenter les situations comportementales des applications et parce que le besoin de réutilisation se fait de plus en plus ressentir dès les premières phases du développement d'une application, nous étudions alors une nouvelle approche proposée récemment appelée **approche à base de patrons**. Cette approche s'oppose au processus descendant classique utilisé par les méthodes traditionnelles en proposant des patrons représentant des abstractions générales de plus haut niveau que la notion de classe et permettant de réutiliser des techniques éprouvées dès la représentation des besoins. Nous proposons donc une adaptation de cette approche permettant de prendre en compte dès le niveau de l'ex-

pression des besoins les situations comportementales des applications réactives.

Cette approche permettant le couplage entre des besoins du monde réel et des composants logiciels, nous traitons enfin de l'aspect conception d'une solution technique. En choisissant un système cible particulier, le système de gestion de bases de données actif NAOS/O_2, nous montrons la faisabilité de nos propositions du point de vue de l'implantation. L'approche à base de patrons que nous préconisons est ainsi expérimentée dans le cadre de la conception d'applications de bases de données actives et autorise un regard critique vis-à-vis des fonctionnalités offertes par de tels systèmes.

Pour résumer, les apports de cette thèse se situent à trois niveaux et sont validés de façon pratique sur une dizaine d'applications du monde réel :

- une classification des besoins comportementaux d'applications de domaines différents ;
- la proposition d'une approche à base de patrons et la justification de son utilisation pour la représentation des besoins comportementaux des applications réactives dès le niveau de l'analyse des besoins ;
- l'expérimentation de cette approche dans le cadre de la conception de bases de données actives.

Plus globalement, l'approche que nous préconisons permet de réutiliser l'expression de besoins comportementaux tant au niveau de l'analyse des besoins d'une nouvelle application qu'au niveau de la conception d'une solution technique.

1.4 Organisation du document

La suite de ce document est organisée en six chapitres.

- Le chapitre 2 présente tout d'abord les applications réactives que nous avons étudiées dans des domaines variés (industrie chimique, domaine énergétique, domaine médical, etc.), l'étude de ces applications constituant le point de départ de cette thèse. Puis, il propose une classification des types de besoins comportementaux selon cinq catégories différenciées par leur nature. Ces besoins comportementaux constituent les mêmes besoins de réactions à des situations significatives se produisant dans ou hors de l'application et sont regroupés selon le modèle commun Situation-Réaction.

- Le chapitre 3 a pour but d'analyser la façon dont les situations comportementales sont prises en compte dans les méthodes traditionnelles d'analyse et de conception d'applications. Après une introduction relatant des problèmes généraux de la conception d'une application, l'essentiel du chapitre est consacré à une analyse des techniques utilisées dans diverses méthodes d'analyse et de conception pour représenter le comportement des applications et conclut quant aux limites de ces techniques.

- Le chapitre 4 est basé sur la constatation des limites des techniques traditionnelles pour la représentation des situations comportementales des applications réactives et présente les principes tout à fait originaux des approches à base de patrons. Ces approches ayant pour but d'améliorer la réutilisation de composants éprouvés dès le niveau de l'analyse des besoins, un rappel sur la notion de réutilisation est auparavant effectué.

- Après une illustration de l'usage de patrons pour représenter des situations comportementales d'applications réactives, le chapitre 5 développe un ensemble de nouveaux patrons que nous préconisons pour prendre en compte plus complètement et plus efficacement les situations comportementales tant au niveau de la représentation des besoins qu'au niveau de l'implantation dans un système cible.

- Le chapitre 6 décrit l'expérimentation que nous avons menée sur l'utilisation des patrons dans le cadre des systèmes de gestion de bases de données actifs, plus particulièrement avec le système NAOS. Après des conclusions sur l'utilisabilité de ces patrons, le prototype développé dans le cadre de cette expérimentation est finalement présenté.

- Enfin, le chapitre 7 conclut ce document en rappelant les apports essentiels du travail et en donnant quelques perspectives.

Des annexes complètent ce manuscrit en explicitant des notions, des systèmes ou des outils utilisés dans le cadre de cette thèse. La première annexe illustre la présentation générale des approches à base de patrons grâce à l'exemple détaillé d'un langage de patrons. Le système de gestion de bases de données actif NAOS est ensuite présenté. Les classes O_2 d'un extrait de notre application privilégiée sont ensuite déclarées à partir d'un modèle objet OMT afin que les règles actives NAOS utilisées dans le chapitre 6 soient plus facilement compréhensibles. Enfin, les méta-outils GraphTalk et LEdit utilisés pour le développement de notre prototype sont présentés ainsi que la traduction de la grammaire NAOS en LEdit.

Chapitre 2

Des situations comportementales dans des applications réactives

AFIN DE MIEUX COMPRENDRE LEURS BESOINS et plus particulièrement leurs besoins "comportementaux", nous étudions une dizaine d'applications du monde réel et établissons une synthèse de ces caractéristiques. Nous identifions différents types de comportements communs à la plupart des applications ; ceux-ci sont appelés **situations comportementales** ou **règles de comportement**. Le but principal de ce travail est de proposer une manière pour exprimer et modéliser de façon pertinente, satisfaisante pour l'utilisateur et si possible réutilisable dans d'autres domaines d'applications, une situation comportementale d'un domaine d'application. Cet objectif qui se situe au niveau de l'expression et de la modélisation des besoins du monde réel peut paraître ambitieux, c'est pourquoi nous l'abordons d'une manière pragmatique et pratique. L'aboutissement de cette étude est la définition d'un ensemble de **situations-réactions** que nous présentons par la suite.

2.1 Des applications réactives

Les applications du monde réel que nous avons étudiées appartiennent à différents contextes : manufacture [Bas94], commande et contrôle dans le domaine militaire, gestion des transports, industrie chimique, workflow, ingénierie concurrente, domaine médical, applications régulatoires (transport et stockage de matériaux dangereux) et gestion de l'énergie. Plusieurs de ces applications ont été étudiées en détail dans le

projet Esprit III IDEA [1] [Con92] [Lop96] dont le but était le développement d'un prototype industriel de bases de données orientées objets, déductives et actives [Fro95c]. D'autres sont issues de collaborations avec d'autres partenaires.

Ces applications sont particulièrement intéressantes pour leur particularité à posséder un cahier des charges comportant des **règles de comportement** [Fro95d]. Dans la suite du document, lorsque nous ne parlons pas explicitement de règles actives d'un SGBD actif, nous employons le terme de règle au sens de la définition encyclopédique utilisée communément : *ce qui se produit ordinairement dans une situation donnée.*

2.1.1 Système de commande et de contrôle

Le système de commande et de contrôle dont nous traitons ici est celui d'un navire militaire [Mon94a]. Il est responsable de conserver des traces des ressources disponibles sur le navire (armes, missiles, etc.), des données de l'environnement (position du navire ou distance de la base) ainsi que des objets variés que des informateurs localisent sur le navire (distance d'un allié ou type d'un ennemi). De plus, il doit gérer de nombreuses **contraintes**, en particulier concernant la position du navire : *la longitude de la position du navire doit être comprise entre 0 et 360 degrés, sa latitude entre 0 et 180 degrés Sud et 0 et 180 degrés Nord.* Enfin, il doit produire un **plan** de défense pour le navire si celui-ci est en danger, selon les étapes suivantes :

- identifier les ennemis menaçants ;
- éliminer les duplications possibles d'ennemis ;
- examiner l'historique associé aux ennemis ;
- assigner des priorités aux ennemis pour combattre les plus dangereux d'abord ;
- créer un plan de défense pour combattre les ennemis tout en respectant des contraintes telles que *l'arme x ne peut être utilisée pour combattre l'ennemi y* ;
- si le système ne peut assigner d'armes à une menace, alors l'état de la menace devient non combattue et une procédure particulière est mise en place pour combattre l'ennemi.

1. Nous remercions les responsables du projet IDEA pour nous avoir donné accès à de nombreux documents relatifs à ces applications.

2.1.2 Gestion des transports

Dans le domaine de la gestion des transports, deux applications présentent plus particulièrement des types de règles de comportement [BM94] : un allocateur de portes dans un aéroport et un système pour le planning du personnel d'une compagnie de trains.

2.1.2.1 Un allocateur de portes

Dans un aéroport, un allocateur de portes a pour but d'affecter les avions à une et une seule porte de départ pour le décollage, selon deux phases principales :

- création, une fois par mois, d'un calendrier des portes assignées aux vols ;
- correction du calendrier après des événements non anticipés (mauvais temps, retards, incidents mécaniques, etc.).

Les **contraintes** sur le calendrier des portes sont nombreuses, par exemple :

- les vols nationaux (respectivement internationaux) arrivent et partent à des portes nationales (respectivement internationales) ;
- certaines portes exceptionnelles peuvent accueillir à la fois des vols nationaux et internationaux.

Le planning est quant à lui soumis à certaines **règles** :

- assigner une porte à tous les avions ;
- respecter les contraintes de priorités entre les avions selon les compagnies ;
- prendre en considération le temps requis pour garer un avion et embarquer ;
- prendre en compte des critères de coût pour l'assignation des portes ; par exemple, il est préférable que les gros avions partent avant les petits.

Enfin, de nombreux **critères de sécurité** doivent être satisfaits, par exemple :

- deux gros avions ne doivent pas être proches ;

– l'avion doit arriver à la porte au moins 15 minutes avant que les passagers ne commencent à embarquer.

La deuxième phase du planning implique quant à elle de réagir à des événements particuliers, par exemple le *mauvais temps*.

2.1.2.2 Un système de support au planning du personnel et des services

Toute compagnie de trains doit bénéficier d'un système capable d'affecter le personnel aux services offerts par la compagnie. Comme pour l'application précédente, les problèmes principaux concernent l'optimisation du planning quotidien du personnel en début de mois et la **réaction à des événements** apparaissant en journée et influençant l'opérabilité du service, par exemple *l'échange de services entre des agents*, *l'utilisation d'agents auxiliaires*, *l'absence ou le retour anticipé d'un agent*, *la réduction de capacité d'un train*, *la suppression d'un train* ou *la substitution d'un train par un autre*, etc. Plusieurs **contraintes** doivent en outre être satisfaites, par exemple :

– l'affectation d'un agent à un service doit correspondre à la qualification de l'agent ;

– l'affectation d'un chemin composé à un agent nécessite que la station d'arrivée du premier chemin soit la même que celle du départ du deuxième chemin ;

– si un agent a déjà travaillé pendant la semaine, il doit se reposer ensuite.

2.1.3 Industrie chimique

L'un des aspects essentiels à considérer dans le domaine de l'industrie chimique est la gestion de la sécurité et plus particulièrement le contrôle de l'hygiène, de la protection de l'environnement ou de la sécurité du personnel. Les deux applications étudiées dans ce domaine traitent tout naturellement de ces aspects [BM94].

2.1.3.1 Un système d'identification du risque

Ce système a pour but l'identification et le rapport des risques importants inhérents à la conception d'un produit chimique. Il faut pour cela gérer de nombreuses **contraintes** sur la configuration du produit et la plausibilité des données, décrire le

modèle d'équipement et le modèle de fluides et discriminer entre les hasards importants ou non. De plus, il est nécessaire de contrôler **la propagation de fautes** :

- un événement initial cause une déviation dans la conception du produit ;
- la déviation est propagée et modifiée ;
- la déviation propagée engendre un événement terminal.

2.1.3.2 Un système de support d'urgence

Ce système construit une simulation d'exercices destinés à tester les réponses prévues à des incidents en cas d'urgence. Les principaux composants du système sont donnés ci-dessous.

- De nombreuses bases de données caractérisent le site et ses environs (géographie et population du site, conditions atmosphériques, etc.).
- Un modèle de génération d'événements détermine comment les éléments d'un produit à risque répondent à une condition d'urgence. Il est réalisé au moyen d'un ensemble de **règles** dérivées d'études de sécurité sur le site et qui décrivent comment un produit réagit à des fautes. A chaque étape de la simulation, cet ensemble est testé pour déterminer si un nouvel événement apparaît.
- Un ensemble de modules de calcul modélise l'urgence et gère des propagations d'événements : libération de gaz et de liquides, dispersion, explosion, propagation de feu, effets sur la santé, transport et trafic, évacuation et comportement humain, etc.
- Enfin, un gestionnaire de simulation, composant central du système, maintient l'enregistrement historique de la simulation et détermine quels modules de calcul doivent être utilisés à des instants particuliers.

La plupart des bases de données sont dynamiques : leur contenu est modifié au fur et à mesure que la simulation progresse.

2.1.4 Gestion d'activités (Workflow)

Un système workflow possède deux composants principaux : une base de données pour gérer des données relatives à la structure du processus automatisé et à l'in-

formation et la documentation gérées par le processus et des **règles** pour aider à l'automatisation du processus de workflow.

Le système workflow que nous avons étudié a été développé pour une compagnie d'assurance italienne [GQVG94]. Il a pour but la gestion par des personnes différentes, de polices d'assurances impayées et a deux besoins principaux :

- contrôler la terminaison d'une tâche ; par exemple, le traitement d'une police d'assurance n'est pas terminé tant que celle-ci n'est pas archivée ;
- vérifier que l'état d'une tâche est correct avant de passer à la tâche suivante.

2.1.5 Ingénierie concurrente

Dans le domaine de l'ingénierie concurrente [FGVLV94], l'activité de conception est partagée par plusieurs participants grâce à des données affectées par tous les participants lors de décisions relatives à la conception globale du projet. Dans ce cadre, l'interaction entre les concepteurs de disciplines différentes peut mener à des incohérences dans les données partagées. Ces incohérences doivent être détectées le plus tôt possible et signalées à tous les autres participants.

Par exemple, en ingénierie civile, la construction d'un bâtiment met en œuvre des architectes, des ingénieurs, des ouvriers, etc. Les concepteurs définissent les limites de leur région d'interaction selon des **règles**. Toute nouvelle donnée insérée par un participant dans la conception est vérifiée par rapport à ces règles :

- si les concepteurs doivent participer à un développement lié à leur région d'interaction, ils modifient une version privée d'une base de données partagée et chaque participant peut être informé ;
- sinon, un nouveau statut commun de la base de données est construit et sert de fondation pour les étapes futures pour tous les participants.

Pour résumer, une application dans le domaine de l'ingénierie concurrente a pour but de stocker les données partagées relatives à plusieurs participants et de gérer les incohérences sur ces données.

2.1.6 Exploration de données (Data Mining)

Le terme "Data Mining" est utilisé pour suggérer la découverte de connaissances non explicitement stockées dans une base de données.

L'exemple étudié a pour contexte une école publique où il est nécessaire d'accéder et d'évaluer des types variés de données socio-économiques aussi bien à l'intérieur de l'école que dans son voisinage [FGVLV94]. Le but est de rendre disponibles des données pour les administrateurs des écoles (le principal, les intendants, etc.) afin de les aider à formuler des politiques et à prendre des décisions basées sur une vue compréhensive et collective de toutes les données individuelles. Il sera ainsi possible de comprendre l'origine des augmentations croissantes d'absences à certains cours et de savoir en particulier si un groupe d'étudiants influence les autres à manquer les cours de façon répétée. Trois groupes d'information particuliers existent :

- les suspensions et expulsions d'étudiants de certains cours,
- les inscriptions des étudiants aux cours,
- les mailing-listes utilisées par les étudiants dans différentes écoles.

Grâce à ces informations, il est possible de comprendre l'origine des augmentations croissantes d'absences. Afin de découvrir si un groupe d'élèves influence les autres à manquer des cours, des **règles** définissent par exemple les étudiants récalcitrants comme ceux qui ont été suspendus au moins un jour pendant l'année scolaire ou encore les étudiants à risque comme les étudiants amis d'étudiants récalcitrants ou amis d'amis d'amis ... d'amis d'étudiants récalcitrants.

2.1.7 Applications régulatoires

Une application régulatoire [FGVLV94] invoque des activités gérées par des **règles** et des standards différents. Par exemple, le transport et le stockage de matières dangereuses implique d'importants standards et des règles locales, nationales et internationales. En effet, certains environnements sont sujets à des situations difficiles (régions frontalières, aéroports, etc.). Chaque pays possède donc un flux intense d'informations concernant les matières dangereuses qu'il doit stocker (clients, compagnies de transport, intendants du stockage, etc.), ainsi que des données et des règles spécifiques à ces matières. Les règles à suivre lors du transport et du stockage des matières dangereuses

sont celles des pays origine et destination ainsi que celles de chacun des pays situés sur le chemin de transport des matières dangereuses.

Une application de stockage de matériaux dangereux contient donc de nombreuses règles comme des **règles de stockage** d'un matériau définies par l'Organisation Internationale Maritime ou des règles spécifiques au site de stockage. L'un des buts d'une telle application est de déterminer et de gérer le placement des containers par rapport à ces règles de stockage. Cependant, de telles règles sont souvent modifiées par l'introduction de nouvelles matières dangereuses, de nouvelles politiques de stockage ou de nouvelles régulations nationales ou internationales. Elles doivent donc être vérifiées après toute modification et peuvent être utilisées dans les cas présentés ci-dessous.

- Le gestionnaire décide de stocker un nouveau container dans un entrepôt. Cette décision est-elle acceptable en considérant les règles de stockage déjà définies ?
- Un accident survient. Les pompiers veulent obtenir toutes les informations sur les matières dangereuses stockées près de l'accident.
- L'accident a pour conséquence la définition d'une nouvelle règle de séparation. Quels containers déjà placés violent cette règle ?

2.1.8 Gestion d'énergie

L'ENEL est une compagnie italienne de distribution d'électricité [Mon94b]. Elle possède deux applications principales : l'une supporte une description dynamique des réseaux de distribution d'électricité, l'autre est un système de support pour l'analyse de la connexion d'un utilisateur à un réseau électrique.

2.1.8.1 Description dynamique de réseaux de distribution d'électricité

La base de données Energy Management System (EMS) supporte la description dynamique de réseaux de distribution d'électricité. Un réseau de distribution d'électricité est composé d'un ensemble de stations et de nœuds connectés deux à deux par des branches (cf. figure 2.1). Une station distribue une puissance (un voltage). Un nœud est un point du réseau où le voltage peut être transformé et où l'énergie peut être distribuée à un autre niveau de voltage ou partagée à travers différentes branches de sortie. Enfin, chaque branche est un arc dirigé et connecte exactement deux stations ou deux nœuds, ou un nœud et une station.

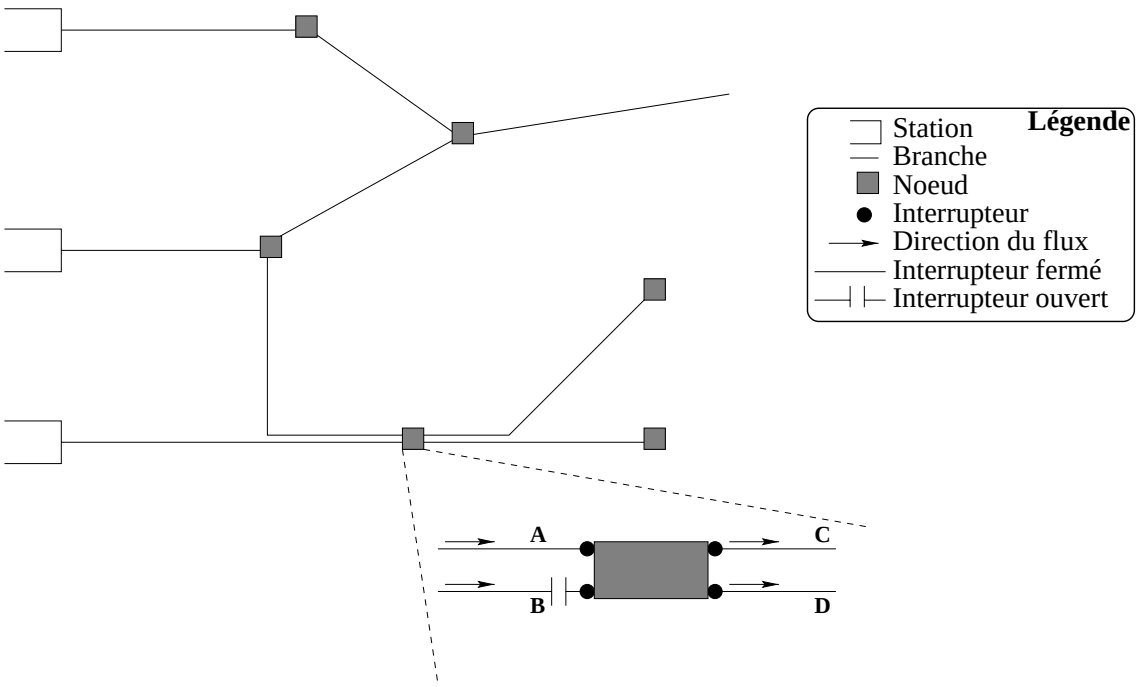


FIG. 2.1 – *Description d'un réseau électrique [Mon94b]*

Le modèle du réseau de distribution électrique offre :

- une description de la topologie du réseau, c'est-à-dire sa structure statique donnée par l'ensemble de stations, de nœuds et de branches les connectant ;
- une description des opérations dynamiques du réseau, c'est-à-dire de la façon dont la puissance est distribuée à travers le réseau étant donné le statut de tous les interrupteurs (ouvert ou fermé) et la direction du flux électrique.

De nombreuses **contraintes** doivent être respectées. Des exemples sont donnés ci-dessous.

- Chaque nœud possède au moins une branche entrante dont l'état est fermé ; toutes les branches dont la puissance passante est nulle sont considérées comme branches de sortie.
- Chaque branche connecte deux nœuds : elle doit être sortante pour un nœud et entrante pour l'autre.
- La puissance passante de chaque branche ne doit pas excéder la puissance passante maximale de cette branche.

– La puissance disponible dans chaque station primaire est supérieure ou égale à la somme de la puissance requise par tous les nœuds de la station.

De plus, comme le montrent les exemples ci-dessous, des données sont calculées à partir d'autres.

– La puissance entrante d'un nœud est égale à la somme des puissances passantes des branches entrant dans ce nœud.

– La puissance sortante d'un nœud est égale à la différence entre sa puissance entrante et sa puissance absorbée.

– L'ensemble de tous les nœuds alimentés par une station est composé de tous les nœuds directement connectés à cette station par une branche fermée, ou récursivement par tous les nœuds directement connectés par une branche fermée à un nœud alimenté par cette station.

Enfin, il est nécessaire de **réagir** à des modifications de la base de données telles celles présentées ci-dessous.

– Quand un interrupteur sur une branche est ouvert, le flux courant de la branche devient nul.

– Les branches reliées à un nœud détruit sont automatiquement détruites.

2.1.8.2 Analyse de la connexion d'un utilisateur à un réseau d'énergie

Le but de cette base de données est d'assurer la qualité du service sur le réseau public de distribution d'électricité (continuité du service, connaissance à tout instant des caractéristiques du voltage ou des perturbations dûes à des survoltages ou à des changements de voltage, etc.). Cette recherche de qualité nécessite une connaissance extensive des standards et des régulations ainsi que l'exécution de nombreux calculs.

Des **règles de connexion** permettent au système de détecter des incohérences dans le réseau après la connexion d'un utilisateur et de traiter toutes les transactions possibles avec cet utilisateur, par exemple de nouvelles connexions, des modifications de contrat ou des plaintes pour mauvaise qualité. L'analyse de la connexion d'un utilisateur se déroule en deux phases :

– vérification de la capacité du réseau existant et étude de renforcements possibles ;

– vérification de l'interaction entre l'installation d'un utilisateur et le système du point de vue des perturbations.

L'application est composée de deux éléments : le système gère la structure du réseau de distribution et une base de données stocke toutes les informations sur les installations des utilisateurs (caractéristiques des réseaux privés, des équipements ou des plaintes des utilisateurs, etc.).

2.1.9 Domaine télématique

Le projet Esprit EUROWARE a étudié une application appartenant au domaine de la télématique destinée à mettre en pratique la réutilisation d' "assets" dans de larges réseaux télématiques [BM94]. Les assets sont de trois types : commerciaux (propositions de clients, présentations de transparents, informations des clients), techniques (produits de travail, rapports d'expérience, processus/méthodes) et organisationnels (structure, personnes, etc.).

Plusieurs **contraintes** assurent la cohérence de la base de données :

– les droits d'accès d'un "réutilisateur" sont contrôlés ;

– la gestion de configuration est permise ;

– différentes versions d'assets sont traitées au moyen de contraintes et des licences sont accordées en respect des droits du réutilisateur.

Les fonctions de l'application dont des exemples sont présentés ci-dessous, concernent des activités **déclenchées sur des événements particuliers**.

– Quand l'utilisateur recherche un asset non disponible, il laisse une requête qui sera satisfaite automatiquement par le serveur quand l'asset sera disponible.

– Le réutilisateur peut laisser des messages pour contacter des partenaires du projet pour une demande de coopération, une proposition, etc... Il peut aussi rechercher des contacts, le serveur l'informant automatiquement quand les données seront disponibles.

– Des opérations pour vendre ou acheter des assets sont réalisées.

2.1.10 Domaine médical

Deux types différents d'applications ont été étudiées dans le domaine médical.

2.1.10.1 Prescription de médicaments

OPADE est un système multi-langues de prescription de médicaments par ordinateur [RdZ94]. Il offre un support au prescripteur non seulement sur le plan médical, mais aussi sur les plans économiques et relatifs à la croyance du patient. Ses objectifs principaux sont d'augmenter l'efficacité et la sécurité de la prescription de médicaments en générant des critiques et des suggestions systématiques, de réduire les coûts en offrant une information sur le coût du traitement et enfin d'améliorer la croyance du patient dans le traitement en générant des explications adaptées.

Il est nécessaire en particulier de stocker de l'information sur les médicaments, de connaître les pratiques de prescription des médicaments, de calculer le coût du remboursement selon des modèles de remboursement utilisés dans différents pays et de déduire les éventuels états pathologiques dûs à la prescription d'un médicament.

2.1.10.2 Fonctionnement de l'unité médico-technique d'un hôpital

Le fonctionnement de l'unité médico-technique du Centre Hospitalier Universitaire de Grenoble[2] constitue notre application privilégiée [Fro95e] et sera utilisée dans la suite du document pour illustrer nos exemples et nos propositions.

Dans un centre hospitalier, la procédure d'accueil d'un patient constitue un processus bien déterminé (cf. figure 2.2). Nous nous intéressons plus particulièrement à l'aspect médico-technique de ce processus. L'Unité Médico-Technique (UMT) est chargée de traiter les examens et les analyses dont le personnel médical intérieur ou extérieur à l'hôpital a besoin pour confirmer un diagnostic ou contrôler l'évolution d'une maladie. Son fonctionnement est dirigé par un ensemble d'étapes à partir de l'accueil d'un patient ou de la réception d'un prélèvement : 1) acquisition des demandes d'actes (examens ou analyses), 2) planification des examens, 3) préparation de la réalisation des actes (examens ou analyses), 4) saisie, acquisition et diffusion des résultats, 5) gestion de l'urgence, 6) gestion du brancardage et 7) saisie du Dossier de Spécialité

2. Cette application a pû être étudiée en détail grâce aux informations fournies par Mr. Daniel Pagonis [Pag94] du Service d'Information et d'Informatique Médicale de Grenoble ; nous l'en remercions vivement.

Médico-Technique. D'autres activités permettent la gestion humaine et matérielle de l'unité médico-technique (réglage des appareils, test de nouveaux appareils, etc.).

Le système d'information d'une unité médico-technique a trois objectifs principaux : assurer une logistique cohérente autour de l'accueil d'un patient ou de la réception d'un prélèvement, avoir une équipe technique et médicale efficace et opérationnelle et disposer d'outils d'aide à l'enseignement et à la recherche. Ses missions sont les suivantes :

- production des actes : acquisition des demandes d'actes, planification des actes, préparation et réalisation des actes, validation et interprétation des actes, enfin acquisition, diffusion et routage des résultats ;
- gestion des dossiers médicaux : gestion des comptes-rendus, codification médicale, évaluation de la démarche médicale, cotation et gestion des archives ;
- support et pilotage du service : gestion des consommables, du matériel, de l'activité du service et du personnel.

Trois types d'actes existent :

- les actes sans préparation ne nécessitent pas d'informations spécifiques, pas de diagnostic, pas de résultat,
- les actes avec préparation nécessitent la connaissance du dossier patient, pour préparation du protocole et interprétation du résultat,
- les actes avec planification nécessitent la réservation de ressources particulières (salle, matériel, médecin) ou la présence du patient dans le service.

Les examens et les analyses sont des types particuliers d'actes. Ils peuvent être internes (demandés depuis un service clinique de l'hôpital) ou externes (demandés depuis un établissement extérieur). Un examen nécessite la présence physique du patient dans le service. Il est planifié par une procédure de demande de rendez-vous, préparé pour l'élaboration d'un protocole individualisé, puis validé et interprété avec production d'un compte-rendu d'examen. Une analyse porte sur un échantillon de prélèvement. Elle n'a pas de phase de planification, mais doit respecter les contraintes horaires imposées par le cycle de validation. Elle nécessite parfois une phase de préparation et doit toujours avoir une phase de validation technique et biologique. Enfin, elle est interprétée par les médecins avec production d'un résultat et d'un compte-rendu.

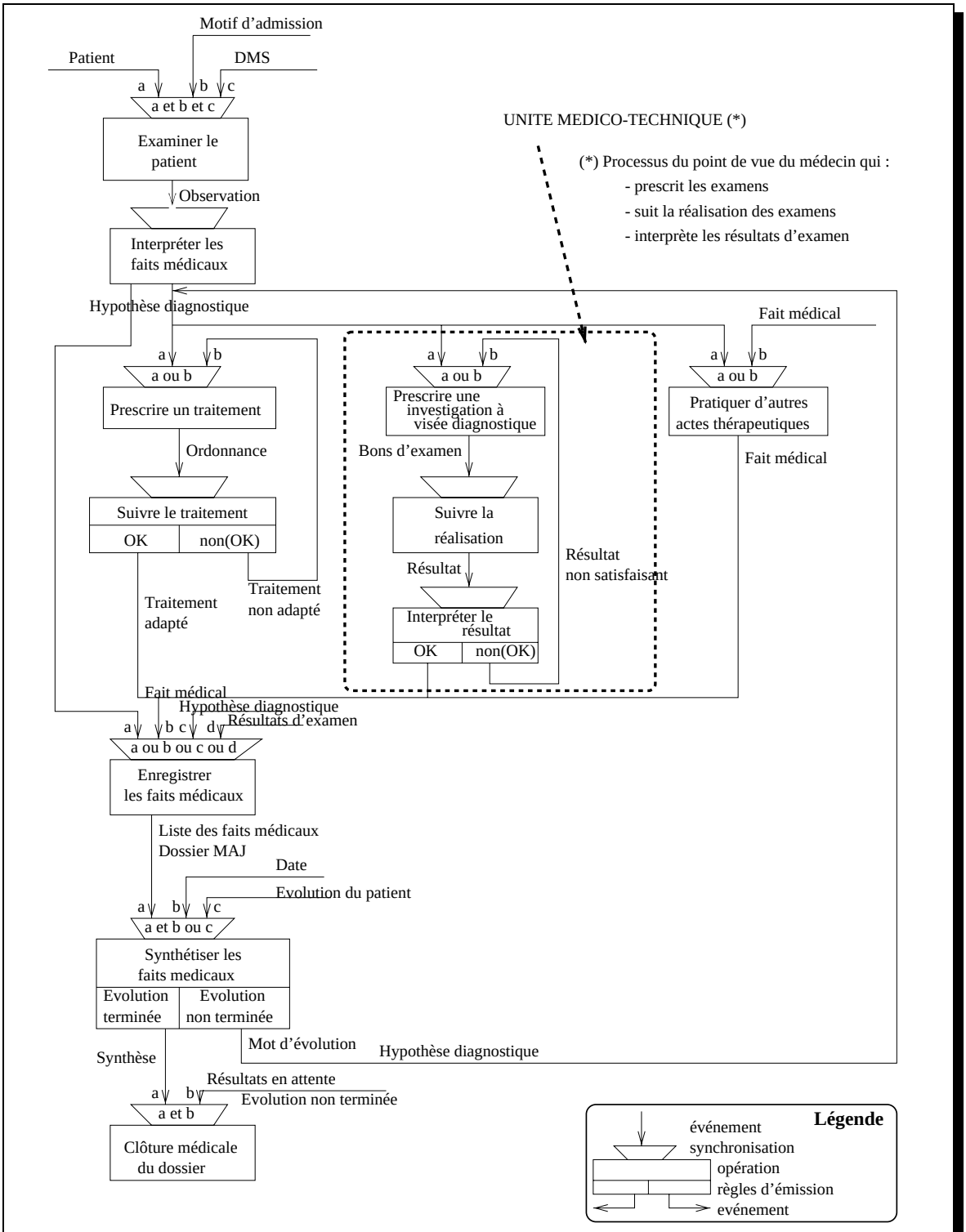


FIG. 2.2 – *MCT MERISE partiel modélisant le processus "Soigner un patient"*

Une demande d'acte nécessitant des ressources rares génère une demande de rendez-vous ou une réservation, qui donne lieu à une réponse sous forme de rendez-vous. Par exemple, tout examen doit être planifié par une procédure de demande de rendez-vous ; une demande d'examen génère donc automatiquement une demande de rendez-vous. Un rendez-vous est aussi nécessaire dans certains cas de demandes d'analyse, par exemple pour des analyses externes avec prélèvement sur le patient (comme une prise de sang). Toute demande de rendez-vous doit être validée : en fonction du degré de gravité et/ou du degré d'urgence, les rendez-vous pré-enregistrés sont envoyés au praticien, avec les informations éventuelles du dossier de spécialité (type d'examen, élaboration du protocole individualisé, confirmation de la priorité, refus ou blocage éventuel de la demande, prescription d'un examen préalable, ...). Lorsque la demande est validée par le médecin, le secrétariat de l'unité médico-technique réserve pour un patient une date et une heure de rendez-vous, un poste (salle, matériel) et un praticien. Une fiche de rendez-vous est éditée en annexe du dossier commun du patient avec selon le cas, une lettre personnalisée à l'attention d'un prescripteur externe ou un message de confirmation de rendez-vous. Si le rendez-vous est accordé le même jour que la demande et si cela est nécessaire, le brancardage est alerté. Enfin, il est nécessaire de maintenir un suivi des demandes de rendez-vous afin de sélectionner celles-ci selon les critères suivants : médecin ou service prescripteur, patient, poste, secteur, praticien, plage de dates de rendez-vous, demande acceptée, refusée ou bloquée.

Des informations sont retournées au prescripteur à l'enregistrement de la demande : ce sont des consignes de préparation à la réalisation de l'acte, des contre-indications éventuelles ou des consignes post-examen.

La réalisation de l'acte de base peut selon le cas induire la réalisation d'autres actes qui font eux-mêmes l'objet d'une demande de rendez-vous, interdire la réalisation d'actes incompatibles avec l'acte de base ou enfin supprimer des actes intervenant après la réalisation de l'acte de base s'ils n'apportent pas d'informations significatives supplémentaires.

La saisie des résultats se déroule en plusieurs étapes :

- acquisition des résultats d'analyse et d'examen,
- préparation de la diffusion des résultats,
- édition locale des résultats,
- validation technique, puis finale des résultats,

- interprétation des résultats avec élaboration d'un diagnostic, d'une observation ou d'un compte-rendu. Parfois, l'interprétation est précédée d'un avis clinique de la part du clinicien en charge du patient. Deux réactions à cette interprétation sont alors à considérer avant tout : alerte à une anomalie grave pouvant porter atteinte à la vie du patient, et orientation de la démarche éthiologique en fonction des résultats ;

- diffusion des résultats,

- enfin, suivi du diagnostic pour valider les hypothèses formulées lors de l'élaboration de l'interprétation. Ce suivi suppose la possibilité d'envoyer au prescripteur une demande de mise à disposition du compte-rendu d'hospitalisation dès la sortie du malade, la codification du diagnostic de sortie dans le dossier de spécialité médico-technique et la possibilité d'effectuer des relances périodiques au patient en fonction d'un diagnostic, par exemple pour contrôler l'évolution d'une grossesse.

Enfin, des cas d'urgence peuvent se produire qui nécessitent une réaction immédiate. Par exemple, la demande d'un examen en urgence implique la gestion d'une priorité au niveau de l'identification de la demande d'acte, qui permet à l'unité médico-technique de prendre en charge la préparation de l'acte avant l'arrivée du patient ou du prélèvement ; puis un accusé de réception est automatiquement retourné au prescripteur et un avis de brancardage est demandé avec indicateur de priorité. De la même manière, il peut être nécessaire d'envoyer en urgence le résultat d'un examen ou d'une analyse à un prescripteur sans aucune intervention préalable à la diffusion ; en l'absence du clinicien du service, les résultats sont systématiquement routés vers un autre clinicien.

2.1.11 Bilan

L'étude présentée ci-dessus permet de constater que des applications de domaines divers présentent des situations semblables, qui se produisent de façon similaire et ont besoin d'être corrigées ou analysées de la même manière :

- suivi de processus,

- déclenchement de diagnostic,

- activation de procédures spéciales en réaction à des situations inhabituelles,

- propagation d'événements,
- contrôle de fonctionnement,
- etc.

Dans la suite de cette section, nous classons ces situations semblables dans des **types de situations comportementales** communs à plusieurs domaines d'applications.

2.2 Proposition d'une classification de situations comportementales

2.2.1 Des règles pour spécifier les besoins et les exigences d'une entreprise

James Odell [Ode93a] introduit la notion de **règle** pour spécifier les besoins et les exigences d'une entreprise, non seulement sur le plan statique, mais aussi sur le plan dynamique. Les **règles** sont des **déclarations de politiques ou de conditions qui doivent être satisfaites**. Elles doivent être comprises et compréhensibles par la communauté d'utilisateurs, exprimées dans un langage proche de la langue naturelle et permettre aux experts de spécifier des politiques ou des conditions en utilisant des traitements explicites de petite taille. James Odell propose ainsi une classification des règles spécifiant les besoins et les exigences d'une entreprise selon deux catégories principales. Nous présentons cette classification avant d'en proposer une extension mieux adaptée aux applications réactives.

Règles de contraintes - Ces règles spécifient les politiques ou les conditions qui restreignent la structure des objets et leur comportement.

- *Règles de stimuli/réponses*: elles expriment le comportement en spécifiant des conditions qui doivent être vraies pour qu'une opération soit exécutée, par exemple :

  Quand un acte est demandé pour un patient
  Si ce patient est connu de l'hôpital
  Alors rechercher le dossier patient du patient
  Sinon créer un nouveau dossier patient pour le patient.

– *Règles de contraintes sur les opérations* : elles spécifient les conditions qui doivent être vraies avant et après l'exécution d'une opération, par exemple :

Accorder un RDV à un patient pour un acte
Ssi ce RDV est disponible
et ce RDV est compatible avec l'acte à effectuer.

– *Règles de contraintes sur la structure* : elles spécifient des conditions qui ne doivent pas être violées sur les types d'objets et leurs associations, par exemple :

Il faut toujours que
Le service dans lequel un malade est hospitalisé corresponde à sa maladie.

Règles de dérivation - Ces règles spécifient les politiques ou les conditions pour inférer ou calculer des faits à partir d'autres faits.

– *Règles d'inférence* : elles spécifient que si certains faits sont vrais, une conclusion peut être déduite, par exemple :

Un malade est diabétique
Ssi un taux important de sucre est relevé dans son sang ou ses urines.

– *Règles de calcul*: elles dérivent des informations à partir d'autres en effectuant un calcul, par exemple :

Le coût total du séjour d'un patient
est calculé de la façon suivante :
Durée du séjour * Taux journalier.

La classification proposée par James Odell présente des *règles* pour spécifier les besoins et les exigences d'une entreprise. Basée sur la nature de ces règles (vérification de contraintes, calcul, inférence, etc.), elle constitue le point de départ de la classification que nous proposons ci-dessous.

2.2.2 Présentation de différentes situations d'applications

Les applications présentées dans la section 2.1 montrent que l'aspect dynamique est critique et que dans de très nombreux cas, il convient de réagir à des situations inhabituelles ou à des modifications de l'application. Nous avons donc inventorié un ensemble de règles de comportement qui mettent en évidence divers types de *situations comportementales* présents dans les applications réactives [FRG96]. Dans notre volonté d'établir des expressions proches de la perception d'un utilisateur, nous classons ces types de situations comportementales par leur objectif d'un point de vue applicatif plus que par leur nature, comme cela est le cas dans la classification proposée par James Odell. Ainsi, certaines règles maintiennent un aspect structurel, d'autres un aspect dynamique. Les autres règles contrôlent l'évolution des objets au cours du temps ou sont utilisées pour planifier des tâches ou pour maintenir des objectifs. Nous présentons par la suite ces types de règles avec pour chacun, des exemples extraits des cahiers des charges des différentes applications présentées précédemment.

Notons que ces règles de comportement peuvent s'appliquer à tout niveau d'abstraction des **entités** de l'application : dans un environnement objet par exemple, elles peuvent porter sur un objet ou sur une classe, sur un attribut d'une classe d'objets ou sur un ensemble de classes, ou encore sur l'application elle-même.

2.2.2.1 Règles de structure

Les règles de structure ou de connaissance permettent de gérer les informations classiques portant sur la connaissance et la structuration de l'application, sur un plan statique : structure des entités, contraintes de cardinalité ou tout autre type de contrainte. Une règle de structure peut être considérée comme purement statique, mais dans la mesure où l'application doit réagir afin de retrouver une configuration "normale" lorsqu'une telle règle n'est plus vérifiée, nous l'acceptons comme *situation comportementale*. Des exemples sont donnés ci-dessous.

- (Commande et Contrôle) La longitude de la position du navire doit être comprise entre 0 et 360 degrés.
- (Energie) La puissance absorbée totale doit être inférieure à la puissance disponible.
- (Energie) Le niveau de voltage du point de connexion avec l'utilisateur doit être consistant avec celui demandé par l'utilisateur.

- (Energie) Quand un interrupteur sur une branche est ouvert, le flux courant de la branche est nul.
- (Energie) La puissance sortante est fonction de la puissance entrante et de la puissance absorbée par les branches.
- (Transport) Les vols nationaux arrivent et partent à des portes nationales.
- (Transport) L'allocation d'un agent à un service doit correspondre à sa qualification.
- (Applications régulatoires) Il y a restriction de la localisation de matières dangereuses dans des containers proches de bureaux.
- (Application médicale) Une patiente enceinte ne doit pas prendre un traitement composé d'aspirine.
- (Application médicale) Un rendez-vous doit être fixé pour qu'un examen puisse être exécuté.

2.2.2.2 Règles d'évolution

Les règles d'évolution prennent en compte l'évolution des entités de l'application au cours du temps. La notion de temps est fortement présente et une règle d'évolution compare une entité de l'application à un certain instant à la même entité ou à une autre, un instant plus tard. Des exemples sont donnés ci-dessous.

- (Transport) Une porte ne peut accueillir un avion qu'au minimum une demi-heure après le départ de l'avion précédent.
- (Transport) L'augmentation du salaire d'un agent ne peut pas être supérieure à 1000 francs.
- (Application médicale) Le rendez-vous doit être fixé moins de deux jours après la demande d'acte.
- (Application médicale) Un traitement n'ayant pas d'effet sous 2 jours doit être revu.
- (Energie) La puissance distribuée par un réseau électrique ne doit pas varier de plus de 100 000 KW d'une heure à l'autre.

2.2.2.3 Règles d'activité

Les règles d'activité ou de comportement gèrent l'activité d'une application et le comportement de ses entités. Elles regroupent tous les comportements "normaux" et attendus des entités de l'application. Des exemples sont donnés ci-dessous.

- (Energie) Chaque fois qu'un interrupteur est ouvert ou fermé, la description dynamique du réseau change.

- (Energie) Quand un nœud du réseau est détruit, les branches l'ayant comme fin sont aussi détruites.

- (Chimie) Le modèle de comportement des éléments chimiques spécifie la configuration unique du produit chimique (les éléments dont est composé le produit, les connexions entre les éléments, etc.).

- (Commande et contrôle) Quand une nouvelle menace apparaît, le système doit éliminer tous les ennemis dupliqués ; après élimination des ennemis dupliqués ou ajustement de la distance de l'ennemi ou de sa position, le système doit calculer les priorités ; après avoir déterminé les priorités des différentes menaces, le système doit créer un plan de défense pour combattre ces menaces.

- (Commande et contrôle) Si le système ne peut assigner d'armes à une menace, alors l'état du système deviendra Non Combattu.

- (Application médicale) L'unité médico-technique doit répondre à une demande d'acte en fixant un rendez-vous à un patient pour un examen ou en traitant des analyses.

- (Application médicale) Un prescripteur demande un acte au service prestataire ; des informations (consignes de préparation, informations de contre-indication, consignes de post-examen...) sont retournées au prescripteur à l'enregistrement de la demande.

2.2.2.4 Règles de contrôle de fonctionnement

Les règles de contrôle de fonctionnement concernent toutes les situations contrôlant le bon fonctionnement de l'application : contrôle de coût, contrôle de sécurité,

contrôle de flux, contrôle de coopération, contrôle d'objectif, etc. Ces situations induisent un comportement spécifique de l'application afin d'être vérifiées. Des exemples sont donnés ci-dessous.

- (Transport) Les gros avions doivent partir avant les petits (coût).
- (Transport) Deux gros avions ne peuvent pas être proches (sécurité).
- (Chimie) Comment déterminer le coût des hasards identifiés (coût)?
- (Ingénierie concurrente) Qui fait quoi, quelles sont les limites des régions d'interaction de chaque personne, quelles sont les tâches à réaliser, dans quel ordre, etc. (coopération)?
- (Application médicale) Comment augmenter les croyances du patient dans les médicaments (culture)?
- (Application médicale) Comment prescrire les médicaments plus sûrement (sécurité)?
- (Application médicale) Le suivi du diagnostic est nécessaire pour valider les hypothèses formulées lors de l'élaboration de l'interprétation.

2.2.2.5 Règles d'exception

De nombreuses applications sont destinées à la gestion d'un système, avec la particularité de pouvoir réagir à une situation ou une erreur exceptionnelle ou inhabituelle. Dans ce cas, il est nécessaire de réagir en prenant en compte ce nouvel élément. Les règles d'exception ou d'erreur permettent de gérer de telles situations. De plus, elles comprennent les règles de diagnostic destinées à détecter un comportement anormal ou une erreur. Des exemples sont donnés ci-dessous.

- (Transport) Certaines portes peuvent accueillir à la fois des vols nationaux et internationaux.
- (Transport) Que faire en cas de mauvais temps, de variation de paramètres particuliers, de suppression de trains, d'absence d'un agent, etc.?
- (Applications régulatoires) Que faire en cas d'introduction d'un nouveau produit dangereux?

Situation comportementale	But
Règle de structure	Maintenir la structure des entités de l'application.
Règle d'évolution	Gérer l'évolution des entités.
Règle d'activité	Gérer l'activité de l'application et le comportement des entités de l'application.
Règle de contrôle	Contrôler la vie de l'application.
Règle d'exception	Réagir à des situations inhabituelles ou à des erreurs ; traiter les exceptions et les erreurs.

TAB. 2.1 – *Synthèse des types de situations comportementales*

- (Chimie) Quelles sont les conséquences quand un événement arrive et cause une déviation d'une variable du processus ou d'une fonction de l'équipement ?
- (Application médicale) Si le rendez-vous est accordé le même jour que la demande, une alerte au brancardage doit être envoyée.
- (Application médicale) Deux réactions à l'interprétation des résultats sont à considérer : alerte à une anomalie grave pouvant porter atteinte à la vie du patient et orientation de la démarche étiologique.

2.2.2.6 Synthèse

La table 2.1 présente une synthèse des types de situations comportementales identifiables dans les applications de nature active.

2.2.3 Identification de situations comportementales

L'identification de situations comportementales (ou règles de comportement) dans des applications s'effectue dès la phase d'étude des besoins. Une situation comportementale est représentée dans le cahier des charges d'une application en langue naturelle. Elle exprime soit une contrainte qui doit toujours être vraie et qui porte sur une ou plusieurs entités de l'application (c'est le cas pour les règles de structure et les règles d'évolution), soit une réaction qui doit avoir lieu après l'occurrence d'un événement de l'un des types suivants :

- un événement prévu et "normal" : c'est le cas pour une règle d'activité ;

- un événement qui doit arriver et qui doit être contrôlé : c'est le cas pour les règles de contrôle ;
- un événement exceptionnel et imprévu : c'est le cas pour les règles d'exception.

Les critères nécessaires pour identifier des situations comportementales sont donnés ci-dessous.

- Une règle de structure s'exprime sous la forme d'une contrainte sur une ou plusieurs entités de l'application.
- Une règle d'évolution d'objet s'exprime sous la forme d'une contrainte temporelle sur une ou plusieurs entités de l'application.
- Une règle d'activité s'exprime sous la forme "Situation-Réaction" où la réaction a lieu suite à une situation attendue et prévisible.
- Une règle de contrôle d'événement s'exprime sous la forme "Situation-Réaction" où la réaction a pour but de contrôler que certaines conditions sont vérifiées avant qu'un événement prévisible n'apparaisse.
- Une règle d'exception par rapport aux situations habituelles s'exprime sous la forme "Situation-Réaction" où la réaction a lieu suite à une situation imprévue et exceptionnelle.

2.3 Conclusion : vers un modèle unifié...

Nous avons déterminé quelques critères pour classer une situation comportementale dans l'un des cinq types de situations comportementales communs à des domaines d'applications différents. Au début de notre travail de recherche, nous pensions que cette classification selon cinq types de situations comportementales serait un atout pertinent d'une part pour l'expression des besoins, d'autre part pour la conception et le développement des applications. Cependant, un besoin d'une application reste difficile à classer de façon précise et unique dans un seul type de situation comportementale. Par exemple, la règle *il ne faut pas prescrire d'aspirine à une patiente enceinte* peut être appréhendée selon quatre vues différentes :

- comme une règle de structure : il ne doit pas exister de relations entre une patiente enceinte et un traitement composé d'aspirine ;

- comme une règle d'évolution : la prescription d'aspirine ne peut être réalisée durant l'intervalle temporel de l'état *patiente enceinte* ;

- comme une règle de contrôle : il est nécessaire de contrôler qu'une patiente n'est pas enceinte avant de lui prescrire de l'aspirine ;

- comme une règle d'exception : si une patiente enceinte a un traitement composé d'aspirine, il faut immédiatement changer son traitement.

C'est donc au concepteur de l'application de décider quelle est plus précisément la fonction d'une règle et de déterminer ainsi à quel type de situation comportementale elle appartient.

Cependant, ce problème nous amène à penser que tous les types de situations comportementales peuvent être regroupés à l'intérieur d'un modèle plus général : le modèle **Situation-Réaction**. En effet, les types de situations comportementales représentent des réactions à des situations attendues ou non. Réagir à une situation attendue a pour but de participer au bon déroulement de l'application et d'atteindre des objectifs. C'est le cas des règles d'activité ou des règles de contrôle. Au contraire, une situation inattendue peut impliquer un mauvais fonctionnement de l'application et doit donc être corrigée le plus tôt possible. C'est le cas des règles de structure ou des règles d'évolution. Les règles d'exception réagissent à des situations inhabituelles et ont aussi pour but de participer au bon déroulement de l'application.

Toutes les situations comportementales sont donc basées sur un modèle commun : le modèle **situation-réaction** : une **situation**, attendue ou non, engendre une **réaction** contribuant au bon fonctionnement de l'application.

Cette notion se rapproche de la notion de **règle de production** définie en intelligence artificielle. Une règle de production a la forme : *SI prémisse, ALORS conséquent* ou encore *Déclencheur* $\rightarrow$ *Corps* et est utilisée pour représenter les connaissances de manière déclarative. Parallèlement à cette notion s'est développée la notion de règle active de la forme *LORSQUE événement, SI condition, ALORS action* utilisée principalement dans les sytèmes de gestion de bases de données actifs, mais aussi dans le domaine du génie logiciel [EC94] [Cas94]. Une étude de technologies appartenant à divers domaines de l'informatique [Fro95d] montre que les systèmes intégrant l'une ou l'autre de ces deux notions sont une bonne cible pour implanter des situations

comportementales des applications, en particulier :

- les langages de programmation logique [AEFdC90] [Lal90] tels Prolog [Bra88] [GKPC85] [SS91],
- les systèmes à base de connaissance [BLR92] [Fro86] [GD93],
- les systèmes experts [CD87] [Far85],
- les systèmes de gestion de bases de données déductifs tels Datalog [Bid93],
- les langages de programmation pour bases de données tels Peplom [DR94] [Fro94],
- les systèmes de gestion de bases de données actifs [ACC+93] relationnels tels Starburst [WF90], Postgres [SHP89], A-RDL [SKdM92] ou encore Ariel [Han92] ou orientés objets tels Hipac [DBB+88], Sentinel [CAM93], Naos [CCS94] [Col97], Samos [GGD91] ou encore Ode [GJ91] [GJS92].

Avant d'apporter notre propre contribution à une expression plus formalisée de telles situations comportementales, nous réalisons dans le chapitre suivant une analyse des techniques utilisables (et pour certaines utilisées professionnellement) dans ce contexte.

Chapitre 3

Approches classiques de l'ingénierie des applications

Ce chapitre a pour but de traiter de la façon dont les situations comportementales mises en évidence dans le chapitre 2 sont actuellement prises en compte au niveau conceptuel dans diverses approches de modélisation. Pour cela, nous rappelons tout d'abord les notions essentielles de la conception de systèmes d'information. Puis nous analysons la façon dont le comportement est représenté dans des méthodes traditionnelles d'analyse et de conception orientées objets.

3.1 La conception de systèmes d'information

La notion de système d'information (SI) est depuis de nombreuses années à la base de la conception d'applications. Dans une application, le système d'information est le système intermédiaire entre le système décisionnel et le système opérationnel, comme le montre le triangle systémique de J.L. Le Moigne [Moi77]. Le système opérationnel permet de réaliser les différentes fonctions de l'organisation, alors que le système décisionnel permet de prendre des décisions. Pour fonctionner correctement, ces deux systèmes doivent s'appuyer sur une information cohérente et à jour : c'est le rôle du système d'information que de garantir la fiabilité et l'adéquation de l'information.

Dès les années 65, on commence à automatiser les systèmes d'information dans le but de gérer des tâches lourdes et répétitives comme la gestion de stocks, la facturation ou la gestion des paies. Dix ans plus tard, les systèmes d'information automatisés font leur apparition dans le domaine des bases de données centralisées dans un but de documentation ou d'aide à la décision. A partir de 1985, ils intègrent peu à peu

les usines et les bureaux où ils sont utilisés pour des applications non seulement de bureautique, mais aussi de télématique et de productique. Enfin, les systèmes d'information automatisés deviennent "experts" et "grand public" et touchent aussi bien les domaines des bases de connaissance, l'imagerie, les messageries ou tout ce qui a trait au multimédia et à l'hypermédia. Colette Rolland [RFB88] définit un système d'information d'une organisation comme "un ensemble formé :

- de collections de **données**, représentations partielles, en parties arbitraires mais nécessairement opératoires, d'aspects pertinents de la réalité de l'organisation sur lesquels on souhaite être renseigné. Ces collections inter-reliées, aussi cohérentes que possible, sont mémorisées et communiquées dans le lieu, le moment et la présentation appropriés aux acteurs qui en ont l'usage ;
- de collections de **règles** qui fixent le fonctionnement informationnel. Ces règles traduisent ou sont calquées sur le fonctionnement organisationnel. Partie intégrante du SI, elles doivent être connues des acteurs qui utilisent le SI et leur sont nécessaires pour l'interprétation et la manipulation des collections de données ;
- d'un ensemble de **procédés** pour l'acquisition, la mémorisation, la transformation, la recherche, la communication et la restitution des renseignements ;
- d'un ensemble de **ressources humaines** et de **moyens techniques** intégrés dans un système, coopérant et contribuant à son fonctionnement et à la poursuite des objectifs qui lui sont assignés."

Cette définition reste tout à fait adéquate aujourd'hui avec les nouvelles technologies.

3.1.1 L'ingénierie de systèmes d'information

Les éléments d'un système d'information, appelés ici objets, peuvent être de deux niveaux d'abstraction différents : conceptuels ou logiciels. Les objets conceptuels apparaissent dès l'analyse des besoins de l'application, alors que les objets logiciels apparaissent lors de son implantation dans un système cible. De plus, il y a correspondance entre les deux types d'objets puisque les objets logiciels représentent un ou plusieurs objets conceptuels.

L'ingénierie d'un système d'information, que l'on peut tout d'abord résumer en trois phases (l'analyse, la conception et l'implantation) a pour but d'implanter les objets conceptuels grâce aux objets logiciels les plus adéquats. La phase d'analyse part

d'un problème du monde réel et dresse une représentation du domaine du problème. Cette représentation est le point de départ de la phase de conception qui aboutit à une représentation du domaine de la solution. Enfin, la phase d'implantation traduit le domaine de la solution dans un système technologique.

D'un point de vue ingénierie, ces trois phases peuvent être placées à l'intérieur de l'architecture de deux processus communiquants, "l'ingénierie des besoins" et "l'ingénierie des systèmes" . Dans cette architecture proposée par Colette Rolland [Rol93], le schéma conceptuel de l'application joue un rôle central car il articule la transformation des besoins de l'application en une implantation concrète (cf. figure 3.1).

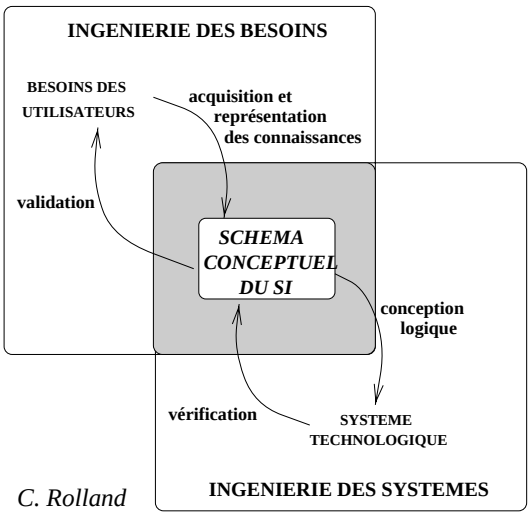


FIG. 3.1 – *Ingénierie d'un système d'information [Rol93]*

L'analyse, la conception et l'implantation sont trois phases du **cycle de vie** d'un logiciel, notion centrale en génie logiciel dans la construction d'un logiciel. Différents modèles de cycles de vie ont été proposés durant ces vingt dernières années parmi lesquels les cycles de vie en cascade [Roy70], en V [GMSB96], en spirale [Boe86] ou plus récemment en fontaine [HSE90]. Le développement d'une application en suivant un cycle de vie permet d'adopter une démarche de conception tout au long du développement de l'application.

Longtemps oubliée, la modélisation des processus de développement constitue de nos jours un développement de recherche à part entière. En particulier, Colette Rolland accorde une grande importance au suivi du processus de développement d'une application en fixant un cadre de référence composé de quatre mondes [Rol97] : le monde du sujet, le monde du système, le monde de l'usage et le monde du développement. Elle

insiste sur le fait que toute méthode associe un modèle de processus (la démarche) à un modèle de produits et qu'il est donc nécessaire de toujours considérer un modèle de produits et un modèle de processus et de représenter le lien existant entre ces deux modèles.

3.1.2 Les approches classiques

Afin d'aider les concepteurs à construire une application tout en respectant au maximum les étapes imposées par un cycle de vie, de nombreuses méthodes ont été mises au point depuis plusieurs années.

3.1.2.1 Historique des méthodes

Les premières méthodes datent du début des années 60 et sont dites **d'analyse**. Des méthodes comme CORIG, PROTEE ou encore ARIANE cherchaient à standardiser le métier de développeur d'application et initialisaient une approche industrielle d'informatisation.

Puis, les méthodes **cartésiennes** des années 1970 comme SADT ou SA ont introduit la notion de traitement permettant une décomposition hiérarchique fonctionnelle d'un SI. Elles mettaient en évidence la nécessité d'une démarche par étape basée sur une programmation structurée et modulaire.

Les méthodes **systémiques** des années 80 comme MERISE, IDA ou REMORA ont provoqué une rupture par rapport aux deux précédents types de méthodes : leurs notions de données, d'actions, d'événements et de relations entre les éléments privilégient une approche conceptuelle globale d'un SI.

Enfin, les méthodes **objets** des années 1990 sont une synthèse des méthodes systémiques et des méthodes cartésiennes. Elles permettent des spécifications détaillées des éléments d'un SI en introduisant la notion d'objet regroupant structures de données et traitements [CPFJF+97] et proposent des spécifications globales d'un SI par l'intermédiaire de spécifications statiques et dynamiques. Elles adoptent deux approches principales :

- une approche orientée logiciel, adoptée par les méthodes dites techniques comme HOOD, BON, MECANO, etc.,
- une approche plus orientée SI adoptée par des méthodes globales comme OOA,

OMT, OOD, OOSA, etc. C'est cette approche qui nous intéresse plus particulièrement dans ce travail [FFre].

Les méthodes objets étant de plus en plus nombreuses, des tentatives d'unification ont été effectuées. En particulier, le langage *UML* (*Unified Modeling Language*) [BR95] [BJR96] [Mul97] mis au point par James Rumbaugh, Grady Booch et Ivar Jacobson unifie les concepts présents dans les deux méthodes objets qui dominent actuellement le marché auprès des industriels (OMT [RBP+91] et OOD [Boo91]) en ajoutant la dimension des cas d'utilisation de OOSE [JCJO92].

3.1.2.2 Concepts généraux d'une méthode

D'après Colette Rolland [RFB88], toute méthode doit mettre en oeuvre quatre composantes indissociables et complémentaires :

- des **modèles** : un modèle est un ensemble de concepts et de règles pour les utiliser, destiné soit à expliquer et construire la représentation des phénomènes organisationnels, soit à expliquer et représenter les éléments qui composent le SI et leurs relations ;

- des **langages** : un langage est un ensemble de constructions qui permettent de décrire formellement les spécifications du SI élaborées aux différents stades du processus de conception en s'appuyant éventuellement sur le ou les modèles de la méthode ;

- une **démarche** : la démarche est le processus opératoire grâce auquel s'effectue le travail de modélisation, de description, d'évaluation et de réalisation du système d'information ;

- des **outils** : des outils logiciels appelés **Ateliers de Génie Logiciel** supportent la démarche. Ils peuvent être des outils de documentation, d'évaluation, de simulation ou d'aide à la conception ou à la réalisation.

Nous détaillons maintenant les concepts sous-jacents aux méthodes orientées-objets globales. Le lecteur souhaitant plus de détails sur les méthodes d'analyse, cartésiennes, systémiques ou encore objets "techniques" pourra se reporter à [Gir95].

3.1.2.3 Les méthodes à objets

Une méthode à objets est une méthode de construction de logiciels qui fonde l'architecture des systèmes sur les objets que ces logiciels manipulent et non sur la fonction qu'ils assurent. Un SI est vu comme un ensemble d'objets inter-reliés.

1. **Développement par objets**

   Le développement adopté dans les méthodes orientées objets suit les principes de base des cycles de vie des logiciels : analyse, conception et implantation. L'analyse transforme des spécifications informelles représentant les besoins des utilisateurs dans un modèle objet descriptif et normatif informatisable. Ce dernier est transformé lors de la phase de conception en un modèle objet effectif et normalisé, à partir duquel la phase d'implantation crée un logiciel utilisable [Ner92]. Les modèles objets résultats des phases d'analyse et de conception utilisent les principes de base et les concepts de la technologie objet.

2. **Concepts utilisés dans les modèles objets**

   La notion de base dans un modèle objet est l'**objet**. Les objets dont la structure et le comportement sont similaires sont regroupés dans des **classes**. Une classe, donc par conséquent un objet, possède trois dimensions principales : un aspect statique ou **structure**, un aspect dynamique ou **comportement** et un aspect fonctionnel ou **traitement**.

   Conformément aux dimensions structurelle, comportementale et fonctionnelle d'un objet, la majorité des méthodes orientées objets considère un système d'information selon trois vues complémentaires : une vue structurelle, une vue comportementale et une vue fonctionnelle. Ainsi, dans un hôpital, une partie du fonctionnement d'une unité médico-technique est modélisée par les trois vues structurelle, comportementale et fonctionnelle représentées dans la figure 3.2.

   Vue structurelle - Elle montre l'aspect statique d'un système d'information et organise le domaine du problème. Elle est essentiellement composée de **diagrammes statiques** parfois appelés **modèles d'objets**. Un diagramme statique regroupe un ensemble de classes d'objets. Chaque classe d'objets possède des **attributs** caractéristiques et peut réaliser des **méthodes**. Les classes d'objets sont reliées entre elles par le biais d'**associations**, de liens d'**héritage** et de liens de **composition**.

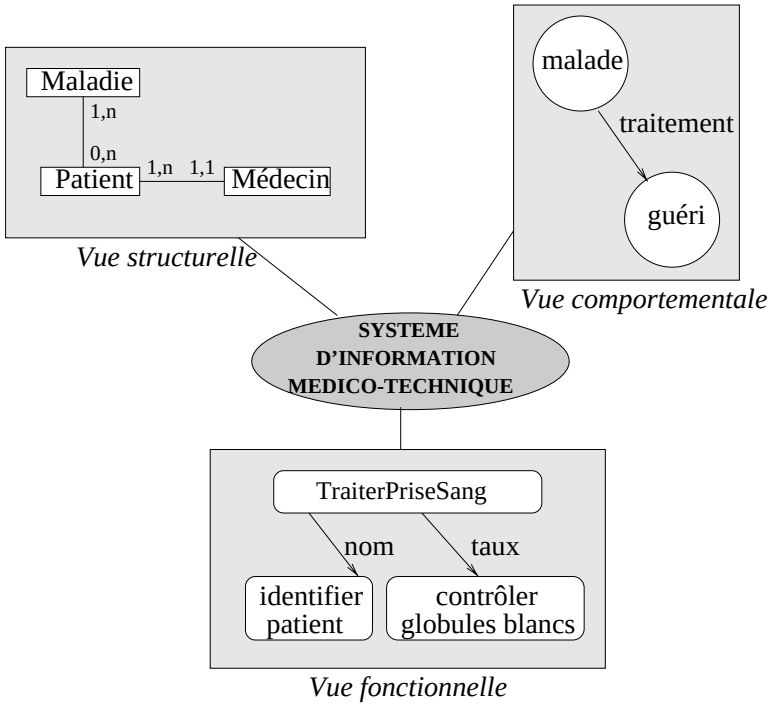


FIG. 3.2 – *Trois vues d'un système d'information médico-technique*

Vue comportementale - Elle organise l'utilisation des objets des classes, en représentant leur comportement et leur cycle de vie essentiellement dans des **diagrammes de transitions d'états** qui montrent les différents **états** dans lesquels un objet d'une classe peut être ainsi que les évolutions de cet objet au cours du temps. Le changement d'état d'un objet peut s'effectuer suite à un **événement** ou un **message** reçu par un objet d'une autre classe et caractérisant une demande de **service** de la part d'un autre objet. L'objet ayant reçu le message peut alors réagir en déclenchant une opération pour demander un service à un autre objet. Ce principe est basé sur **l'envoi de messages**. Il existe en général un diagramme d'états par classe et l'ensemble de tous les diagrammes d'états représente la dynamique globale du système.

Vue fonctionnelle - Elle donne le fonctionnement du système d'information global en montrant les échanges de données entre les classes d'objets. Elle est essentiellement composée de **diagrammes à flots de données** qui représentent les valeurs entrantes, les valeurs calculées et les valeurs sortantes. Des **traitements** peuvent être appliqués sur les données avec ou sans effet de bord ; des **flots de données** transportent des données sans les modifier entre un produc-

teur et un ou plusieurs consommateurs ; des **acteurs** produisent ou consomment des données ; des **réservoirs de données** stockent et rendent accessibles les données. Enfin, des **flots de contrôle** autorisent ou interdisent l'exécution de certains traitements.

3. **Principes de base de la technologie objet**

Encapsulation - Tout objet est vu comme une boîte noire ; on a accès seulement à ses entrées et sorties (le quoi) sans savoir ce qui se passe à l'intérieur (le comment).

Classification - La classification consiste en la recherche d'une structuration regroupant dans une même classe tous les objets dont les caractéristiques et le comportement sont similaires dans leur environnement.

Agrégation - L'agrégation apporte une structuration locale aux objets en mettant en évidence des compositions regroupant plusieurs objets en un seul.

Héritage - L'héritage ou généralisation/spécialisation consiste en la recherche d'un treillis de classes organisant hiérarchiquement la spécification d'un système.

Les situations introduites dans le chapitre 2 mettent en évidence un aspect comportemental, y compris dans le cas des règles de structure où l'application doit réagir afin de maintenir la structure des entités. C'est donc la vue comportementale qui nous intéresse plus particulièrement. Dans la suite de ce chapitre, nous analysons donc plusieurs techniques de représentation du comportement afin de voir comment les situations comportementales présentées dans le chapitre 2 pourraient être prises en compte dans les méthodes d'analyse et de conception orientées objets actuelles.

3.2 Le comportement dans les méthodes orientées objets

C'est avec les méthodes systémiques que la notion de comportement des entités de l'application apparaît : par une conception globale du SI, ces méthodes permettent la

prise en compte du comportement des applications en introduisant le concept d'événement. Mais ce n'est réellement que lors de l'apparition des méthodes orientées objets que de gros espoirs ont été fondés pour l'expression du comportement des applications : le concept d'**objet** [Ous97] en regroupant données et traitements, permet à la fois une spécification statique et une spécification dynamique d'un système d'information.

Or, force est de constater que les méthodes orientées objet n'ont pas véritablement tenu leur promesse : en effet, si elles ont su exprimer le comportement d'un objet en représentant son cycle de vie grâce aux notions d'événements et d'états, elles n'offrent pas réellement de moyens efficaces pour exprimer la dynamique d'un SI. Les méthodes orientées objets se concentrent plutôt à représenter le comportement local des objets, c'est-à-dire les réactions des objets des classes face à des événements. Par contre, le comportement global de l'application n'est pas exprimé grâce à des techniques efficaces et fiables. De nombreux travaux [CCC+96] [CPT96] [Mad96] cherchent à améliorer les techniques de représentation du comportement d'une application. Les diagrammes de séquence, de collaboration et d'activités sont développés dans ce sens [Lai97] [Mul97].

Dans la suite de cette section, nous faisons un choix pragmatique de méthodes couramment utilisées ou enseignées, ou explicitement destinées à la représentation du comportement. Nous synthétisons les techniques utilisées dans les modèles de ces méthodes pour représenter le comportement d'une application. Puis nous concluons quant aux atouts et aux limites de ces techniques dans la représentation des situations comportementales [Fro95a] [FGL95].

3.2.1 L'envoi de messages

L'**envoi de messages** est une notion de base dans les méthodes orientées objets. Il permet de représenter la communication soit entre des objets, soit entre un utilisateur et un objet. Un message est composé du nom de l'objet qui le reçoit, d'un sélecteur qui représente le nom de l'opération qui sera exécutée et de paramètres définis par la signature de l'opération cible. Après l'envoi d'un message par l'objet émetteur, l'objet récepteur définit quelle opération sera exécutée selon l'interface de sa classe ou d'une de ses sur-classes. Cette opération est appelée **service** : un service est un comportement spécifique exhibé sous la responsabilité d'un objet.

Dans la méthode OOA [CY90], les diagrammes représentent des envois de messages qui modélisent la dépendance d'exécution des services des objets en montrant à quels services externes un objet doit faire appel pour accomplir ses responsabilités. Dans

la figure 3.3, un médecin externe envoie un message à une unité médico-technique pour qu'elle examine des analyses (1). L'unité médico-technique sollicitée examine les analyses et envoie un message d'abord au médecin (2), puis au patient (3) afin de leur communiquer les résultats.

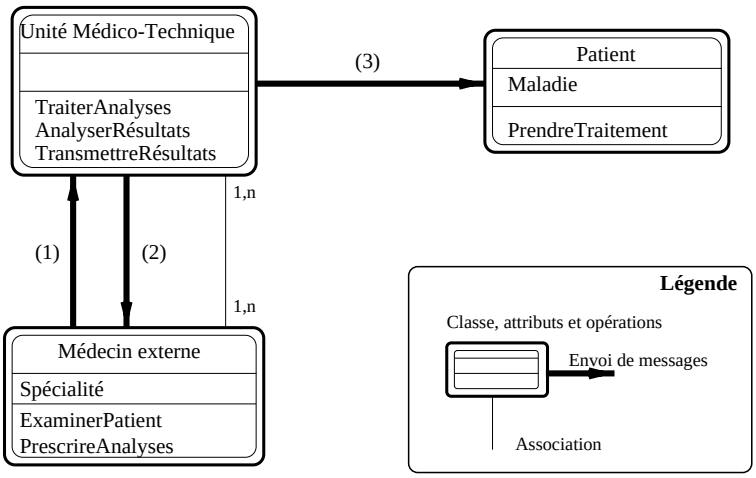


FIG. 3.3 – *Des connexions de messages OOA dans une unité médico-technique*

De la même façon, les méthodes OMT [RBP+91] et OOD [Boo91] introduisent des diagrammes d'interaction des objets pour montrer les interactions entre les objets par l'intermédiaire de messages et présenter la séquence de messages en rapport avec l'exécution d'une opération.

3.2.2 Les événements

Le terme d'**événement** est apparu avec les méthodes systémiques comme REMORA [RFB88] où Colette Rolland introduit la notion d'événement dans une conception intégrée de la structure et du comportement du SI : *un événement est quelque chose qui survient dans l'organisation ; il est vu comme une cause ou une condition de l'exécution d'opérations ou d'activités réalisées par des entités de l'application ou par des acteurs extérieurs et qui modifient l'état de l'organisation.* Plusieurs sortes d'événements existent :

- des événements temporels dont les définitions sont liées au temps et dont les arrivées sont causées par l'écoulement naturel du temps (fin de mois, début d'année),

- des événements aléatoires ou imprévisibles dont les causes ne sont pas connues (panne d'une machine, accident d'un employé, incendie d'un atelier),
- des événements liés au changement d'état des objets (l'arrivée de la commande 2850 qui correspond au passage de cette commande de l'état inexistant à l'état existant).

Cependant, il est possible de généraliser ces trois catégories d'événements et de rapprocher tout événement du changement d'état d'un ou de plusieurs objets. On dit alors qu'un événement est la constatation d'un changement d'état d'un ou plusieurs objets par un détecteur. Il déclenche l'exécution d'une ou plusieurs opérations (la fin du mois déclenche le calcul des salaires des employés) et provoque ainsi un nouveau changement d'état. Le déclenchement peut être conditionnel ou itératif.

Un événement a des propriétés qui permettent de le situer dans le temps et l'espace, par exemple la date et le lieu où il se produit. De plus, il a un identifiant et appartient à une classe d'événements. Une classe d'événements est un ensemble d'événements constatant des changements d'états de même nature, d'objets ou d'associations de mêmes classes déclenchant des opérations aux effets semblables et définis par des propriétés identiques par leur sémantique.

Dans REMORA, l'ensemble des événements est représenté dans un cycle dynamique avec les opérations déclenchées, les conditions de déclenchement et les facteurs de répétition (cf. figure 3.4).

De nos jours, la notion d'événement est présente dans les méthodes orientées objets et est à la base de l'expression du comportement des applications. L'importance qu'elle revêt ainsi que la façon dont elle est prise en compte varient d'une méthode à l'autre, ce qui se traduit par des techniques différentes comme les scénarios d'événements, les diagrammes d'états, les modèles de traitement et les modèles événementiels.

3.2.3 Les scénarios d'événements

Un scénario d'événements représente une séquence d'événements se déroulant durant une exécution particulière d'un système. La portée d'un scénario peut varier : il peut inclure tous les événements du système, ceux qui entrent en conflit ou encore ceux produits par certains objets. Un scénario peut être un enregistrement historique de l'exécution d'un système ou l'expérimentation d'exécution du système proposé.

La figure 3.5(a) montre un scénario de traitement décrivant une demande de prise

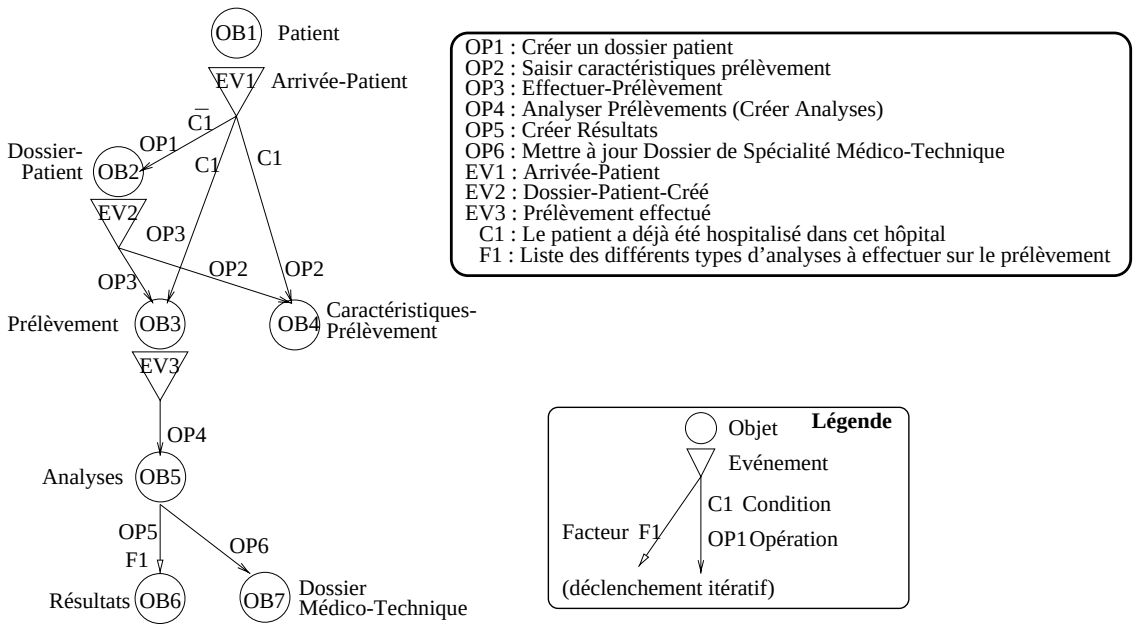


FIG. 3.4 – *Un cycle dynamique en REMORA*

de sang dans la méthode OMT [RBP+91]. Ce scénario est complété dans la figure 3.5(b) par les objets et les échanges d'événements dans un "diagramme de suivi des événements". Les objets sont représentés par des lignes verticales et les événements par des flèches entre l'objet qui envoie l'événement et celui qui le reçoit.

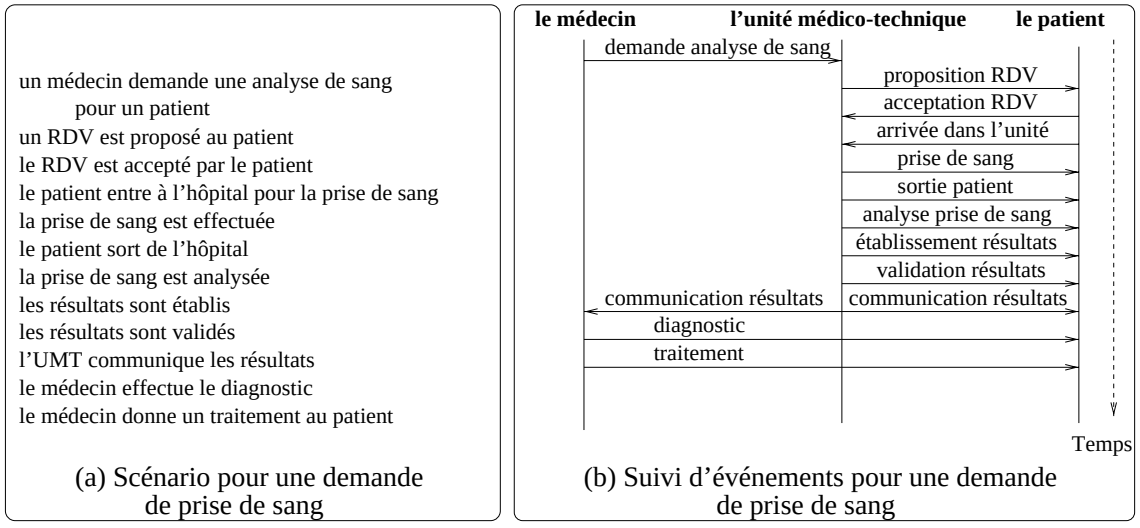


FIG. 3.5 – *Scénario et suivi d'événements en OMT*

Notons enfin que les diagrammes de temps ou **chronogrammes** proposés par cer-

taines méthodes telles OOD [Boo91] permettent d'obtenir de l'information sur les événements externes asynchrones, l'ordre de déclenchement des opérations sur les différents objets en interaction et l'ordre d'arrivée des messages transmis.

3.2.4 Les diagrammes de transitions d'états

La majorité des méthodes orientées objets (OOA, OOD, OMT, etc) basent l'expression du comportement des applications sur les diagrammes de transitions d'états des classes de leur modèle statique. Cette notion provient de la proposition des "statecharts" de Harel [Har87]. Un *diagramme de transitions d'états* [Boo91] est aussi appelé *diagramme d'états* [RBP+91] ou *diagramme de l'histoire des objets* [CY90].

Un diagramme d'états donne le comportement dynamique des objets de certaines classes. C'est un graphe dont les nœuds sont des états et dont les arcs sont des transitions entre les états provoquées par des événements. Les **événements** représentent des stimuli externes ; les **états** représentent les valeurs d'attributs et de liens pris par les objets. Les événements surviennent à un instant donné et provoquent des changements d'états des objets. Les diagrammes d'états de différentes classes interagissent grâce à des événements partagés.

Les concepts d'états et d'événements sont les concepts minimaux d'un diagramme d'états. Cependant, des concepts supplémentaires sont introduits par certaines méthodes, en particulier par OMT (cf. figure 3.6).

- Des **conditions** portant sur les valeurs des objets sont utilisées comme garde sur les transitions. Une **transition gardée** se déclenche quand son événement apparaît, mais seulement si la condition est vérifiée.

- Des **opérations** attachées à des états ou à des transitions sont effectuées en réponse aux états ou aux événements correspondants. On distingue les **actions** (opérations instantanées en réponse à un événement, par exemple envoi d'un événement à un autre objet, mise à jour d'attributs, etc) et les **activités** (séquences d'actions non instantanées associées à un état).

- Enfin, des **actions d'entrée** et des **actions de sortie** sont effectuées en entrant ou en quittant un état. De la même façon, des **actions internes** sont réalisées à l'intérieur d'un état et n'impliquent aucun changement d'état. Enfin, des **transitions automatiques** se déclenchent quand leurs conditions sont satisfaites.

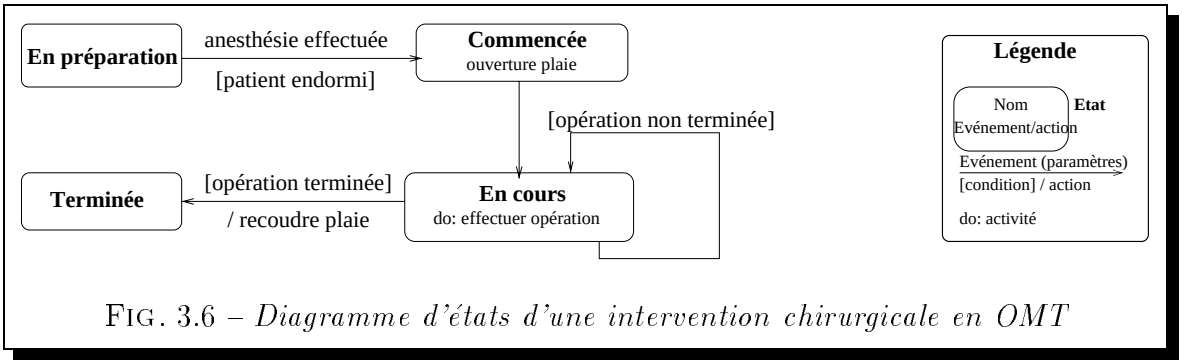


FIG. 3.6 – *Diagramme d'états d'une intervention chirurgicale en OMT*

3.2.5 Les modèles de traitement

Nous appelons modèles de traitement des modèles généraux destinés à exprimer le comportement global des applications. Les modèles de traitement sont composés des acteurs intervenant dans le système d'information, des événements auxquels ce dernier réagit, des opérations qu'il déclenche en réaction aux événements, des événements engendrés par ces opérations et des enchaînements de réactions entre les différents acteurs. Les **cycles dynamiques** de la méthode systémique REMORA dont on a déjà parlé en sont un exemple et récemment, l'approche des cas d'utilisation [JCJO92] constitue un modèle de traitement centré catégories d'utilisateurs.

La méthode MERISE [TRC83] [NECH92] introduit les **Modèles Conceptuels de Traitements** (MCT) pour exprimer les besoins dynamiques et les traitements de l'application : dans Merise, le traitement s'assimile au fonctionnement du système d'information à travers ses couplages avec le système opérant et le système de pilotage. Décrire les traitements, c'est décrire les processus déclenchés dans le domaine en réponse aux stimulations de son environnement. Un MCT a donc pour objectif de représenter formellement les activités exercées par le domaine. Il exprime ce que fait le domaine et non par qui, quand, où et comment les activités sont réalisées. Il précise les frontières du domaine en décrivant les activités qui lui sont associées et les échanges avec son environnement. Enfin, il permet une simulation de l'activité du système d'information : fonctionnement pas à pas, mise en évidence de conflits et de parallélismes, etc. Un MCT (cf figures 2.2 et 3.7) comporte les concepts suivants :

- les acteurs externes au domaine,
- l'événement/résultat-message : les flux reçus (stimuli) et émis (réactions) par le domaine sont respectivement modélisés en événements et résultats, externes ou internes. Un **événement** caractérise le fait qu'il s'est produit quelque chose demandant une réaction du système ; il est émis par un acteur à destination du

domaine. Un **résultat** est la formalisation d'une réaction du domaine et de son système d'information ; il est émis par une activité du domaine à destination d'un acteur. A un événement ou à un résultat sont éventuellement associés des ensembles d'informations appelés **messages**. Un message est un ensemble structuré d'informations décrivant un événement ou un résultat type. Une occurrence d'événement ou de résultat doit être distinguable des autres par le contenu de son message ainsi que par l'instant ou l'endroit où il se produit ;

- l'opération : c'est la description du comportement du domaine et de son système d'information par rapport aux événements types. Elle est déclenchée par la survenance d'un ou de plusieurs événements synchronisés. Elle est composée de l'ensemble des activités que le domaine peut effectuer à partir des informations fournies par le ou les événements et de celles déjà connues dans la mémoire du système d'information. Elle émet en retour des résultats dont l'émission est soumise à des conditions traduites par des expressions logiques ;

- la synchronisation : c'est une liste d'événements qui doivent s'être produits avant qu'une opération soit déclenchée. Elle se traduit par une expression logique s'appliquant sur la présence des occurrences d'événements sollicitant l'opération. Cette contrainte de synchronisation est généralement complétée par une condition locale portant sur le contenu informationnel des occurrences d'événements. Si elle est vérifiée, l'opération peut démarrer et les occurrences déclencheuses sont consommées par l'opération. Sinon, la synchronisation et les occurrences d'événements présents restent en attente jusqu'à vérification.

Les concepts d'un MCT sont interprétés en terme de réseaux de Petri [Bra83] [TSI85] qui permettent de mettre en évidence des **conflits** et des **cycles**. Il y a un conflit sur un événement E si celui-ci contribue à n synchronisations. Deux solutions sont alors possibles pour résoudre ce problème. Tout d'abord, la capacité de production de E peut être n (dans ce cas, les n jetons sont consommés dans les n synchronisations). Sinon, les conditions de participation de E aux n synchronisations peuvent être exclusives. Il y a un cycle lorsqu'une synchronisation a comme élément contributif un événement dont elle déclenche elle-même l'émission directement ou à travers plusieurs opérations. Dans ce cas, il faut prévoir des événements début et fin pour démarrer ou arrêter le cycle.

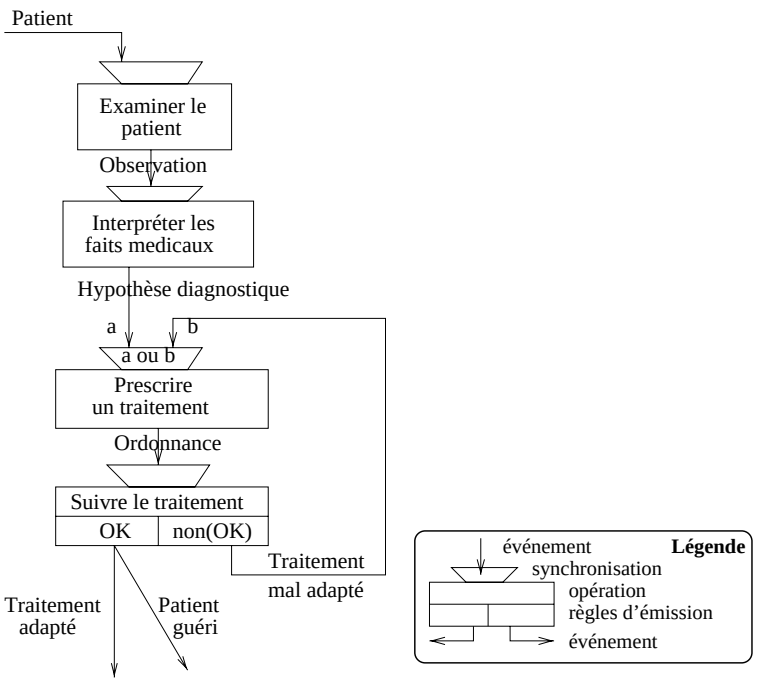


FIG. 3.7 – *Examen d'un patient avec un MCT Merise*

3.2.6 Les modèles événementiels

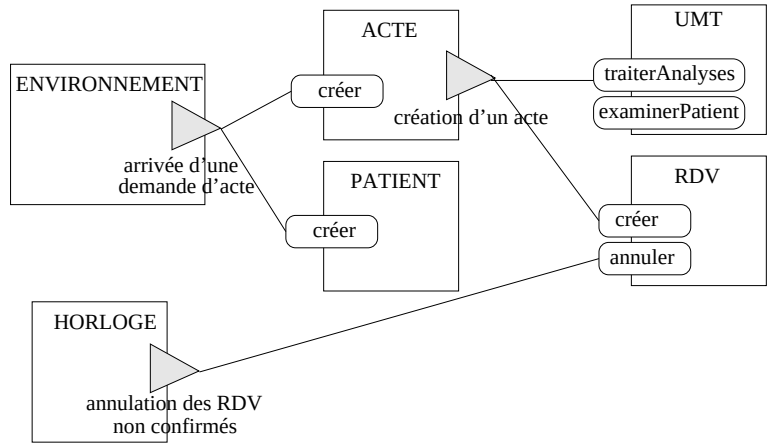
Les modèles événementiels mettent en exergue le concept d'événement pour exprimer et concevoir le comportement des applications. La notion d'événement peut soit être liée à celle d'objet comme dans O* [Bru93], soit en être indépendante comme dans IFO2 [Tei94].

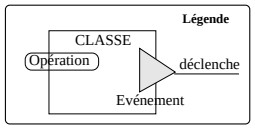
1. **O***

   Dans O*, Joël Brunet [Bru93] propose d'introduire le concept d'événement dans le paradigme orienté objet afin d'autoriser la description des aspects dynamiques liés à la causalité et à la concurrence. Il reprend le concept d'événement de type procédural introduit dans REMORA [RFB88] qui lui permet d'utiliser simultanément le concept d'opération pour exprimer comment un objet évolue (le quoi) et le concept d'événement pour exprimer quand un objet évolue (le pourquoi). La causalité (inférence d'événements) est spécifiée par l'intermédiaire d'événements internes aux objets qui se produisent à la suite de changements d'états particuliers. La concurrence (synchronisation d'événements) est prise en compte dans la définition même de l'événement qui déclenche simultanément un ensemble d'opérations.

Au même titre que l'opération, l'événement est vu comme un lien dynamique entre les objets. Un événement décrit un ensemble d'occurrences d'événements de même nature. Cette description comprend la cause de la survenance d'une occurrence d'événement et son impact sur les objets du système d'information. Un événement est ainsi caractérisé par sa condition d'occurrence et l'emplacement dans lequel il est défini dépend de sa condition d'occurrence.

– Un événement **interne** constate le changement d'état remarquable d'un objet et est défini dans la classe de l'objet dont il constate le changement d'état (par exemple, l'événement Création d'un acte de la figure 3.8) ; sa condition d'occurrence est décrite par un prédicat pouvant porter sur les attributs et sur les états d'une unique classe.





classe ACTE
événements
création d'un acte
prédicat
déclenchement
créer ***sur*** RDV
traiterAnalyses ***sur*** UMT ***si*** la demande d'acte porte sur des analyses
examinerPatient ***sur*** UMT ***si*** la demande d'acte porte sur un patient

classe HORLOGE
événements
annulation des RDV non confirmés
prédicat
chaque jour à 8h00
déclenchement
annuler ***sur*** RDV
pour (tous les RDV non confirmés / date de RDV < date du jour + 3), identité(RDV))

classe ENVIRONNEMENT
événements
arrivée d'une demande d'acte
message
structure(ACTE)
déclenchement
créer ***sur*** ACTE
créer ***sur*** PATIENT ***si*** le patient n'existe pas

FIG. 3.8 – *Des événements en O**

- Un événement **temporel** survient à un instant déterminé à l'avance et sa condition d'occurrence s'exprime par un prédicat temporel. Il est défini dans une classe particulière nommée HORLOGE qui regroupe tous les événements temporels du SI (cf. figure 3.8).
- Un événement **externe** a pour origine un stimulus en provenance de l'environnement du système d'information et est associé à un message décrivant sa condition d'occurrence et le type des paramètres transmis par l'environnement. Il est défini dans une classe particulière nommée ENVIRONNEMENT qui regroupe tous les événements externes du SI (cf. figure 3.8).

Comme pour REMORA, l'impact d'une occurrence d'événement sur les objets du système d'information est décrit par l'ensemble des opérations qu'elle déclenche. A une occurrence d'événement est associé un contexte qui représente l'information nécessaire à l'exécution des opérations déclenchées : le contexte d'une occurrence d'événement interne est l'objet sur lequel un changement d'état est constaté ; le contexte d'une occurrence d'événement temporel relatif est celui de l'événement initiateur (il est vide si l'événement est absolu) ; enfin, le contexte d'une occurrence d'événement externe est l'ensemble des paramètres du message.

Enfin, un événement peut être hérité : un événement défini dans une classe spécialisée ayant même nom qu'un événement de la classe généralisée est appelé événement spécialisé ; il hérite du prédicat de l'événement généralisé correspondant, lequel est ajouté à son propre prédicat. Un événement spécialisé ne peut survenir qu'additionnellement à l'événement généralisé correspondant.

2. **IFO2**

Au contraire de la méthode O*, la méthode IFO2 [Tei94] adopte une approche tout-événement pour exprimer le comportement d'une application alors que l'aspect statique du système d'information est décrit indépendamment de son aspect dynamique. Spécifier le comportement d'un système en IFO2 impose au programmeur d'applications une autre manière d'aborder le comportement. En effet, en mettant un accent tout particulier sur les enchaînements d'apparitions des événements, IFO2 offre des mécanismes permettant une véritable organisation des réactions du système. Le principe de base d'IFO2 est que tout fait pertinent par rapport au comportement modélisé est représenté comme un événement. Les événements sont identifiés, ont une valeur dépendante de leur type et sont caractérisés par un ensemble de paramètres représentant les objets ou les entités réagissant à ou concernés par les événements examinés. Les types d'événements

peuvent être de base (ils modélisent des événements élémentaires) ou complexes (ils expriment des conditions de synchronisation entre événements). Les types d'événements sont organisés au sein de **fragments** (cf. figure 3.9) en spécifiant les relations de causalité existant entre eux. Les fragments décrivent les réactions de l'application dans une situation particulière et proposent des modules de spécification qu'il est possible de réutiliser ou de compléter à volonté. Ces fragments sont regroupés au sein de **schémas événementiels** offrant une vision globale du comportement.

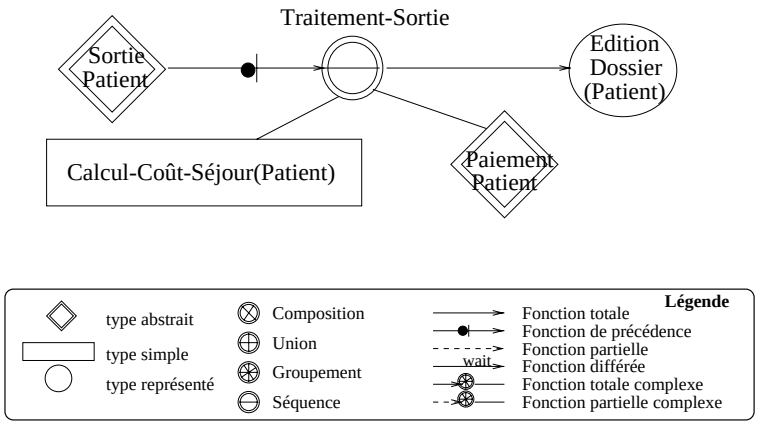


FIG. 3.9 – *Un fragment IFO2 pour le traitement de la sortie d'un patient*

Trois types d'événements de base sont proposés par le modèle : simple, abstrait ou représenté. Les types simples représentent l'invocation d'une méthode spécifiée pour un type d'objet structurel, un ordre commit ou rollback. Une distinction est établie entre les opérations bases de données (création ou destruction d'objet, ...), les opérations observables par l'utilisateur (du type requête) et les autres méthodes. Les types abstraits modélisent les événements externes et temporels émanant de l'environnement de l'application. Ils sont également utilisés pour représenter des événements internes au système. Enfin, les types représentés participent à la réutilisabilité des spécifications en permettant de symboliser tout autre type, simple ou complexe, décrit par ailleurs dans les spécifications.

L'expression des conditions de synchronisation s'appuie sur l'utilisation de constructeurs permettant de combiner non seulement des événements, mais aussi des actions. Les constructeurs proposés sont la *composition* qui reflète la conjonction d'événements de différents types, la *séquence* qui inclut un ordre chronologique entre les occurrences des événements composants, le *groupement* qui exprime des conjonctions d'événements de même type et l'*union* qui représente la disjonction

d'événements.

Les relations de causalité entre événements sont exprimées par des fonctions au sein des fragments. Les fonctions prennent en compte des conditions globales sur l'enchaînement des événements, en combinant les caractéristiques suivantes :

- *simples* ou *complexes*, i.e. un événement de leur origine déclenche un ou plusieurs événements de leur cible ;
- *partielles* ou *totales*, i.e. un événement de leur origine peut ou doit déclencher un événement de leur cible ;
- *différées* ou *immédiates* s'il existe ou non un délai entre les occurrences des événements de leur origine et de leur cible.

Enfin, une distinction est établie entre les fonctions de *déclenchement* et de *précédence*. Ces dernières sont particulièrement utiles si le type cible est externe ou temporel, pour exprimer des contraintes sur l'existence d'autres événements.

3.2.7 Les règles de comportement

La notion de règle a été introduite récemment dans les méthodes d'analyse et de conception. Exprimées dans un langage proche de la langue naturelle, les règles permettent d'appréhender le comportement d'une application de la même manière que le ferait un utilisateur. Nous présentons ici deux approches où les règles sont utilisées pour exprimer le comportement des applications.

1. **Des règles pour spécifier les besoins et les exigences d'une entreprise dans la méthode OOAD**

   La classification de règles destinées à spécifier le comportement et les exigences d'une entreprise que nous avons présentée dans la section 2.2.1 a d'abord été proposée de manière informelle par James Odell [Ode93a], puis a été introduite dans la méthode OOAD [MO96] pour suppléer des techniques telles que les diagrammes objets, les diagrammes de transitions d'états, les diagrammes d'événements, etc. [Ode93b]. Par exemple, la règle de contrainte Il faut toujours que le salaire d'un interne soit inférieur au salaire de son chef de service exprime une contrainte entre les salaires de deux employés d'un hôpital et est attachée à l'attribut salaire de la classe Employé (cf. figure 3.10).

   En particulier, les règles de stimuli-réponse de la forme *si... alors...* peuvent être combinées avec la notion de **règle de déclenchement** introduite dans

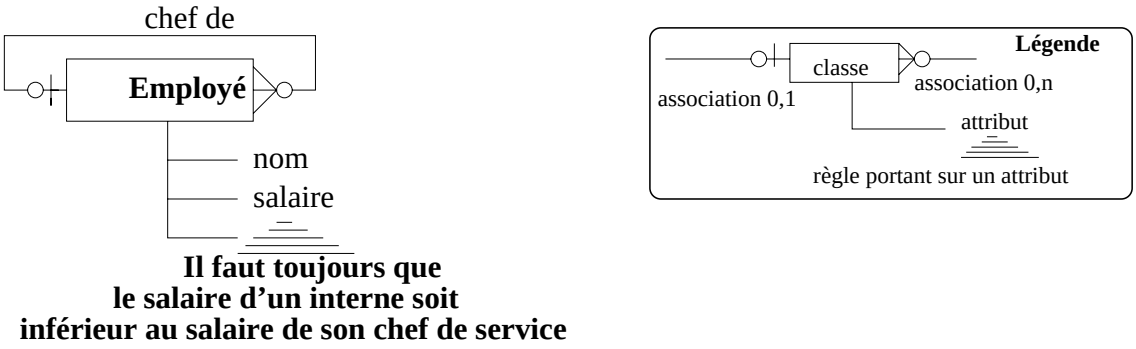


FIG. 3.10 – *Une règle de contrainte attachée à un attribut en OOAD*

la méthode OOAD pour représenter le comportement des applications. Dans cette méthode, le comportement des objets est représenté en termes d'opérations, d'événements, de conditions de contrôle et de règles de déclenchement.

- Une **opération** est une unité de traitement responsable d'un changement d'état sur un objet.
- Un **événement** est l'achèvement d'une opération invoquée.
- Des **conditions de contrôle** permettent de vérifier que certaines conditions sont vraies avant le déclenchement d'un traitement.
- Enfin, une **règle de déclenchement** lie la cause à l'effet : elle invoque une opération spécifique quand un type spécifique d'événements apparaît et détermine les arguments nécessaires pour exécuter l'opération invoquée.

Chaque règle de déclenchement possède trois éléments (cf. figure 3.11) : un type d'événement (la cause), une opération (l'effet) et une fonction. Le rôle de la fonction est de passer comme arguments à l'opération invoquée, les objets résultants de l'événement. Ainsi, la fonction crée une chaîne de cause à effet entre les événements et les opérations. Les fonctions sont dites "triviales" quand l'objet résultant de l'événement est directement passé à l'opération invoquée (l'objet manipulé par les opérations est alors le même) et "spécifiques" quand les opérations manipulent différents objets pour invoquer les opérations déclenchées. Un exemple de fonction triviale est donné dans la figure 3.11.

2. **Des règles de gestion dans la méthodologie IDEA**

La méthodologie IDEA [CF97] est destinée à la conception d'applications à l'aide d'objets et de règles et introduit la notion de **règles de gestion** (*Business Rule*).

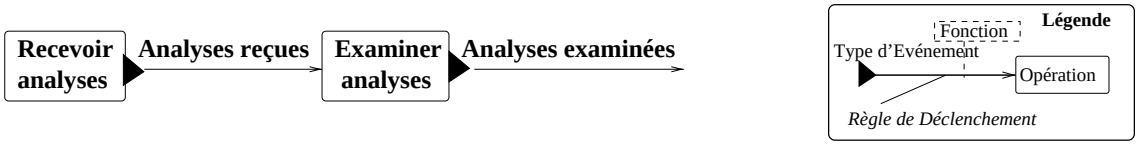


FIG. 3.11 – *Une règle de déclenchement OOAD avec fonction triviale*

Ces règles répondent à des besoins de l'application en modélisant la réaction à des événements externes du monde réel. Elles correspondent à une tâche applicative claire puisqu'il est possible en particulier de les associer à une métrique qui mesure l'amélioration de l'objectif de la tâche.

Dans la méthodologie IDEA, les règles de gestion apparaissent dans la phase de conception ; la stratégie de conception des règles de gestion est la suivante :

(a) identifier les tâches applicatives pour des règles actives ; associer chaque tâche à une condition sous laquelle elle doit être exécutée ;

(b) détecter pour chaque tâche les événements qui causent son exécution ; identifier pour chaque tâche une métrique qui indique le "progrès" à réaliser pour aboutir à la solution de la tâche ;

(c) générer les règles actives répondant aux événements associés à la tâche ; le concepteur doit constamment vérifier que les règles exécutées améliorent la métrique et permettent de progresser vers l'aboutissement de la tâche.

3.2.8 Bilan

3.2.8.1 Prise en compte du vocabulaire du domaine

Les figures 3.12 et 3.13 montrent comment chacun des concepts du vocabulaire du domaine (cf. section 1.2) est pris en compte par les méthodes étudiées.

3.2.8.2 Avantages et limitations des techniques

Après un fort intérêt des méthodes d'analyse et de conception pour les aspects statiques d'un SI, de nombreux efforts ont porté sur la représentation du comportement des applications. Motivées par le concept d'objet regroupant structure et comportement des entités de l'application, les méthodes ont alors proposé nombre de techniques

Technique	UML	OOA	OOD	OMT	OOAD
Evénement	Une occurrence qui engendre un changement d'état.		Une certaine occurrence provoquant la modification de l'état du système.	La transmission d'information asynchrone d'un objet vers un autre.	Un changement dans l'état d'un objet (création, terminaison, classification, déclassification, connexion, déconnexion).
Etat	une condition instantanée dans laquelle se trouve un objet.	L'identification de valeurs d'attributs qui reflètent un changement dans le comportement de l'objet.	Le résultat cumulé du comportement d'un objet (les propriétés de l'objet avec leurs valeurs courantes).	Une période de temps pendant laquelle un objet attend qu'un événement se produise.	La collection des associations qu'un objet a avec d'autres objets.
Condition	Une expression booléenne qui valide ou non une transition dans un automate d'états finis.	Porte sur le changement d'état d'un objet.	Sur le déclenchement d'un événement.	Porte sur une transition d'états.	Une condition de contrôle = un mécanisme associé à une opération lui permettant de commencer seulement si certaines contraintes sont vérifiées (permet la synchronisation d'opérations).
Action	Une opération exécutée instantanément lors d'une transition d'un état vers un autre ; ne peut être interrompue.	Exécutée lors d'un changement d'état.	Une opération qui s'exécute immédiatement : appel de méthode, déclenchement d'un événement, démarrage ou arrêt d'une activité.	Une opération instantanée invoquée lors d'une transition d'états ou à l'entrée ou à la sortie d'un état.	Une opération = un processus qui exécute pas à pas l'interrogation d'un objet ou le changement d'état d'un objet.
Règle					Une règle = une déclaration, un principe ou une condition qui doit être satisfait (contraintes ou dérivation) ; Une règle de déclenchement : invoque une opération particulière lorsqu'un événement se produit.
Situation					
Réaction					

FIG. 3.12 – *Le vocabulaire du domaine dans les méthodes*

Technique	Merise	REMORA-O*	IFO2	IDEA	Synthèse
Evénement	Un stimulus par lequel le domaine, puis son SI, prend connaissance des comportements de son environnement.	Quelque chose qui survient dans l'organisation (lié aux changements d'états des objets).	Un déclencheur d'une méthode ; externe, temporel ou générateur d'autres événements.	Toute occurrence notifiable à un objet : manipulation de la base de données, invocation d'un service ou tout objet du monde réel.	Notion similaire dans toutes les approches, bien que plus ou moins proche du monde réel.
Etat		Un événement décrit un type de changement d'état élémentaire particulier déclenchant des types d'actions élémentaires.		Une représentation abstraite de l'histoire passée d'un objet.	Apparaît essentiellement dans les diagrammes de transitions d'états des méthodes orientées objets.
Condition	Une synchronisation = une condition préalable au démarrage d'une opération, s'appliquant sur la présence ou l'absence des occurrences d'événements sollicitant l'opération.	Porte sur le déclenchement des opérations.	Porte sur les enchaînements entre événements.	Une précondition = une formule booléenne sur le déclenchement d'une transition d'état.	Dans la majorité des cas, conditionne le déclenchement d'une transition d'état.
Action	Une opération = une description du comportement du domaine et de son SI par rapport aux événements.	Une opération = une action qui peut être exécutée isolément dans l'organisation et qui modifie l'état de ses objets.		La production d'un événement symbolique, invocation d'une méthode, exécution de la manipulation d'une donnée, ou tout autre action dépendante de l'application.	En principe, instantanée ; apparaît dans les diagrammes de transitions d'états ou les modèles de traitement.
Règle				Règle de gestion.	Très peu présent dans les méthodes ; apparaît dans les nouvelles méthodes.
Situation					Non pris en compte.
Réaction	Un résultat = la formalisation d'une réaction du domaine et de son SI à un événement.	Un événement = un stimulus pour l'organisation qui sollicite la réaction de celle-ci.	Un événement = une représentation de faits participant aux réactions du système modélisé.		Non pris directement en compte, mais apparaît par rapport à un événement.

FIG. 3.13 – *Le vocabulaire du domaine dans les méthodes (suite)*

Technique	Avantages	Limites
Messages	Notion très utile pour l'implantation.	Expression du comportement d'un objet limitée à la notion d'appel de méthodes.
Evénements	Notion de base pour modéliser le comportement, utilisée par toutes les méthodes.	Notion proche de l'implantation et éloignée du monde réel.
Scénarios	Vision simple proche du monde réel. Exprime le comportement global de l'application.	Modélisation restreinte : pas de condition sur l'occurrence des événements ni d'actions à effectuer en réaction à ces événements.
Diagrammes de transitions d'états	Bonne vision du comportement local des objets de l'application.	Absence de comportement global de l'application.
Modèles de traitement	Bonne vision du comportement global de l'application.	Faible vision du comportement local des objets de l'application.
Modèles événementiels	Représentation originale du comportement de l'application. Bonne vision du comportement global de l'application.	Indépendant de l'aspect structurel de l'application. Faible vision des comportements locaux des objets de l'application. Un type privilégié de technologie visé pour l'implantation : programmation événementielle.
Règles	Représentation des besoins de l'application proche du monde réel et de la perception d'un utilisateur.	Notion récente, encore très peu intégrée dans les méthodes.
Langages formels	Apport de précision. Formalisation des concepts. Démonstration de faisabilité.	Difficulté de compréhension des langages pour un utilisateur non spécialiste.

TAB. 3.1 – *Avantages et limites des techniques pour la représentation du comportement*

destinées à exprimer, analyser, concevoir et enfin implanter le comportement des applications. Les avantages et inconvénients de ces techniques sont représentées dans la table 3.1.

Parfois, des aspects formels complétent les techniques utilisées pour modéliser le comportement des applications. Ces langages sont difficiles à comprendre et à appréhender (c'est pourquoi ils sont souvent associés à des notations semi-formelles [FJ97]), mais permettent néanmoins de [FGF97] :

– formaliser les concepts afin d'apporter plus de crédibilité, de cohérence et de

validité par rapport au modèle utilisé ;

- compléter les concepts du modèle pour exprimer des besoins difficiles à représenter dans le modèle graphique ;
- démontrer la faisabilité de l'implantation des concepts dans un système cible afin de valider l'application.

Ainsi, IFO2 utilise un langage formel afin de formaliser les concepts introduits intuitivement dans le modèle en les précisant relativement à la notion de **temps** et d'expliciter les conditions d'évaluation des fonctions définies dans les fragments. De même, la méthodologie IDEA utilise le langage Chimera [CM93] dans sa phase de conception afin d'obtenir une spécification précise, formelle et exécutable.

La plupart des techniques proposées sont basées sur le concept d'**événement**, même si l'importance dont bénéficie ce dernier est plus ou moins grande selon les méthodes. Les avantages de la notion d'événement pour la modélisation du comportement d'une application sont reconnus car le comportement d'une application est guidé par les réactions de celle-ci à des événements émanant de l'application ou de son environnement. Cependant, certaines limites que nous présentons ci-dessous subsistent globalement dans toutes les approches [Fro95b]. Naturellement, nous aurions pu évaluer plus précisément d'autres techniques (programmation par contraintes, logique temporelle, etc.), mais notre choix a été guidé par le contexte bases de données que nous avons donné à notre travail.

- **Difficulté d'uniformisation** des techniques utilisées pour exprimer le comportement. Nous l'avons vu, les techniques proposées pour exprimer le comportement sont nombreuses bien que majoritairement basées sur le même concept d'événement. Pour bien comprendre les besoins d'une application, nous pensons qu'il est nécessaire de considérer le comportement d'une application à la fois d'un point de vue global et de plusieurs points de vue locaux. Or, les différentes techniques utilisées dans les méthodes permettent l'expression soit d'un comportement local à un objet, soit du comportement global de l'application. Les diagrammes d'états de OMT ou de OOD par exemple sont destinés à exprimer un comportement local mono-objet. Quant au comportement global, les scénarios introduits pour exprimer une exécution particulière du fonctionnement du système sont trop généraux et bien trop peu détaillés pour réellement représenter le comportement global de l'application. D'autres méthodes sont orientées vers l'expression du comportement global de l'application (par exemple Merise),

mais les comportements locaux des objets n'apparaissent pas. Pour résoudre cet aspect, deux solutions sont envisageables : soit des techniques complémentaires provenant de modèles de méthodes différentes sont combinées afin d'exprimer à la fois le comportement local des objets et le comportement global de l'application ; soit une solution générique permet d'exprimer de la même manière les comportement locaux aux objets et le comportement global de l'application. C'est cette approche que nous adoptons pour nos propositions (cf. section 5).

- **Difficulté d'utilisation**. Certaines techniques telles que les langages formels sont intéressantes, mais difficiles à utiliser alors que nous souhaitons proposer des techniques accessibles aussi bien à un spécialiste de la conception d'applications qu'à un utilisateur demandeur d'une application et devant rédiger ou évaluer un cahier des charges.

- **Difficulté d'interprétation**. Certains concepts sont intéressants, mais manquent de maturité et sont difficiles à interpréter. Les règles de fonctionnement de OOAD ou les règles de gestion de IDEA sont ainsi très prometteuses pour représenter le comportement des entités de l'application ou de l'application elle-même, mais le support pour utiliser ces concepts reste minime.

- **Difficulté d'indépendance** par rapport à l'implantation. Afin de permettre à un utilisateur demandeur d'une application d'utiliser sans difficultés les techniques proposées, et pour augmenter la généricité de ces techniques, il est nécessaire d'éloigner les techniques proposées de la phase d'implantation. Or, des méthodes comme IFO2 ou IDEA explicitement destinées à la conception de bases de données respectivement actives et déductives, ne se détachent pas des concepts proposés dans les systèmes. Les notions de règles actives et de règles déductives introduites dans la phase de conception d'IDEA aussi bien que les fonctions de précédence ou les fonctions différées des schémas événementiels d'IFO2 sont des concepts de l'implantation remontés au niveau des phases d'analyse et de conception. Il apparaît alors que l'approche est prise dans le mauvais sens : elle n'est plus descendante en transformant les besoins d'une application dans un système cible, mais ascendante en utilisant les technologies introduites dans les systèmes cibles dès l'analyse et la conception d'une application. Ainsi, le fait d'introduire dans IDEA les règles de gestion comme celles qui correspondent à des règles actives Chimera illustre très bien ce point de vue.Cela ne nous satisfait pas car nous souhaitons conserver une approche d'une part descendante en traduisant les besoins d'une application dans un système cible particulier, d'autre part gé-

nérique afin de permettre une éventuelle implantation dans plusieurs systèmes technologiques susceptibles de supporter les besoins de l'application.

3.3 Conclusion : vers une nouvelle approche...

L'ingénierie des systèmes d'information est un domaine complexe partant de l'analyse des besoins d'une application pour aboutir à une solution logicielle fiable dans un système cible choisi. Elle fait partie du domaine plus large et plus complexe qu'est l'ingénierie des logiciels. Le problème qui nous intéresse de la représentation et de la prise en compte des situations comportementales dans une application n'est donc pas indépendant des autres composants de l'application et pour tenter d'apporter des solutions à ce problème, nous avons fait un choix pratique de techniques couramment enseignées et pour certaines utilisées professionnellement. Notons cependant que ce domaine difficile a été abordé d'une manière significative par quelques équipes francophones d'un point de vue système d'information [AJLGLP94] [RFB88] [BP93] [Pig96] ou d'un point de vue génie logiciel en terme de cycle de vie [Béz95] et de réutilisation [Ner92] [Mor96].

Cette dernière dimension qu'est la réutilisation prend aujourd'hui de plus en plus d'importance. En effet, elle est devenue avec les notions d'objets et de classes l'un des critères déterminant pour le succès d'un langage de programmation ou d'un système. Les bibliothèques d'applicatifs de SmallTalk [GR83] par exemple permettent de réutiliser des techniques acquises et favorisent ainsi une diminution du coût du développement d'une application. Si elle est actuellement assez bien traitée au niveau de l'implantation, la réutilisation reste très limitée, voire inexistante aux niveaux de l'analyse et de la conception. L'approche que nous préconisons propose des éléments d'une part pour pallier les limites des méthodes traditionnelles concernant leur capacité à représenter les situations comportementales, d'autre part pour permettre la réutilisation des connaissances acquises par les développeurs d'applications dès le niveau de l'analyse des besoins d'un système d'information.

Chapitre 4

Approches de conception à base de patrons

DANS L'APPROCHE CLASSIQUE présentée dans la section 3.1 et utilisée dans les méthodes à objets comme OOA, OMT ou OOD, la partie située en amont de l'activité de modélisation des systèmes d'information concerne l'ingénierie des besoins centrée sur l'acquisition et la représentation des connaissances intégrées dans un schéma conceptuel. La partie située en aval correspond à l'ingénierie du système et transforme le schéma conceptuel dans un système cible opérationnel (cf. figure 3.1). Une telle approche de l'ingénierie des systèmes logiciels s'organise par la succession de modèles afin de favoriser un continuum de la définition des besoins des clients jusqu'au système développé et exploité. Ce continuum est grandement facilité par le concept d'objet.

L'un des objectifs des technologies par objets est cependant de faciliter et d'améliorer les tâches de conception et de codage grâce à la réutilisation de composants. L'approche à base de patrons sur laquelle se sont concentrées récemment plusieurs équipes (R. Johnson [GHJV94], P. Coad [CM96], C. Rolland [Rol93], etc.) et que nous présentons dans la section 4.2 consiste à identifier et à définir des abstractions générales appelées **patrons** applicables à différentes situations de même type. Le développement d'une nouvelle application combine alors d'une part la réutilisation directe de composants souvent très élémentaires, d'autre part l'adaptation ou la spécialisation de composants plus génériques et plus complexes. Avant d'aborder cette approche, rappelons quelques notions générales sur la réutilisation.

4.1 La réutilisation

De nos jours, une véritable industrialisation se caractérise par la conception et le développement d'un ensemble de produits à partir d'éléments réutilisables. Les industries qui travaillent de cette façon sont alors capables de développer des produits sur mesure, rapidement et à des prix compétitifs, en les assemblant à partir d'éléments standards pris à différentes étapes du processus de fabrication. Ainsi, quel que soit le métier, être *industriel* dans la définition d'un nouveau produit, c'est **réutiliser** des composants. Ces procédés sont maintenant éprouvés dans plusieurs domaines industriels, par exemple la construction automobile ou l'électronique.

Dans le secteur de l'informatique, les informaticiens bancaires ont été les premiers à découvrir et à appliquer ces principes de réutilisation, compte-tenu de la centralisation et de l'homogénéité des systèmes d'information dans les banques. D'une manière générale, en informatique de gestion, les années 70-80 ont été marquées par une réutilisation globale sous la forme de progiciels paramétrés adaptables à des systèmes d'information spécifiques dans divers domaines : comptabilité, gestion du personnel, etc. Aujourd'hui, cette démarche se généralise dans tous les secteurs d'activité de la production de logiciels : de grandes organisations aussi bien que de petits centres de développement ont montré qu'un taux de réutilisation de plus de 80% pouvait être atteint. La NASA affirme par exemple que la réutilisation lui a permis de diviser son taux d'erreur par quatre et de multiplier sa productivité par deux [Mor96].

La réutilisation lors de la production d'un logiciel vise trois objectifs principaux essentiels pour répondre à la compétitivité et à la concurrence sur les marchés économiques : diminuer les coûts de développement et de maintenance, réduire les délais et améliorer la qualité du logiciel. Ces trois objectifs ne peuvent cependant être obtenus qu'au prix de l'application systématique d'une démarche méthodologique tout au long du cycle de production, c'est-à-dire depuis l'analyse des besoins jusqu'à la maintenance. Aujourd'hui, bien que la pénétration d'approches industrielles de méthodes de réutilisation soit encore faible, des travaux de plus en plus nombreux concentrent leurs efforts sur la recherche de méthodes adaptées au développement de logiciels par réutilisation. En particulier, le consortium européen REBOOT a étudié comment introduire et organiser la réutilisation dans l'industrie du logiciel [Kar95] [Mor96]. Il a ainsi mis au point une démarche méthodologique pour créer, gérer et utiliser des composants réutilisables dans le développement de nouveaux produits ou de nouveaux systèmes logiciels. Cette démarche est destinée à aider les producteurs de logiciels à mettre en œuvre une vraie politique de réutilisation.

4.1.1 Des couches pour la réutilisation

L'informatique et plus particulièrement le logiciel par nature reproductible quasiment instantanément à coût très faible, s'avèrent adéquats pour la réutilisation systématique, laquelle représente un potentiel scientifique et économique primordial pour atteindre les trois objectifs précédents.

Pour mettre de l'ordre dans l'ensemble des composants qui peuvent être mis à la disposition d'un développeur d'applications, L. Mondemé [Mon96] distingue sept couches pour la réutilisation de composants dans le domaine de l'informatique de gestion (cf. figure 4.1). Cette représentation du modèle des sept couches de la réutilisation a pour objectifs principaux de :

- prouver que la gamme des composants potentiellement réutilisables est large, allant de composants très techniques (la gestion de listes d'objets en mémoire) jusqu'à des composants très fonctionnels (la gestion de contrats) ;

- montrer qu'une bibliothèque de composants doit posséder une structure interne et ne pas simplement être une collection de composants les uns à côté des autres.

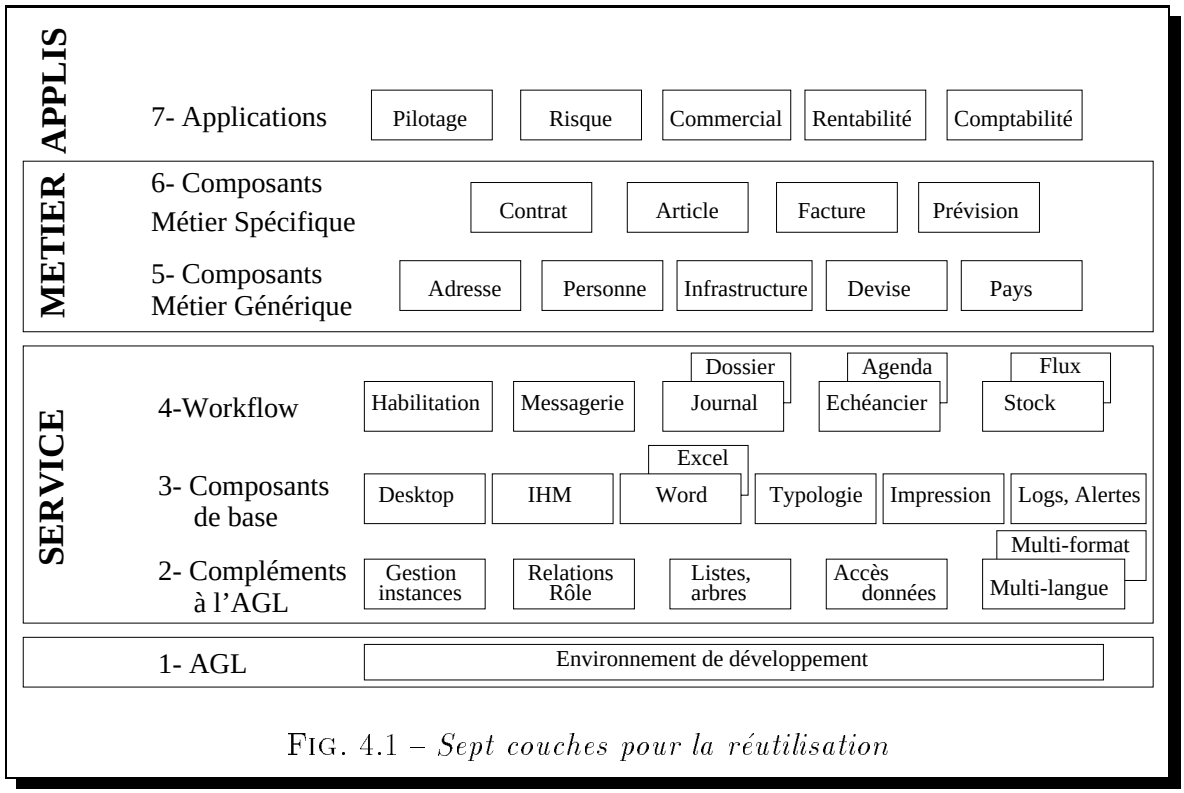


FIG. 4.1 – *Sept couches pour la réutilisation*

Les sept couches présentées dans la figure 4.1 sont détaillées ci-dessous.

- **Niveau 1 - environnement de développement** : les produits de développement (ateliers de génie logiciel, dictionnaires, langages, ...) offerts par le marché. Les services rendus peuvent être très variables en fonction de l'AGL choisi : manipulation de chaînes de caractères, gestion de listes d'instances, etc.

- **Niveau 2 - architecture des services - les compléments à l'environnement de développement** : par exemple, la gestion du multi-linguisme si l'AGL ne l'offre pas en standard. Cette couche sera d'autant plus légère que l'AGL utilisé est puissant.

- **Niveau 3 - architecture des services - les services de base** : une multitude de services utiles au développement d'applications de gestion, comme la gestion d'une interface homme-machine homogène, l'intégration de logiciels bureautiques, la définition de types communs comme des montants, des services d'impression, etc.

- **Niveau 4 - architecture des services - les composants nécessaires au workflow** : les services utiles dès lors qu'il est nécessaire d'organiser la répartition de tâches entre acteurs : contrôle d'habilitation, journalisation, gestion d'exceptions, messagerie, échéancier, etc.

- **Niveau 5 - architecture du métier - les composants référence (ou métier commun)** : les services de gestion des personnes, des correspondants, des adresses, de la structure de l'entreprise, des devises, des pays, des valeurs mobilières, etc. qui constituent généralement les référentiels de l'entreprise.

- **Niveau 6 - architecture du métier - les composants métier spécifique** : les concepts propres au métier à informatiser. Chaque métier possède ses propres concepts et donc ses propres composants "métier spécifique". Par exemple dans l'assurance, les composants garantie, risque, couverture, etc.

- **Niveau 7 - les applications** : celles développées sur la base des composants réutilisables, telle que Pilotage, Risque client, Commercial, Facturation, etc.

Les couches 1 à 6 sont les couches de réutilisation. Si elles sont bien construites, on peut atteindre des taux de 70% de réutilisation. Il ne reste alors plus qu'un tiers du logiciel à développer pour une application à priori très spécifique.

Certains composants des niveaux hauts (4 à 6) peuvent être utilisés à la fois comme des applications opérationnelles (les applications de gestion des référentiels Personne ou Devise par exemple) et comme des composants intégrables dans des applications fonctionnelles de plus haut niveau : la transaction Virement Bancaire utilise les services de niveau 5/6 comme ChercherClient ou Mettre à jour Compte. Les composants du niveau 6 sont spécifiques à un métier (celui de la banque par exemple pour les opérations de caisse, etc.), ce qui n'est pas le cas des couches 2 à 5 où une même bibliothèque de composants peut être réutilisée dans des métiers différents.

Cette représentation en couches ne doit pas être vue comme un modèle étanche où une application (couche 7) ne pourrait réutiliser que des composants du métier spécifique (couche 6). Bien au contraire, l'ensemble des composants présents sur cette figure peut être réutilisé au sein d'une application pour atteindre l'objectif de 70% de réutilisation. Par exemple, l'application Crédit Immobilier peut réutiliser les composants Personne pour le ou les titulaires, gestion des exceptions, types communs (date, montant), IHM (modèles de présentation et d'enchaînements standard), système d'habilitation (pour contrôler la signature du prêt), intégration bureautique (pour la saisie de données textuelles sur le client), etc.

Par rapport à ce modèle, l'approche que nous proposons se situe à trois niveaux.

- Le but général de notre approche étant de concevoir le comportement d'applications réactives, nous intervenons au niveau 7 - Applications.

- Puisque le point de départ de notre approche est la classification de situations comportementales communes à plusieurs domaines d'applications, nous intervenons au niveau 5 - Métier Générique.

- Enfin, nous intervenons au niveau 1 - Environnement de Développement en cherchant à concevoir des applications dans un système cible particulier, en l'occurrence dans ce document le système de gestion de bases de données actifs NAOS (cf. chapitre 6).

L'étape ultérieure serait d'aller plus loin sur l'étude des applications du domaine médical pour enrichir le niveau 6 - Composants Métier Spécifique comme d'autres proposent de le faire dans d'autres domaines [RTBG97].

4.1.2 Réutilisation et abstraction

L'expressivité des langages de programmation de haut niveau avantage l'écriture de programmes où sont exprimés des concepts abstraits. De tels langages prennent en compte les problèmes de structuration des programmes par des constructions linguistiques telles les modules, les objets, les *packages* d'ADA [Dep83] ou encore les *clusters* de CLU [Lis81]. De plus, ces constructions modulaires supportent des formes d'abstraction basées sur le masquage d'information.

Le terme **abstraction** permet l'expression des algorithmes indépendamment de toute représentation des données en machine. L'abstraction des programmes est l'un des éléments essentiels pour d'une part contrôler la complexité croissante des systèmes, d'autre part écrire des composants logiciels réutilisables. Il existe quatre principales formes d'abstraction : la modularité, le polymorphisme, la fonctionnalité et l'héritage.

Modularité - Elle consiste en la définition de **modules**, comme dans Simula-67 [DN66], CLU [Lis81] ou encore Modula-2 [Wir83]. Les protocoles d'accès et d'utilisation des modules sont représentés dans des interfaces formées par la déclaration d'un ensemble d'opérations typées primitives au module. Une interface peut être spécifiée et compilée indépendamment de son implantation. De plus, l'**encapsulation** d'une structure de données dans un type défini par une interface la rend réutilisable dans d'autres contextes, les modifications de la représentation interne d'un module n'ayant aucune incidence sur les autres composants du programme.

Polymorphisme - Il permet de traiter des valeurs dont le type n'est pas unique. Une fonction polymorphe peut être appliquée avec des arguments de différents types (par exemple, la fonction length qui calcule la longueur d'une liste pourra être appliquée sur n'importe quel type de liste). La **surcharge** permet en outre de lier différentes valeurs à un même symbole. Un programme surchargé ou polymorphe peut être exécuté dans différents contextes, c'est-à-dire en choisissant différentes représentations des structures de données.

Fonctionnalité - Elle permet de traiter les fonctions comme les autres valeurs d'un programme. Les fonctions étant des objets du langage, elles peuvent être passées en arguments à d'autres fonctions appelées fonctionnelles ou être retournées comme résultat d'une fonctionnelle. De telles constructions avantagent l'expression factorisée

des algorithmes qui majore l'abstraction des traitements par rapport aux données sur lesquelles ils s'appliquent.

Héritage - Construction caractéristique des langages de programmation orientés-objets, l'extension par héritage est typique d'une réutilisation logicielle puisqu'aucune duplication explicite de code n'est réalisée par l'utilisateur lors d'un héritage entre deux classes.

Le besoin de réutilisation est l'une des bases du succès de l'approche objet ces dernières années. Pour la première fois peut-être, ces langages ont montré de manière constructive avec la création de bibliothèques de classes outils et à travers leurs principes d'encapsulation et d'héritage, qu'on pouvait répondre à une demande de réutilisabilité justifiée tant sur le plan technique que sur le plan économique. Les langages orientés-objets SmallTalk [GR83], Eiffel [MNM87] ou C++ [BK93] permettent par exemple de répondre à ce besoin en mettant à la disposition du développeur d'applications des **bibliothèques** de composants réutilisables.

4.1.3 La réutilisation dans le développement orienté objet d'une application

La réutilisation est aujourd'hui un terme à la mode. Le plus souvent cependant, elle n'apparaît qu'au niveau du codage par la réutilisation de quelques classes ou de quelques fonctions de la bibliothèque de composants fournie avec l'environnement de développement. Ces bibliothèques de composants logiciels réutilisables représentent effectivement une réelle progression vers la réutilisation maximale dans la construction d'un programme, mais le taux de réutilisation reste limité à 15-20%. La figure 4.2 montre les niveaux de réutilisation accomplis dans le processus de développement par objet d'une application.

De nombreux progrès restent à faire pour réellement intégrer la réutilisation dans le processus de développement d'une application, en particulier lors des phases d'analyse et de conception. Au plus bas niveau, de nombreux mécanismes comme l'héritage permettent de réutiliser des composants existants lors de l'implantation d'une application. De la même façon, les bibliothèques logicielles (par exemple, dans SmallTalk)

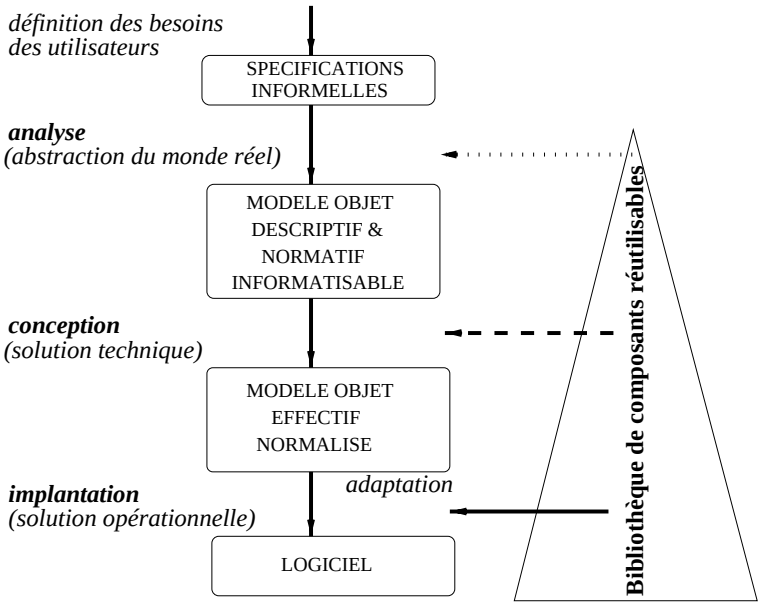


FIG. 4.2 – *La réutilisation de composants dans le cycle de vie [Ner92]*

sont destinées à mécaniser la réutilisation de composants logiciels. L'intégration de tels outils dans les environnements de programmation avantage une approche *bottom-up* du processus de développement des systèmes par réutilisation systématique de composants logiciels prédéfinis et validés. Cependant, à un plus haut niveau, les mécanismes permettant de réutiliser des connaissances acquises lors de l'analyse ou de la conception restent encore pratiquement inexistants. Dans ce cadre, l'approche à base de patrons a pour but de participer à la réutilisation de connaissances acquises par des développeurs d'applications dans l'analyse et la conception d'applications.

4.2 L'approche à base de patrons

D'après le Petit Robert [Rob93], un patron est "un modèle sur lequel travaillent les artisans pour fabriquer certains objets". En réalité, le terme *patron* est utilisé dans plusieurs domaines : en couture, un patron est un modèle de papier ou de toile préparé sur un mannequin ou selon les mesures d'une personne et utilisé pour créer des mêmes vêtements ; en décoration, un patron est une figure apparaissant dans une fourniture ou un accessoire ; en manufacture, un patron est une forme ou un style d'une pièce à fabriquer ; en aviation, un patron est un ensemble d'approches, de tours et d'altitudes que doit respecter un avion en approchant une ville ; en radiodiffusion,

un patron est un diagramme standard pour tester des circuits de télévision ; dans un jeu d'échecs, un patron est un ensemble de mouvements qui peuvent être appliqués pour une stratégie donnée ; en linguistique, un patron est la manière avec laquelle de petites unités de langage sont groupées dans des unités plus larges ; enfin, de nombreux autres domaines tels que l'archéologie utilisent la notion de patrons. En général, les patrons sont composés de petites unités standardisées et regroupées et sont utilisés répétitivement comme des blocs de construction et de conception.

De façon abstraite, un patron est l'équivalent français proposé pour *pattern*. D'après le Petit Robert, un *pattern* est synonyme de modèle, schéma, structure et type et en sciences humaines, c'est un modèle simplifié d'une structure.

4.2.1 Historique

L'origine des patrons de conception remonte à des études réalisées par C. Alexander à la fin des années 1970 dans le domaine de l'architecture des bâtiments [AIS+77] [Ale79]. Cet architecte a développé l'idée d'un *langage de patrons* permettant à des personnes non-architectes de construire leurs propres maisons. Ce langage est formé d'un ensemble de patrons dont chacun décrit comment résoudre un problème particulier de la construction d'une maison. Les problèmes abordés par les différents patrons peuvent être assez larges ou se spécialiser petit à petit jusqu'à devenir très spécialisés (comment disposer les pièces dans la maison, quel matériel utiliser pour construire les murs, comment décorer les pièces ou encore comment installer l'électricité). Le langage de patrons d'Alexander ne nécessite pas de connaissances spécifiques et se concentre sur des problèmes de conception communs ou moins communs lors de la construction d'une maison.

C'est une dizaine d'années après les travaux de C. Alexander que la notion de patron a été introduite dans le domaine de l'informatique. K. Beck et W. Cunningham ont ainsi présenté en 1987 lors de la conférence OOPSLA'87 un article intitulé *Using Patterns Languages for Object-Oriented Programs* [BC87]. Depuis, les recherches sur les patrons dans le domaine de l'informatique sont nombreuses et plusieurs livres spécialisés ont été publiés [Pre94] [GHJV94] [BMR+96] [CM96] [Cop96] [Fow97]. Les conférences OOPSLA et ECOOP comportent régulièrement des articles traitant de la notion de patrons. Enfin, des conférences annuelles telles que PLoP (Pattern Languages of Programming) [CS95] [VCK96] et EuroPLoP (European Conference on Pattern Languages of Programming) ainsi que des workshops tels que UP (Using Patterns) sont dédiés aux patrons.

4.2.2 Définition et exemples de patrons

D'une façon générale, J. Rumbaugh définit un patron comme une "tentative pour la représentation de l'expérience personnelle des développeurs de manière uniforme". D'après P. Coad [Coa92], un patron est "une forme entièrement réalisée, originale ou un modèle accepté ou proposé pour une imitation ; quelque chose qui est vu comme un exemple normatif pouvant être copié, archétypé ou utilisé comme exemple". Une définition plus générale a été introduite récemment dans un survey de l'ACM [ACM96] : "un patron a pour but de **décrire avec succès des solutions récurrentes à des problèmes logiciels communs dans un certain contexte, et d'aider les gens à réutiliser des pratiques vraies et résolues**". La définition d'un patron a été adaptée au domaine de la conception orientée objet d'applications. Peter Coad [Coa92] définit ainsi un patron orienté objet comme **une abstraction d'un doublet, triplet ou autre petit groupe de classes qui peut être utile encore et encore dans tout développement orienté objet**.

Pour concrétiser cette notion de patron, nous en donnons ci-dessous deux exemples.

4.2.2.1 Patron des rôles

Dans de nombreuses situations humaines ou matérielles, les agents ou les machines sont capables d'assumer plusieurs rôles ou tâches dans certains cas dans un ordre chronologique, dans d'autres cas simultanément. Par exemple, dans le domaine du travail, une personne peut être simultanément ou successivement programmeur, analyste, chef de projet, etc. De même, dans le domaine de la manufacture, une machine peut être successivement (si elle est utilisée par une seule personne) ou simultanément (si elle est utilisée par deux personnes en même temps) perceuse, ponceuse, etc.

Dans certains cas, la date de début et la date de fin de chaque rôle sont communes : le chevauchement des dates de début et des dates de fin de chaque rôle est alors caractéristique de rôles simultanés ; dans d'autres cas, les rôles d'un objet ou d'une personne sont successifs. Il est donc nécessaire de représenter tous les cas possibles de combinaison de rôles. Ce problème est fréquemment modélisé avec de nombreuses relations de généralisation-spécialisation entre la classe générale des agents (ou des machines) et les classes spécialisées des rôles (ou des tâches) spécifiques : spécialiser la classe Employé en sous-classes Programmeur, Chef de Projet, etc. et la classe Machine en sous-classes Perceuse, Ponceuse, etc. Ce type de modélisation ne permet cependant pas de répondre facilement à de nombreuses questions combinant les aspects rôles (ou

tâches) et les aspects temporels (dates de début et de fin de chaque rôle).

Pour représenter ce besoin de façon plus concise et plus flexible que l'héritage multiple, P. Coad [Coa92] [CM96] propose le patron des rôles qui combine agrégation et spécialisation afin de modéliser le fait qu'une même entité (personne, objet, machine, ...) peut avoir un grand nombre de rôles spécifiques qui peuvent évoluer et être combinés (cf. figure 4.3).

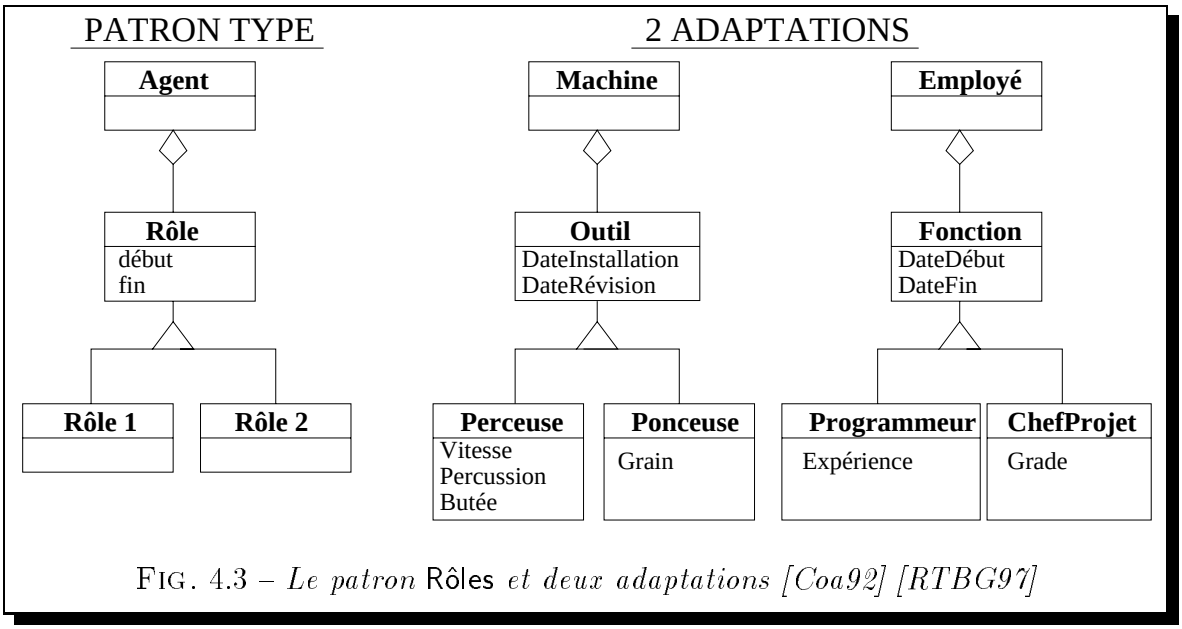


FIG. 4.3 – *Le patron* Rôles *et deux adaptations [Coa92] [RTBG97]*

Il ne faut pas confondre la notion de rôle avec celle d'héritage : un rôle peut être instancié plusieurs fois pour le même objet à des dates différentes. Ce patron est donc utilisable chaque fois qu'une telle combinaison d'agrégations et de spécialisations multiples de rôles doit être mise en place. L'utilisation du patron des rôles consiste à :

- identifier que cette notion de rôles multiples de chaque employé ou d'outils multiples sur chaque machine correspond aux principes du patron des rôles,

- utiliser cette architecture générale de schéma à trois niveaux de classes organisées successivement par une relation d'agrégation et une relation de généralisation,

- personnaliser les classes et les sous-classes ainsi que les multiplicités des liaisons selon l'application (gestion des personnes, gestion des machines, etc.).

4.2.2.2 Méta-schéma Ressources

Parallèlement aux travaux de P. Coad qui se sont limités dans un premier temps aux aspects strictement statiques, C. Rolland [Rol93] a mis l'accent sur les aspects dynamiques en mettant en évidence la notion de **similarité des situations** dans de nombreux domaines d'application. A l'aide d'un formalisme objet-événement-opération, elle montre en particulier qu'un modèle de gestion de réservation de chambres d'hôtels est similaire à un modèle de gestion de réservation de livres dans une bibliothèque : il s'agit de deux applications particulières d'une gestion de ressources ou chaque ressource est soit disponible, soit momentanément affectée. Ainsi, des ressources diverses (un livre dans une bibliothèque, une chambre dans un hôtel, un spool d'impression dans un système de gestion d'imprimantes, etc.) se comportent dynamiquement de la même manière : elles peuvent être créées, supprimées, réservées, libérées, etc. C. Rolland a donc défini un méta-schéma dynamique donnant le comportement type d'une ressource (cf. figure 4.4).

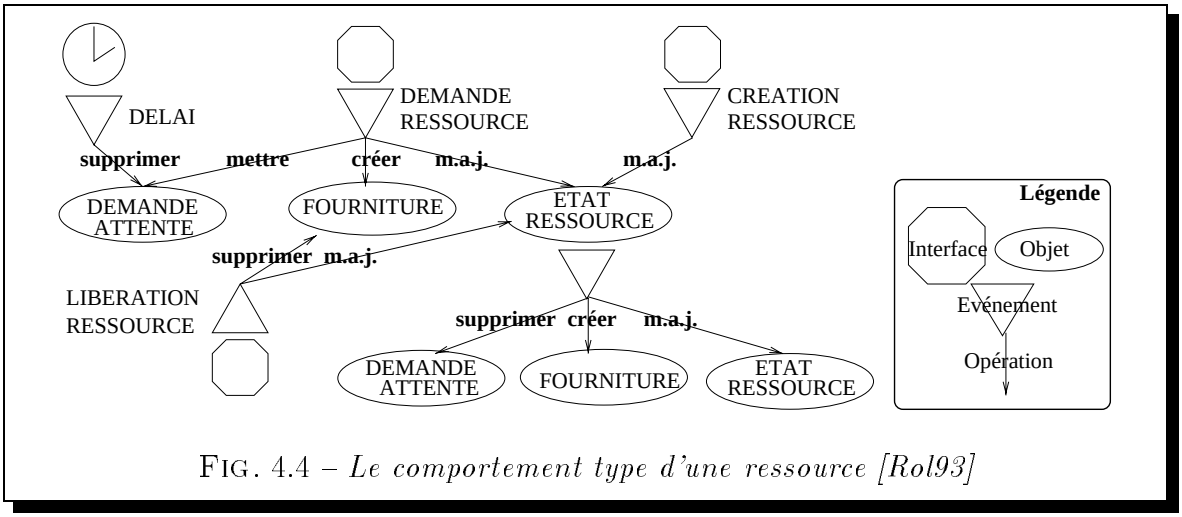


FIG. 4.4 – *Le comportement type d'une ressource [Rol93]*

Un article récent [RTBG97] complète la représentation de cet aspect dynamique de gestion des ressources par un aspect statique mis en évidence par un patron du même ordre que ceux de P. Coad. Ainsi, la figure 4.5 illustre deux exemples du patron *Ressources* où les ressources sont des livres (respectivement des outils), les producteurs des bibliothécaires (respectivement des magasins outils) et les consommateurs des lecteurs (respectivement des machines).

Enfin, ce patron succint peut être complété par exemple par des opérations, des diagrammes d'interactions ou encore des diagrammes d'états attachés aux classes, en

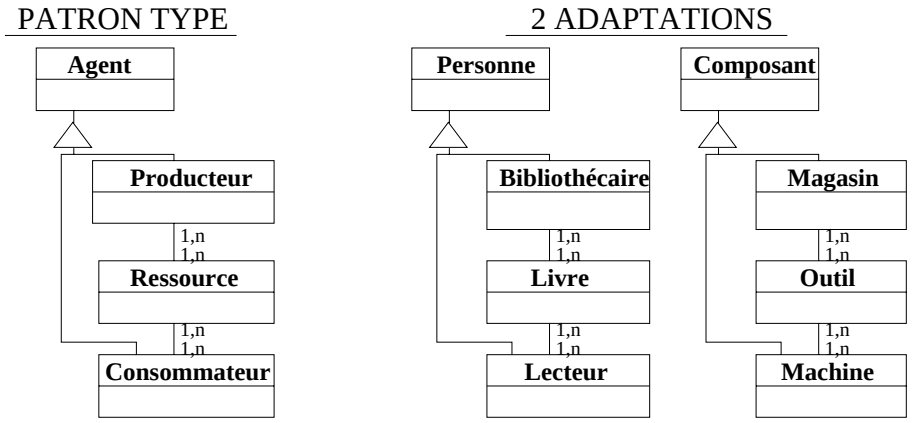


FIG. 4.5 – *Le patron* Ressources *et deux adaptations [RTBG97]*

particulier pour exprimer les informations suivantes :

- tout producteur (bibliothécaire, hôtelier, magasin, etc.) peut ajouter ou allouer une ressource (livre, chambre, outil, etc.),

- tout consommateur (lecteur, client, machine, etc.) peut demander une ressource ou annuler une demande de ressource,

- toute ressource est dans l'état disponible ou non disponible. Elle peut également être à disposition des consommateurs, en examen ou en recyclage (cf. figure 4.6).

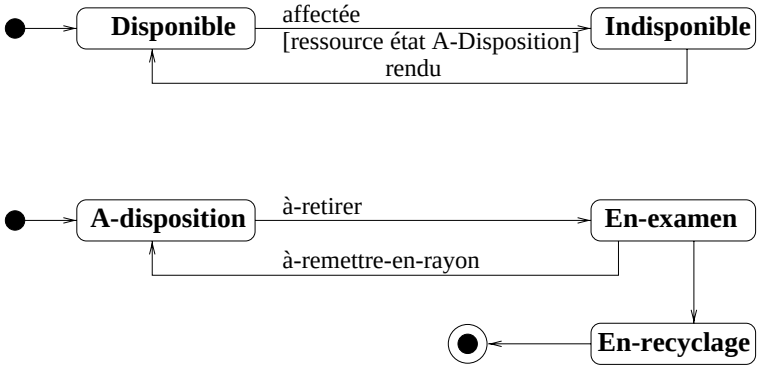


FIG. 4.6 – *Diagrammes d'états concurrents des objets ressources [RTBG97]*

4.2.3 Formalismes de représentation d'un patron

D'après la définition donnée par l'ACM [ACM96], un patron est caractérisé par trois composants principaux : le **problème**, le **contexte** et la **solution**. De nombreux formalismes de représentation d'un patron intégrent ces trois aspects et sont globalement équivalents les uns par rapport aux autres. Nous nous limitons à présenter ici trois formalismes. Le premier a été introduit par C. Alexander pour décrire les patrons nécessaires à la construction de bâtiments dans le domaine architectural. Les deux suivants sont ceux que nous considérons comme les plus représentatifs des formalismes utilisés pour représenter des patrons dans le domaine de l'analyse, la conception et l'implantation orientées objets. En particulier, le formalisme utilisé par le *Gang of Four* s'impose actuellement. En annexe A, nous présentons le formalisme de M. Fowler [Fow97] sur un exemple plus particulièrement adapté aux situations temporelles.

4.2.3.1 Formalisme d'Alexander

Selon C. Alexander, un patron comprend cinq composants principaux :

- le **nom** : un nom ou une phrase courte, familière et descriptive, par exemple *Alcôve*, *Entrée Principale*, *Fenêtres Intérieures*, etc. ;
- un ou des **exemples** : un/des dessins, diagrammes et/ou descriptions illustrant l'application ;
- le **contexte** : les situations dans lesquelles le patron s'applique ;
- le **problème** : une description des forces essentielles du patron et des contraintes sous lesquelles il s'applique, ainsi que les interactions entre ces deux éléments ;
- la **solution** : la manière de résoudre le problème, composée de relations statiques et de règles dynamiques décrivant comment construire les artéfacts en accord avec le patron. Souvent, des variantes sont proposées ainsi que des manières d'ajuster la solution en fonction des circonstances. Parfois, la solution nécessite l'utilisation d'autres patrons.

Ces cinq composants sont regroupés sous la forme textuelle suivante :

IF you find yourself in CONTEXT
for example EXAMPLES

with PROBLEM
entailing FORCES
THEN for some REASONS
apply DESIGN FORM AND/OR RULE
to construct SOLUTION
leading to NEW CONTEXT and OTHER PATTERNS

Globalement, ce formalisme de représentation est narratif et peu structuré et invite tout lecteur qui recherche un élément particulier à une lecture complète. En comparaison, les patrons d'analyse, de conception et d'implantation orientées objets sont souvent très complexes et en incluant le code nécessaire à la réalisation de la solution proposée dans un langage cible, leur description atteint souvent une dizaine de pages. La représentation de tels patrons nécessite donc d'être très claire. C'est pourquoi les formalismes de représentation des patrons utilisés de nos jours dans ce domaine sont structurés, bien que leur contenu soit globalement équivalent à celui proposé par C. Alexander. Nous présentons ci-dessous deux formalismes structurés utilisés régulièrement pour représenter des patrons en informatique. Précisons que la reconnaissance d'un patron est soumise à discussion dans des congrès scientifiques et nécessite l'expression du patron dans l'un ou l'autre de ces formalismes.

4.2.3.2 Formalisme de P. Coad

P. Coad [CM96] offre 148 stratégies et 31 patrons destinés à guider un concepteur dans la construction de modèles objets effectifs et complets. En particulier, les quatre dernières stratégies se concentrent sur l'identification de nouveaux patrons : comment découvrir de nouveaux patrons, comment les nommer, comment les raffiner et comment les décrire. La description d'un patron nécessite l'inclusion des rubriques données dans la table 4.1.

4.2.3.3 Formalisme du *Gang of Four*

E. Gamma, R. Helm, R. Johnson et J. Vlissides, regroupés sous le patronyme du *Gang of Four*, proposent 23 patrons [GHJV94] exprimés dans une structure composée de 14 rubriques données dans le tableau 4.2. Par souci de simplicité, nous appelons ce formalisme commun "formalisme d'E. Gamma".

Rubrique	Signification de la rubrique
Nom	Nom du patron constitué des noms des différents acteurs composant le patron.
Catégorie	Catégorie du patron précisant le type du patron selon son contexte. Elle est choisie parmi cinq types définis par l'auteur : le patron **fondamental**, les patrons de **transaction**, les patrons d'**agrégation**, les patrons de **plan** et les patrons d'**interaction**. Elle peut éventuellement être une nouvelle catégorie.
Gabarit	Représentation graphique type OMT montrant les classes composant le patron et les relations entre ces classes.
Interactions	Interactions entre les objets des classes composant le patron.
Exemples d'utilisation	Instanciations possibles des différentes classes composant le patron.
Combinaisons	Combinaisons possibles avec d'autres patrons.
Remarques	Remarques éventuelles complémentaires.

TAB. 4.1 – *Rubriques du formalisme de représentation de P. Coad*

4.2.3.4 Comparaison entre les formalismes

La figure 4.7 compare les rubriques des trois formalismes précédents et permet de dresser les conclusions que nous présentons ci-dessous.

- Le formalisme original proposé par C. Alexander recouvre très bien celui proposé de nos jours par E. Gamma et al., puisque chacune des rubriques de C. Alexander y est présente sous une forme plus détaillée (par exemple, la *solution* est détaillée en cinq rubriques dans le formalisme d'E. Gamma).

- Le formalisme de P. Coad est assez peu détaillé et il est difficile de retrouver des rubriques de ce formalisme dans chacun des deux autres formalismes, en particulier celui de C. Alexander. Notons que la partie Remarques de ce formalisme est ambiguë : nous ne l'avons rattachée à aucune autre partie des deux autres formalismes, mais elle peut aussi être considérée comme incluant chacune des rubriques des autres formalismes.

- Enfin, toutes les rubriques proposées par le formalisme d'E. Gamma sont présentes dans l'un ou l'autre des formalismes de C. Alexander et de P. Coad. La partie Classification est rattachée à la rubrique Catégorie de P. Coad car le but de ces deux catégories est le même (classer le patron dans une catégorie), mais les

Rubrique	Caractéristiques de la rubrique
Nom	Reflète l'essence même du patron.
Classification	Porte sur deux aspects : la **juridiction** et la **caractérisation**. La juridiction concerne le domaine sur lequel le patron s'applique : dans une **juridiction de classe**, le patron traite de relations entre des classes et des sous-classes ; dans une **juridiction d'objet**, le patron traite de relations entre des objets ; enfin une **juridiction composée** signifie que le patron concerne des structures récursives d'objets. La caractérisation reflète ce que fait un patron et peut être de trois types différents : **créationnel** (portant sur le processus de création des objets), **structurel** (concernant la composition des classes et des objets) ou **comportemental** (caractérisant la façon avec laquelle les classes et les objets interagissent et distribuent leurs responsabilités).
Intention	Ce que fait le patron, le problème de conception particulier auquel il s'adresse.
Aussi connu comme	D'autres noms connus du patron.
Motivation	Un scénario d'application du patron, les problèmes particuliers auxquels il s'adresse et les classes et les objets qu'il met en jeu.
Applicabilité	Les situations dans lesquelles le patron peut être utilisé.
Participants	Les classes et/ou les objets composant le patron ainsi que leurs responsabilités.
Collaborations	Comment les participants collaborent pour accomplir leurs responsabilités.
Diagramme	Une représentation graphique d'un patron en notation OMT [RBP+91] augmentée avec du pseudo-code pour expliciter les méthodes.
Conséquences	Comment le patron atteint ses objectifs, quels sont les échanges et les résultats suite à l'utilisation du patron.
Implantation	Des astuces, des conseils et des techniques utilisables pour implanter le patron.
Exemple de code	Des échantillons de code illustrant l'implantation du patron dans un langage tel que C++ ou SmallTalk.
Utilisations	Des exemples issus du monde réel dans au moins deux domaines différents.
Voir aussi	Les éventuels patrons ayant une intention proche de celle du patron décrit et leurs différences ainsi que les patrons susceptibles d'être utilisés avec celui décrit.

TAB. 4.2 – *Rubriques du formalisme de représentation d'E. Gamma*

catégories proposées par chacun des formalismes sont différentes : P. Coad propose de classer les patrons selon leur contexte (agrégation, interaction, etc.) alors que E. Gamma distingue les patrons selon leur nature (créationnel, structurel ou comportemental) et leur portée (une classe, un objet ou une combinaison de classes et d'objets).

Le formalisme d'E. Gamma inclut donc chacun des deux autres formalismes et nous l'adoptons pour présenter les patrons du langage que nous proposons.

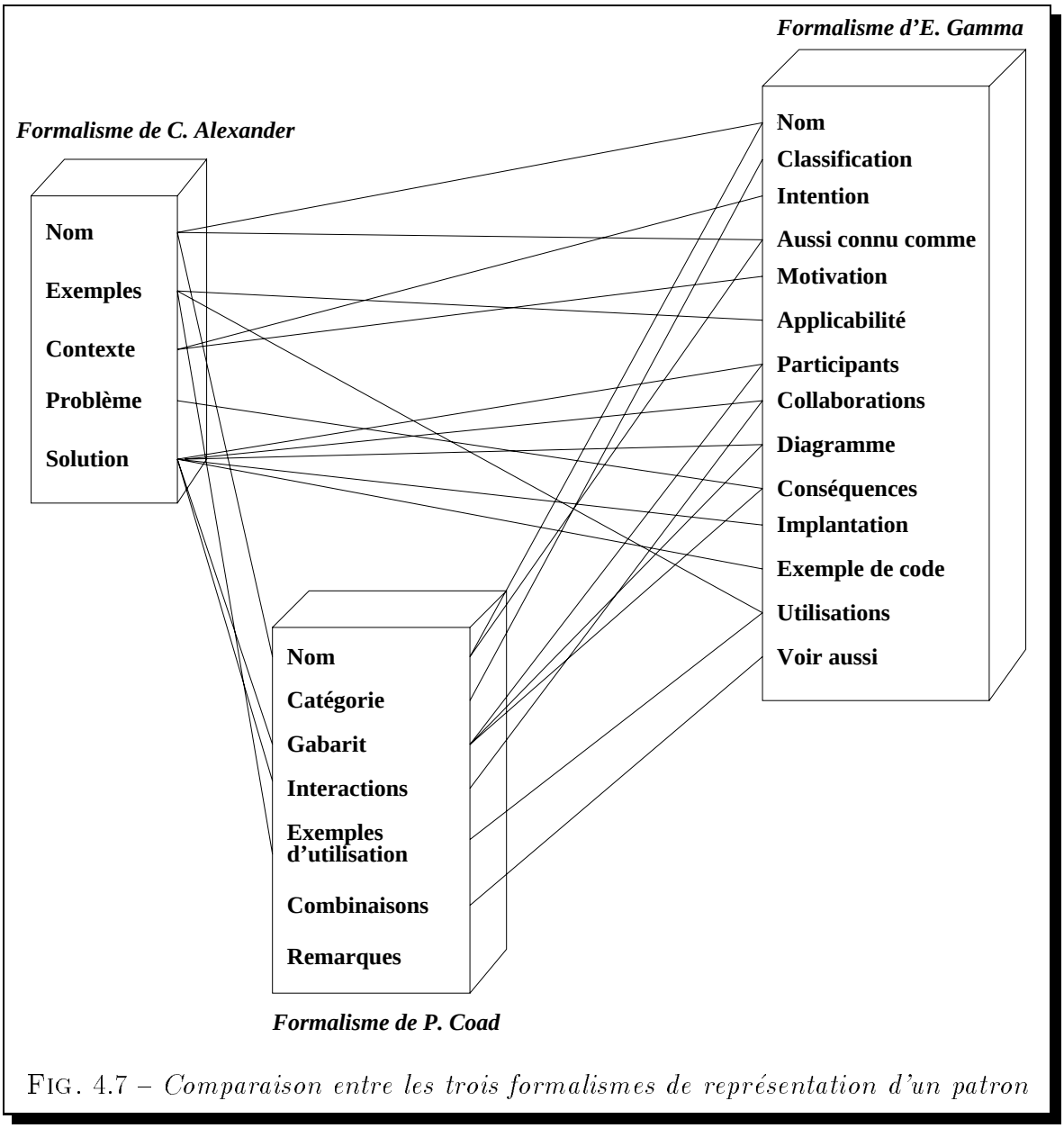


FIG. 4.7 – *Comparaison entre les trois formalismes de représentation d'un patron*

4.2.4 Classification des patrons

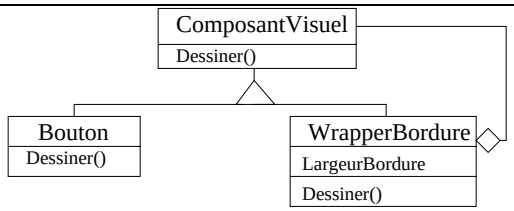
Un patron résout un problème spécifique du cycle de vie d'un système logiciel. On distingue ainsi trois principaux types de patrons : les patrons d'analyse aident à l'expression des besoins ; les patrons de conception résolvent des problèmes de conception particuliers ; les patrons d'implantation traduisent des solutions dans un langage cible.

4.2.4.1 Patrons d'analyse

P. Coad définit un patron d'analyse comme une "combinaison typique pour aider à construire des modèles" [CM96]. Les patrons d'analyse se situent au niveau de l'analyse des besoins d'une application et aident le concepteur d'une application dans la construction de modèles pour représenter les besoins de l'application. Ils identifient des problèmes répétitifs dans l'expression des besoins d'applications de différents domaines et transforment l'expression de ces besoins dans des modèles réutilisables. Les patrons Rôles [Coa92] et Ressources [Rol93] que nous avons présentés précédemment sont des patrons d'analyse utilisables dans plusieurs domaines d'applications (ils appartiennent donc à la couche Niveau 5 - Métier générique). Des patrons d'analyse peuvent aussi être proposés dans des domaines d'application spécifiques (Niveau 6 - Métier spécifique), comme le fait M. Fowler [Fow97] en identifiant des patrons d'analyse dans des domaines tels que la santé, le commerce, la comptabilité, la planification, etc. C'est aussi au niveau 6 que se situent les travaux du groupe interdisciplinaire POSEIDON dans le domaine des applications industrielles de type systèmes de gestion de données techniques pour systèmes d'information produits [RTBG97].

4.2.4.2 Patrons de conception

Les patrons de conception (*Design Patterns*) identifient, nomment et abstraient des thèmes communs du domaine de la conception orientée objet. Les patrons de conception capturent l'expérience et la connaissance liées à la conception en identifiant les objets, leurs collaborations et la distribution de leurs responsabilités. Ils jouent plusieurs rôles dans le processus de développement d'un logiciel : ils offrent un vocabulaire commun pour la conception, ils réduisent la complexité du système en nommant et en définissant des abstractions, ils constituent une base d'expérience pour construire des composants réutilisables, enfin ils agissent comme des blocs de construction à partir desquels des systèmes plus complexes peuvent être construits [GHJV93].

Rubrique	Signification
Nom	Wrapper.
Classification	Juridiction Composée - Caractérisation Structurelle.
Intention	Attache des services, des propriétés ou des comportements supplémentaires à des objets et peut être appliqué récursivement pour assigner des propriétés multiples aux objets.
Motivation	Parfois, il est souhaitable d'attacher des propriétés à des objets individuels plutôt qu'à des classes. Dans une interface graphique par exemple, des propriétés telles que des bordures, des ombres ou des ascenseurs ou des services tels que le scrolling ou le zoom peuvent être attachables à certains composants de l'interface. La solution proposée est d'*emballer* le composant dans un autre objet qui ajoute la bordure. L'objet emballant est transparent auprès des clients et est appelé Wrapper (*Emballage*). Le wrapper transfère les requêtes qu'il reçoit à ses composants et peut réaliser des actions additionnelles avant ou après la requête, tel que dessiner une bordure autour d'un composant.
Applicabilité	Quand des propriétés ou des comportements sont attachables de façon dynamique et transparente à des objets individuels, ou quand il est nécessaire d'étendre les classes dans une hiérarchie ; plutôt que de modifier les classes de base, le wrapper emballe alors les instances et leur ajoute des propriétés, des comportements ou des services. Ceci est particulièrement utile quand la classe de base provient d'une librairie et ne peut pas être modifiée.
Participants	**Composant** : l'objet auquel les propriétés ou les services sont ajoutés ; **Wrapper** : l'objet qui encapsule le composant ; il définit une interface qui maintient une référence à ce composant.
Collaborations	Le wrapper transmet les requêtes au composant. Il peut éventuellement réaliser des actions additionnelles avant ou après avoir transmis les requêtes.
Diagramme	
Conséquences	Utiliser Wrapper pour ajouter des propriétés est plus flexible qu'utiliser l'héritage : les propriétés peuvent être attachées et détachées simplement en changeant le wrapper. De plus, Wrapper évite de créer de nouvelles classes pour chaque combinaison de propriété et permet aisément de mixer différentes propriétés.
Utilisations	La plupart des interfaces orientées objets (par exemple l'interface de librairies SmallTalk, InterViews ou encore ET) utilisent Wrapper pour ajouter des outils graphiques ergonomiques aux widgets.

Implantation	L'implantation d'un ensemble de classes Wrapper est simplifiée par une classe de base abstraite qui assure que toutes les requêtes sont envoyées directement aux composants.
Voir aussi	**Adapter** : un wrapper est différent d'un adapter car il change seulement les propriétés d'un objet et ne modifie pas son interface ; **Composite** : un wrapper peut être considéré comme un composite dégénéré à seulement un composant. Cependant, un wrapper ajoute des services ou des propriétés et ne se situe pas dans le contexte de l'agrégation.

TAB. 4.3 – *Le patron de conception* Wrapper

De nombreux auteurs s'intéressent aux patrons de conception, en particulier le *Gang of Four* [Joh92] [GHJV93] [BJ94] [GHJV94]. Un exemple est le patron Wrapper [GHJV93] présenté dans la table 4.3 selon le formalisme de Gamma.

Le langage unifié UML définit un patron de conception comme un regroupement remarquable de classes collaborant à la réalisation d'un ou de plusieurs services particuliers [BRJ97a] [BRJ97b] [Lai97]. La notion de patron est ainsi liée à celle de collaboration générique utilisable plusieurs fois dans des conceptions différentes. Un patron est représenté par une ellipse en pointillé contenant uniquement le nom du patron (cf. figure 4.8). L'instance d'un patron est représentée par les classes participant au patron. Ces classes sont reliées au patron par des relations de dépendance (flèches en pointillé) étiquetées dans des noms de rôles utilisés comme des paramètres liés de manière à spécifier les éléments de chaque occurrence d'un patron à l'intérieur d'un modèle.

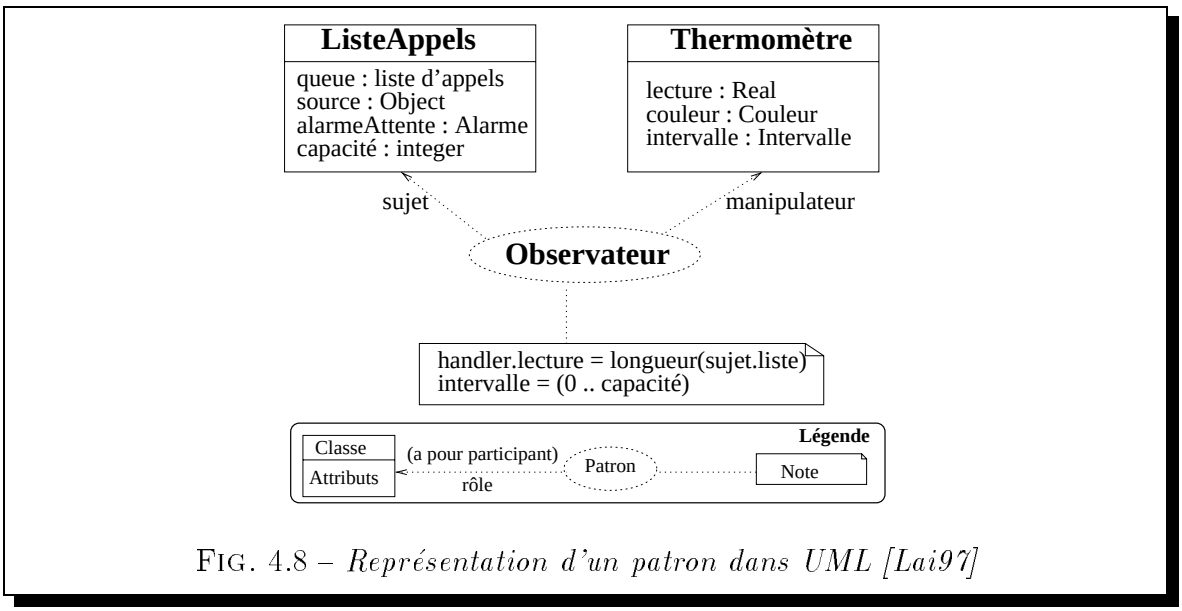


FIG. 4.8 – *Représentation d'un patron dans UML [Lai97]*

4.2.4.3 Patrons d'implantation

Les patrons d'implantation sont de bas niveau et en général spécifiques à un langage : ils décrivent comment implanter certains aspects particuliers des composants ou des relations entre eux dans un langage donné. Un exemple est donné en C++ avec le patron Type Promotion représenté dans la table 4.4 selon le formalisme de Gamma.

4.2.5 Utilisation des patrons

Les patrons sont utilisés dans une nouvelle approche de l'ingénierie des besoins qui s'oppose à l'approche descendante traditionnelle : l'approche par **couplage à base de patrons** (cf. figure 4.9) autorise ainsi une représentation commune des mêmes besoins du monde réel par des composants logiciels similaires dans un système particulier. Cette approche consiste en la proposition d'artéfacts prédéfinis et adaptables à des problèmes similaires et à des technologies différentes d'implantation.

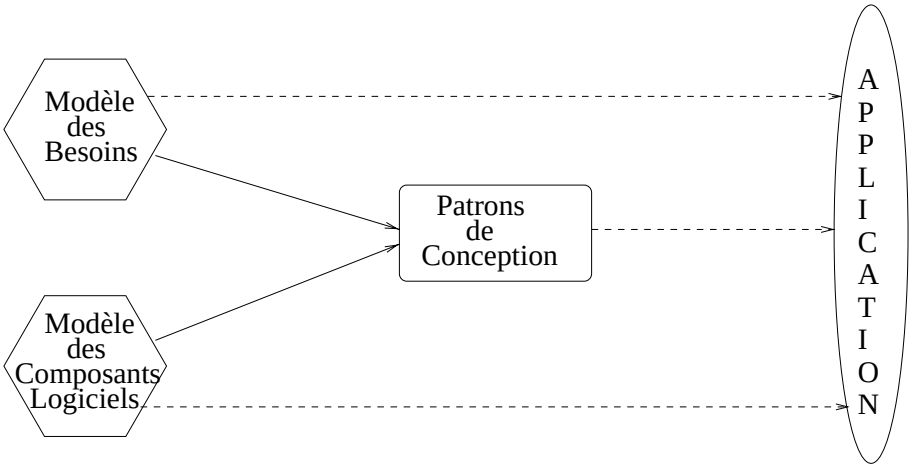


FIG. 4.9 – *Principe simplifié des approches à base de patrons*

Dans ce but, tout patron doit être :

- d'une part défini structurellement (classes composant le patron, relations entre ces classes, acteurs intervenant, etc.) et dynamiquement (diagrammes d'états des classes, diagrammes de flots de données, etc.),
- d'autre part implanté selon diverses technologies informatiques (programmation classique ou objet, bases de données relationnelles ou objets, etc.).

Rubrique	Signification
Nom	Type Promotion
Intention	Promotion entre des objets de différents types C++ intégrés ou non dans le programme ou importés d'une librairie dont le programmeur n'a pas la source.
Applicabilité	Le patron s'applique à C++ ou éventuellement à tout autre langage de programmation orienté-objet. Le choix de la promotion est effectué au moment de la compilation, mais le contexte est inadéquat pour que le compilateur puisse appliquer des règles de translation, par exemple entre types intégrés ou entre une classe dérivée et ses classes de base.
Motivation	L'implantation de la promotion entre un objet d'un type et un objet d'un autre type fait d'habitude partie de l'implantation de ces deux types. C++ permet au programmeur d'associer une telle implantation à l'un des types seulement. Le type contenant l'implantation de la conversion doit être une classe, puisque le programmeur ne peut pas redéfinir l'implantation des types intégrés ou des types exportés par des librairies. Deux mécanismes de langages supportent la définition des conversions par l'utilisateur : les constructeurs et les opérateurs de conversion. Individuellement, chacun d'eux apporte une solution adaptée au problème, mais l'utilisation des deux mécanismes mène à une ambiguïté.

Implantation Exemples de code	Un programme doit permettre la promotion d'un objet d'une classe vers un objet d'un type intégré ou importé d'une librairie en utilisant un opérateur de conversion :
	`class NombreRationnel {`
	`public :`
	`operator float() const;`
	`... };`
	
	Un programme doit utiliser des constructeurs pour les autres promotions :
	
	`class Complex {`
	`public :`
	`Complex(const NombreRationnel&);`
	`Complex(double)`
	`...};`

TAB. 4.4 – *Le patron d'implantation* Type Promotion

4.2.5.1 Les langages de patrons

Tout comme la notion de patron, le terme **langage de patrons** a été introduit par l'architecte C. Alexander. D'après J. Coplien [Cop96], un langage de patrons est "une collection structurée de patrons construits l'un sur l'autre pour transformer les besoins et les contraintes dans une architecture". Ce n'est pas un langage de programmation au sens ordinaire du terme, mais un document dont le but est de guider et d'informer le concepteur dans la conception d'un système en utilisant des patrons. Ainsi, chaque patron s'applique dans un contexte et transforme le système dans ce contexte en un nouveau système dans un nouveau contexte. Un autre patron peut alors être utilisé pour résoudre ce nouveau problème. Un patron étant une solution récurrente à un problème dans un contexte donné, un langage de patrons est un "ensemble de solutions qui travaillent ensemble pour résoudre un problème complexe selon une solution ordonnée relativement à un but prédéfini".

Un langage de patrons est donc une collection de patrons formant un vocabulaire pour comprendre et communiquer des idées. Les patrons doivent être tissés entre eux dans un tout cohésif qui révèle les structures et les relations inhérentes à chacun de ses composants dans l'atteinte d'un objectif commun. De nombreux langages de patrons sont actuellement proposés. Un exemple proposé par M. Fowler [Fow97] est donné dans l'annexe A et résumé dans la table 4.5. Ce langage a pour but de décrire la manière de détecter et de traiter les événements récurrents d'une application, par exemple Tous les lundis à 8h00. Le problème des événements récurrents est communément abordé par certains types de systèmes comme les SGBD temporels [FCS96].

4.2.5.2 Propriétés des patrons

D'après B. Appleton [App97], tout patron doit avoir les propriétés données ci-dessous.

- **Encapsulation**, c'est-à-dire la capacité à cacher la complexité d'un problème et d'une solution. En effet, les patrons sont indépendants, spécifiques et formulés précisément pour éclaircir le problème auquel ils s'appliquent et les solutions qu'ils offrent.
- **Simplicité**, c'est-à-dire la capacité à être utilisable par tous les participants au développement et non pas seulement par des concepteurs expérimentés.

Nom	Problème	Solution
Calendrier	Représenter des agents réagissant à des événements qui apparaissent récursivement certains jours.	Créer un objet Calendrier associé à l'agent.
Interface du calendrier	Difficulté de dire à quoi doit ressembler le calendrier.	Considérer qu'un objet Calendrier est déjà créé et imaginer la façon dont celui-ci serait utilisé. Déterminer les opérations clés de son interface.
Elément du calendrier	Représenter des événements récurrents sans les énumérer.	Créer des élément du calendrier associés à des événements et à des expressions temporelles.
Un jour Tous les Mois	Représenter des traitements de la forme *2^{eme} Lundi du mois.*	Utiliser une expression temporelle *Un Jour Tous les Mois* avec un jour de la semaine et un compteur du rang dans le mois.
Périodes d'Année	Représenter des traitements de la forme *du 14 Mars au 12 Octobre.*	Utiliser une expression temporelle *Périodes d'année* avec un jour de début et un jour de fin.
Expression Ensembliste	Représenter des combinaisons d'expressions temporelles.	Définir des combinaisons d'ensembles pour l'union, l'intersection et la différence.

TAB. 4.5 – *Le langage de patrons* Evénements Récurrents pour Calendriers *[Fow97]*

- **Equilibre**, grâce à un espace de solutions contenant un invariant qui minimise le conflit entre les forces et les contraintes. Pour être équilibré, un patron doit comporter des définitions théoriques et formelles, une abstraction des données mises en jeu, des observations du patron dans des cas réels, une série d'exemples convaincants et une analyse des solutions.

- **Abstraction** de l'expérience empirique et de la connaissance des développeurs d'applications.

- **Ouverture** à des niveaux de détail de plus en plus fins résolus par des patrons de plus en plus spécialisés pour permettre des ajouts de contraintes, des ajustements, des spécialisations et des raffinements spécifiques à l'application développée.

- **Composabilité**, c'est-à-dire la capacité à être organisé hiérarchiquement avec d'autres patrons et à être utilisé à l'intérieur d'un langage de patrons.

4.2.5.3 Intérêts de l'approche à base de patrons

L'utilisation systématique de patrons facilite la conception d'un modèle en permettant à une démarche de conception de procéder par assemblages et connexions d'instances de patrons. En effet, une approche à base de patrons s'intégre à une méthode de conception orientée objet classique comme OOA ou OMT et permet en particulier de découvrir les classes d'objets et les associations pertinentes et d'organiser le modèle du système d'information en sous-modèles ou en paquetages. Le modèle devient ainsi organisé et plus facile à comprendre. La qualité de la conception en est augmentée, les patrons mettant en œuvre des mécanismes éprouvés. Une telle approche est donc prometteuse quel que soit le domaine d'ingénierie et permet en particulier :

- d'archiver et de réutiliser des solutions tant conceptuelles que techniques éprouvées dans un même domaine ou dans des domaines différents. La réutilisation se fait par "bloc" et il est ainsi possible de conserver et de réutiliser des schémas de plus haut niveau que ceux de classes et de types offerts par les modèles objets traditionnels,

- de communiquer en des termes compréhensibles par les acteurs du domaine. En particulier, on ne parlera plus de classes ou de types, mais de ressources, de contrats, de contrôles, de rôles, etc.

- de guider une activité d'ingénierie en organisant hiérarchiquement et fonctionnellement les problèmes et leurs solutions.

La notion de patron étant récente, il n'existe pour l'instant pas de règles précises concernant son utilisation. Une méthode guidant l'utilisation des patrons pour concevoir des applications est nécessaire, mais aucune n'a encore été proposée pour l'instant. De nos jours, les patrons sont généralement utilisés de deux manières différentes :

- **Réutilisation directe de composants** : la réutilisation directe de composants concerne le cas où les besoins cernés pour l'application correspondent exactement à des patrons pré-définis. Il suffit dans ce cas de trouver le ou les patrons correspondant aux besoins et d'appliquer la solution proposée ;

- **Adaptation ou spécialisation de composants plus génériques** : dans ce cas, il n'existe pas de patrons exactement adaptés aux besoins de l'application. Il faut alors adapter ou spécialiser des patrons existants pour qu'ils correspondent

aux besoins cernés pour l'application. Dans ce cas, la solution proposée ne correspond pas exactement aux besoins de l'application, et il faut aussi adapter ou spécialiser la solution.

En pratique, le développement d'une application par utilisation de l'approche à base de patrons combine systématiquement réutilisation directe de composants et adaptation ou spécialisation de composants plus génériques.

4.2.5.4 Variante : patrons de conception et frameworks

La notion de framework est proche de celle de patron. En effet, les définitions traditionnelles d'un framework sont les suivantes :

- un ensemble de classes reliées que l'on spécialise et/ou instancie pour implanter une application ou un système,
- une conception réutilisable d'un programme exprimé comme un ensemble de classes,
- une collection de classes qui fournit un ensemble de services pour un domaine particulier ; un framework exporte ainsi un nombre de classes individuelles et de mécanismes que les clients peuvent utiliser ou adapter [Boo91].

Cependant, on constate en regardant plus précisément chacune de ces deux notions que des différences existent.

Tout d'abord, une définition plus précise d'un framework est la suivante [App97] : *un framework est une micro-architecture réutilisable qui fournit la structure générique et le comportement pour une famille d'abstractions logicielles, dans un contexte qui spécifie leurs collaborations et leur utilisation à l'intérieur d'un domaine donné. Cette micro-architecture est accomplie en codant le contexte dans une sorte de "machine virtuelle", une unité de travail désignée avec des "points-tampons" spécifiques implantés à l'aide de polymorphisme ou de rappel automatique. Ainsi, le framework peut être adapté et étendu à divers types de besoins et de domaines ou être composé avec d'autres frameworks. Un framework supporte l'infrastructure et les mécanismes qui exécutent l'interaction entre des composants abstraits dans des implantations ouvertes.*

De plus, certains patrons décrivent des frameworks et sont employés dans la conception et la documentation de frameworks. De tels patrons peuvent être vus comme des

descriptions abstraites de frameworks, lesquels facilitent la réutilisation large d'architectures logicielles. Similairement, les frameworks peuvent être vus comme des réalisations concrètes de patrons, lesquels facilitent la réutilisation directe de conception et de code. Un framework recouvre ainsi plusieurs patrons de conception et peut être vu comme l'implantation d'un système de patrons.

Enfin, les patrons sont décrits indépendamment du langage et représentent de la connaissance et de l'expérience sur le développement d'un logiciel alors que les frameworks sont généralement exécutables et implantés dans un langage particulier. Ainsi, les frameworks sont de nature physique alors que les patrons sont de nature logique : les frameworks sont la réalisation physique d'une ou de plusieurs solutions logicielles alors que les patrons sont les instructions pour implanter ces solutions.

4.3 Conclusion : vers une adaptation...

L'approche à base de patrons représente aujourd'hui un apport capital pour le développement de logiciels à moindre coût. Elle permet de réutiliser lors du développement d'une nouvelle application des techniques éprouvées depuis le niveau de l'expression des besoins jusqu'à celui de l'implantation dans un système cible.

Cet objectif de meilleure réutilisation est l'un de ceux que nous cherchons à atteindre dans nos propositions. Par ailleurs, le chapitre 2 a mis en évidence une classification de situations comportementales communes à plusieurs domaines d'application et regroupées dans le modèle unifié **Situation-Réaction** qui constitue un ensemble de besoins du monde réel communs à plusieurs domaines d'applications. Un patron ayant pour but de "décrire avec succès des solutions récurrentes à des problèmes logiciels communs dans un certain contexte", le parallèle entre les situations-réactions mises en évidence et ces problèmes logiciels apparaît évident.

La suite de ce document est basée sur cette constatation et présente une adaptation de l'approche des patrons afin de permettre la réutilisation de techniques tant au niveau de l'expression des situations comportementales d'une application qu'à celui de leur implantation.

Chapitre 5

Proposition de patrons pour les applications réactives

Le parallèle entre les situations-réactions communes à plusieurs domaines d'application et les problèmes logiciels communs qu'expriment les patrons dans un contexte particulier nous amène à nous intéresser à une adaptation de l'approche des patrons dans le cadre de la représentation de situations comportementales. Nous adoptons cette approche avec deux objectifs principaux :

- contribuer à la réutilisation de connaissances dès le niveau de l'analyse des besoins ; nous appelons cet objectif **objectif de réutilisation** ;
- réutiliser des mêmes mécanismes au niveau de l'analyse pour aboutir à des implantations diverses (systèmes de gestion de bases de données actifs, mais aussi systèmes de gestion de bases de données déductifs ou encore systèmes à base de connaissances, etc.) ; nous appelons cet objectif **objectif de généricité**.

5.1 Prise en compte des situations comportementales avec les patrons existants

Les situations comportementales présentent des caractéristiques particulières qu'aucun patron existant à notre connaissance n'est en mesure de prendre en compte. En étudiant le cahier des charges d'applications réactives, nous nous situons en effet au niveau de la représentation et de l'analyse des besoins comportementaux des applications. Les patrons susceptibles de représenter des situations comportementales sont

donc naturellement des patrons d'analyse. Or, les patrons d'analyse proposés aujourd'hui ne sont pas aptes à représenter complètement les situations comportementales construites selon le modèle **Situation-Réaction**.

Le patron **Mémoire d'événements** [Coa92] rend un objet responsable de détecter qu'un événement le concernant est apparu. En effet, souvent, des événements apparaissent qui doivent être détectés et mémorisés dans le but d'être traités ou analysés. Ainsi dans le domaine de la manufacture, le seuil de tolérance d'un outil peut être dépassé. De la même manière peut se produire dans le domaine médical le dépassement du taux de diabète autorisé pour un patient. Enfin, dans un système de gestion de stock, tout passage en dessous du seuil limite doit être contrôlé et corrigé par un processus de réapprovisionnement. Le patron "Mémoire d'événements" (cf. figure 5.1) mémorise de tels événements avec la date et la valeur caractéristiques de leur occurrence, en particulier pour gérer des synchronisations avec d'autres événements.

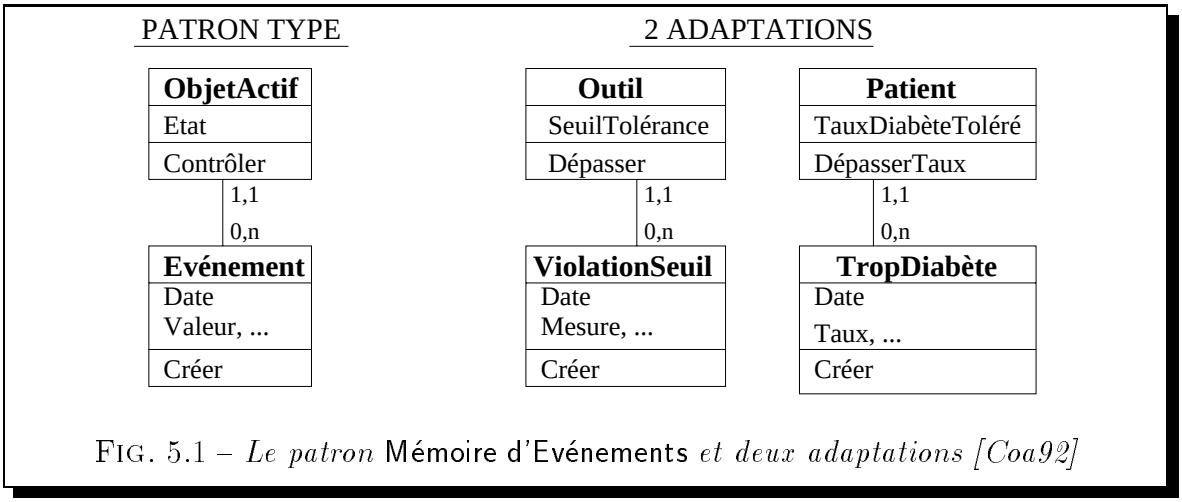


FIG. 5.1 – *Le patron* Mémoire d'Evénements *et deux adaptations [Coa92]*

Ce patron appartient à l'ensemble des sept patrons proposés par P. Coad dans l'un des premiers travaux sur la notion de patrons [Coa92]. De par l'originalité de cette nouvelle approche et parce qu'il paraissait bien adapté pour représenter des situations susceptibles d'être prises en compte par les nouvelles technologies de bases de données actives expérimentées dans notre équipe, ce patron constitue le point de départ de notre étude sur les patrons. Cependant, il ne considère que l'aspect production d'événement anormal local à un objet d'une classe et n'est adapté ni pour représenter la production d'événements globaux, ni pour exprimer l'occurrence d'événements normaux. De plus, il ne prend pas en compte la réaction à des événements ni la synchronisation avec d'autres événements pour une réaction globale. Ce patron ne convient donc pas pour représenter complètement les situations comportementales où les situations sont

complexes et où la composante "réaction" est primordiale.

Le langage de patrons **Evénements récurrents** [Fow97] (cf. annexe A) regroupe plusieurs patrons dans le but de représenter des événements récurrents tels que *Tous les lundis à 10h00*. Cependant, il n'est pas apte à représenter les situations comportementales pour la raison principale que les événements récurrents ne couvrent pas toutes les situations comportementales possibles. L'aspect réaction est pris en compte par le fait qu'un événement apparaît chaque fois qu'une date est atteinte. Cette solution est un bon point de départ pour représenter le comportement des applications, mais ne convient pas pour représenter tous les types de situations comportementales et de réaction de l'application à des événements.

Enfin, le patron **Producteur-Ressource-Consommateur** [Rol93] [RTBG97] est spécialisé dans la gestion de ressources. Il prend en compte l'aspect statique des ressources grâce à un diagramme des classes intervenant dans la gestion des ressources et considère le comportement des ressources. De plus, il est complété par des diagrammes de transitions d'états modélisant l'aspect dynamique des ressources. Ce double apport est intéressant et satisfaisant. Cependant, la spécialisation de ce patron pour la gestion des ressources ne convient pas pour la représentation des situations comportementales fondées sur des événements plus complexes que les ressources.

Constatant que les patrons d'analyse existants ne sont pas suffisants pour représenter complètement les situations comportementales telles que celles que nous avons présentées dans la section 2.2.2, nous introduisons un ensemble de nouveaux patrons et le langage **SCalP**. Ce langage de patrons est destiné à guider un concepteur d'applications dans la représentation et la réutilisation des situations comportementales à l'aide des patrons introduits [Fro97].

5.2 Nouveaux patrons et langage SCalP

Nous utilisons le formalisme de représentation d'E. Gamma pour présenter le langage **SCalP** (Situations Comportementales à l'aide de Patrons) pour deux raisons principales. D'abord, il recouvre les formalismes de C. Alexander et de P. Coad tout en étant plus détaillé et plus apte à capturer différents aspects. De plus, il a une forme plus proche d'un utilisateur car exprimé globalement en langue naturelle.

Pour atteindre notre but de généricité, nous ne visons pas pour l'instant un système cible particulier, mais nous nous concentrons sur la représentation des situations

comportementales dès l'analyse des besoins de l'application. Le chapitre 6 établit la faisabilité de notre approche pour la conception dans le cadre d'un système cible particulier : les systèmes de gestion de bases de données actifs. La partie Exemple de code du formalisme de représentation d'E. Gamma est par conséquent en particulier reportée à ce chapitre. Enfin, pour une meilleure compréhension des concepts que nous introduisons, nous ajoutons au formalisme d'E. Gamma une partie Méta-modèle qui utilise une représentation graphique type OMT [RBP+91] pour représenter le méta-modèle des concepts introduits.

5.2.1 Le patron Situation-Réaction

La première étape consiste à traduire les situations comportementales d'une application selon un même modèle **Situation-Réaction** exprimant une réaction à une situation normale ou anormale. Cette étape est traitée par le patron **Situation-Réaction** qui met en évidence les deux classes Situation et Réaction reliées par l'association engendre : une situation engendre une réaction. On ne prend en compte que des situations engendrant au moins une réaction et on considère qu'une situation engendre une seule réaction. En effet, même si elle est complexe, la réaction prise en compte ici est globale et unique. Pour la même raison, on considère qu'une réaction correspond à une et une seule situation.

La manière la plus simple de passer d'une situation comportementale exprimée dans un langage naturel par définition informel à un modèle Situation-Réaction consiste à traduire la situation comportementale selon l'expression informelle standardisée **Si... Alors....** Cette expression conditionnelle très utilisée dans les langages de programmation a l'avantage d'être suffisamment naturelle et facilement compréhensible. De plus, elle s'adapte très bien au modèle *Situation-Réaction* en exprimant que *si une Situation se produit, alors une Réaction doit être engendrée.* C'est le but du patron Situation-Réaction exprimé selon le formalisme d'E. Gamma dans la table 5.1.

5.2.2 Le patron Production-Evénement-Consommation

Le patron précédent permet d'exprimer une situation comportementale selon l'expression générale Si... Alors... représentant le modèle commun Situation-Réaction. L'étape suivante consiste à décomposer la situation et la réaction. En particulier, il est nécessaire de savoir exactement quand la situation engendre une réaction.

Rubrique	Signification
Nom	Situation-Réaction.
Classification	Juridiction composée, caractérisation comportementale.
Intention	Traduire les situations comportementales selon l'expression **Si... Alors....**
Motivation	Transformer des situations comportementales dans le modèle **Situation-Réaction**.
Applicabilité	Toute situation comportementale exprimant une réaction à une situation normale ou anormale.
Participants	Le patron introduit deux composants principaux : une Situation et une Réaction. Les deux composants sont rassemblés selon l'expression **Si Situation, Alors Réaction**.
Collaborations	La Situation caractérise une situation générale normale ou anormale. La Réaction représente la réaction à cette situation.
Méta-modèle	Le patron met en jeu deux classes : une classe Situation et une classe Réaction reliées entre elles par l'association engendre. Une situation engendre une et une seule réaction. Une réaction est engendrée par une et une seule situation. 
Conséquences	Représentation de toutes les situations comportementales selon l'expression *Si... Alors...*
Utilisations	Si le médecin prescrit de l'aspirine à une femme enceinte, Alors il doit changer le traitement. Si on détermine qu'un patient est atteint de diabète, Alors il faut contrôler son taux de glycémie tous les 3 mois.
Voir aussi	**Evénements récurrents** [Fow97] : ce langage de patrons prend en compte un type particulier de situations comportementales (par exemple *Tous les lundis à 8h00, mettre à jour le planning des infirmières*), mais ne permet pas de représenter tous les types de situations comportementales possibles.

TAB. 5.1 – *Le patron* Situation-Réaction

Une **situation** est composée de plusieurs éléments :

- une ou plusieurs **entités** participant à la situation (un attribut d'un objet d'une classe, un objet d'une classe, une classe, l'application, etc.) ;
- un **événement** au sens de la définition encyclopédique donnée dans la section 1.2 : un événement est "tout fait important ou toute circonstance marquante, à un certain moment et dans un certain contexte". Un événement peut donc aussi bien être un événement qui arrive (donc dynamique) qu'un fait qui est (donc statique). Des événements aussi divers que *un traitement est donné à une patiente* ou *un traitement composé d'aspirine est donné à une patiente enceinte* sont autorisés. Cependant, pour représenter la situation comportementale *il ne faut pas prescrire d'aspirine à une patiente enceinte*, l'événement *un traitement composé d'aspirine est donné à une patiente enceinte* porte plus de sémantique. Par contre, l'événement *un traitement composé d'aspirine est prescrit à une patiente* est pertinent pour représenter la situation comportementale *il faut contrôler qu'une patiente n'est pas enceinte avant de lui prescrire de l'aspirine*. Les événements d'une situation sont donc divers et d'un haut niveau conceptuel.

  De même, une **réaction** est identifiée par deux éléments principaux :

  - une ou plusieurs **entités** participant à la réaction (un attribut d'un objet d'une classe, un objet d'une classe, une classe, l'application, etc.) ;
  - un **type** de réaction : la réaction peut être destinée soit à continuer une activité selon un processus normal et attendu, soit à retrouver une configuration normale après une violation de contrainte par exemple. Dans ce cas, la réaction peut soit engendrer un retour à une situation normale en annulant ce qui a été fait depuis que la contrainte a été violée, soit provoquer la modification des entités de l'application de façon à obtenir un résultat satisfaisant et une contrainte non violée. Par exemple, pour maintenir la situation comportementale *l'augmentation du salaire d'un agent ne doit pas être supérieure à 1000 francs*, la réaction engendrée suite à l'événement *le salaire a été augmenté de plus de 1000 francs* peut être soit une annulation complète de l'augmentation, soit une augmentation du salaire de l'agent plafonnée à 1000 francs.

  Nous acceptons alors de considérer le concept d'**événement** de la situation comme composant central, d'autant plus que comme nous l'avons vu dans la section 3.2, l'événement est la notion de base pour la modélisation du comportement

d'une application. Il est alors nécessaire de savoir d'une part si cet événement était inattendu ou au contraire entièrement prévisible, d'autre part comment il a été produit. Nous introduisons dans ce but le patron **Production-Evénement-Consommation** qui vise à mettre en évidence la façon dont l'événement correspondant à la situation a été produit dans une Situation de Production ou **Production**. De la même manière, il fait ressortir la façon dont l'événement est consommé dans une Situation de Consommation ou **Consommation**. Ce patron est présenté dans la table 5.2 selon le formalisme d'E. Gamma.

Le niveau de granularité des situations-réactions se raffine avec ce patron. En effet, dans le patron Situation-Réaction, une situation engendre une et une seule réaction et une réaction correspond à une et une seule situation. Ici, la situation se décompose en un ou plusieurs événements dont chacun est produit par une ou plusieurs situations de production et consommé par une ou plusieurs situations de consommation.

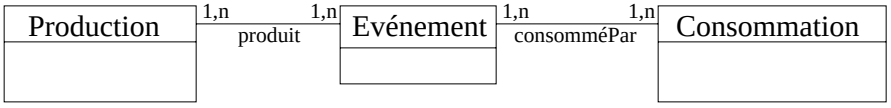
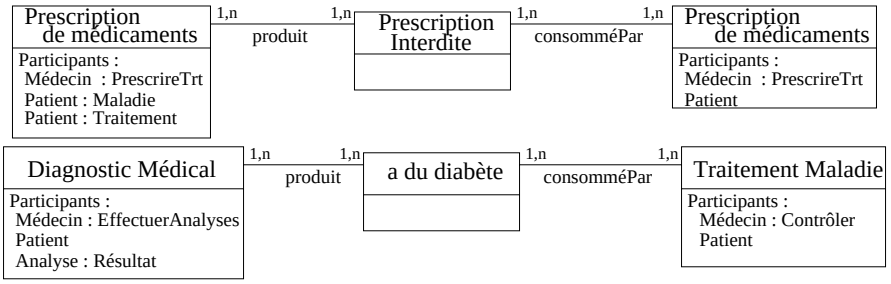
Dans le cas de la production d'un événement, la **synchronisation** d'un événement traduit le cas où l'événement est la combinaison de plusieurs situations de production (cf. figure 5.4). La **portée** d'une situation de production caractérise le cas où une même situation de production produit plusieurs événements distinguables.

Dans le cas de la consommation d'un événement, la **synchronisation** d'une consommation exprime le cas où la réaction à l'événement correspond à la consommation de plusieurs événements en même temps ; la consommation d'un événement est alors liée à la consommation d'autres événements (cf. figure 5.4). La **portée** d'un événement traduit le fait qu'un événement est consommé par plusieurs consommateurs.

5.2.3 Le patron Evénement

Le patron `Production-Evénement-Consommation` met en exergue le concept d'événement dans la représentation de situations comportementales : une situation comportementale est composée d'un ou de plusieurs événements qui sont produits et consommés. Le patron **Evénement** donné dans la table 5.3 a pour but la définition de cette notion centrale.

De par sa définition encyclopédique, un événement est *tout ce qui se produit, arrive ou apparaît, tout fait ou toute circonstance importante ou marquante.* Un

Rubrique	Signification
Nom	Production-Evénement-Consommation.
Classification	Juridiction composée, caractérisation comportementale.
Intention	Décomposer des situations comportementales exprimées sous la forme **Si... Alors...** avec le patron Situation-Réaction en des événements produits et consommés.
Motivation	Reconsidérer les situations comportementales en terme de production et de consommation d'événements.
Applicabilité	Toute Situation-Réaction exprimée sous la forme *Si... Alors.*
Participants	Le patron met en jeu les classes Production, Evénement et Consommation reliées entre elles par les associations produit et consommé par.
Collaborations	La classe Evénement est la classe centrale. Elle caractérise un événement qui apparaît ou une situation qui est. Une instance de la classe Evénement est reliée à une ou plusieurs instances de la classe Production par l'association n-p produit signifiant qu'une ou plusieurs situations de production produisent un ou plusieurs événements. De même, elle est reliée à une ou plusieurs instances de la classe Consommation par l'association n-p consomméPar signifiant qu'un ou plusieurs événements sont consommés par une ou plusieurs situations de consommation. Les instances des classes Production et Consommation ont un nom et sont caractérisées par des participants, instances de classes mises en évidence dans le modèle objet de l'application.
Diagramme	
Conséquences	Caractérisation d'un ou de plusieurs événements centraux produits par une ou plusieurs situations de production et consommés par une ou plusieurs situations de consommation.
Utilisations	
Voir aussi	**Mémoire d'événements** [Coa92] : mémorise un événement local à un objet d'une classe, mais la réaction à son occurrence ou la synchronisation avec d'autres événements pour une réaction globale ne sont pas prises en compte. **Producteur-Ressource-Consommateur** [Rol93] [RTBG97] : gère des ressources et non pas des événements.

TAB. 5.2 – *Le patron* Production-Evénement-Consommation

Rubrique	Signification
Nom	Evénement.
Classification	Juridiction composée, caractérisation comportementale.
Intention	Détailler la notion d'événement mise en évidence dans le patron Production-Evénement-Consommation.
Motivation	Le concept d'événement est une notion centrale dans la représentation de situations comportementales.
Applicabilité	Tout événement d'une situation comportementale mis en évidence par le patron Production-Evénement-Consommation.
Participants	Le patron détaille la classe Evénement mise en évidence par le patron Production-Evénement-Consommation en lui affectant un nom et en instanciant son attribut Mémorisation.
Collaborations	La classe Evénement représente la classe centrale du patron Production-Evénement-Consommation. Elle caractérise un événement qui apparaît ou une situation qui est. Elle possède un nom et un attribut Mémorisation qui indique si l'événement doit être mémorisé (attribut Mémorisation à mémorisé) ou non (attribut Mémorisation à nonMémorisé).
Diagramme	
Conséquences	Un événement a un nom et un attribut Mémorisation qui indique s'il doit être mémorisé ou non.
Utilisations	

TAB. 5.3 – *Le patron* Evénement

événement concerne donc aussi bien une situation du monde réel qui est qu'un événement qui se produit. Un événement a un **nom** et un attribut qui indique s'il est **mémorisé** ou non. Un événement mémorisé est un événement important qui doit être gardé en mémoire tout au long de l'existence de l'application. Par exemple, l'événement *on a découvert que le patient Dupont avait du diabète* doit être mémorisé car il implique de nouveaux traitements qu'il est dorénavant nécessaire d'effectuer sur le patient, par exemple un contrôle de son taux de diabète par prise de sang tous les 3 mois. Par contre, l'événement *le médecin a prescrit de l'aspirine à une patiente enceinte* n'est pas mémorisé : il est nécessaire de réagir immédiatement à cet événement en changeant le traitement, mais une fois la réaction effectuée, l'événement peut ne pas être mémorisé car il n'a pas d'importance quelconque sur le déroulement à long terme de l'application.

5.2.4 Le patron Action

Le patron Evénement nous a permis de préciser l'événement central des situations comportementales. Il nous faut maintenant savoir exactement comment cet événement est produit et consommé. On peut cependant constater que les situations de production et les situations de consommation sont symétriques et se différencient simplement par leur nature : une situation de production **produit** un ou plusieurs événements alors qu'une situation de consommation **consomme** un ou plusieurs événements. Nous introduisons donc une nouvelle classe appelée **Action** (cf. figure 5.2) dont hérite chacune des deux classes Production et Consommation introduites par le patron Production-Evénement-Consommation.

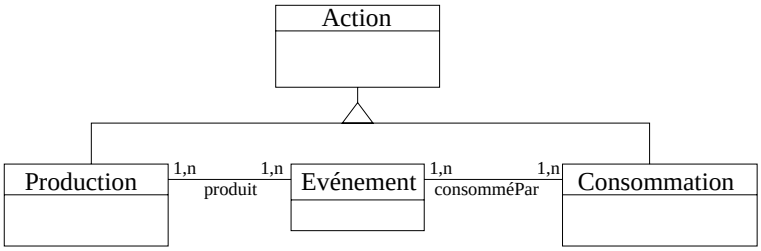


FIG. 5.2 – *Généralisation des classes Production et Consommation*

5.2.4.1 Définition du patron Action

Le patron Action exprimé dans la table 5.4 grâce au formalisme d'E. Gamma a deux buts principaux. D'une part, il vise à **décomposer** la classe Action en trois parties distinctes : un Contexte, des Concepts et une Condition. D'autre part, il permet d'**instancier** chacune de ces trois parties (cf. figure 5.3).

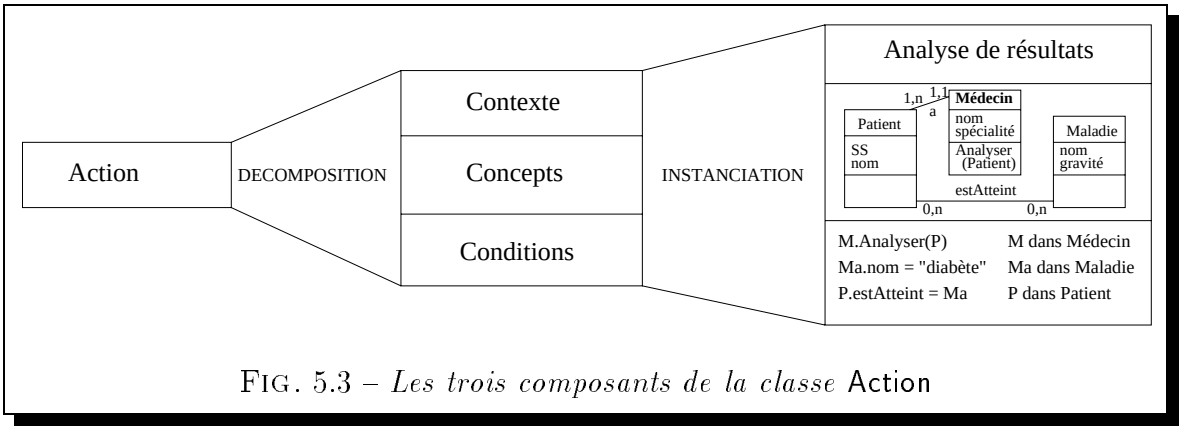
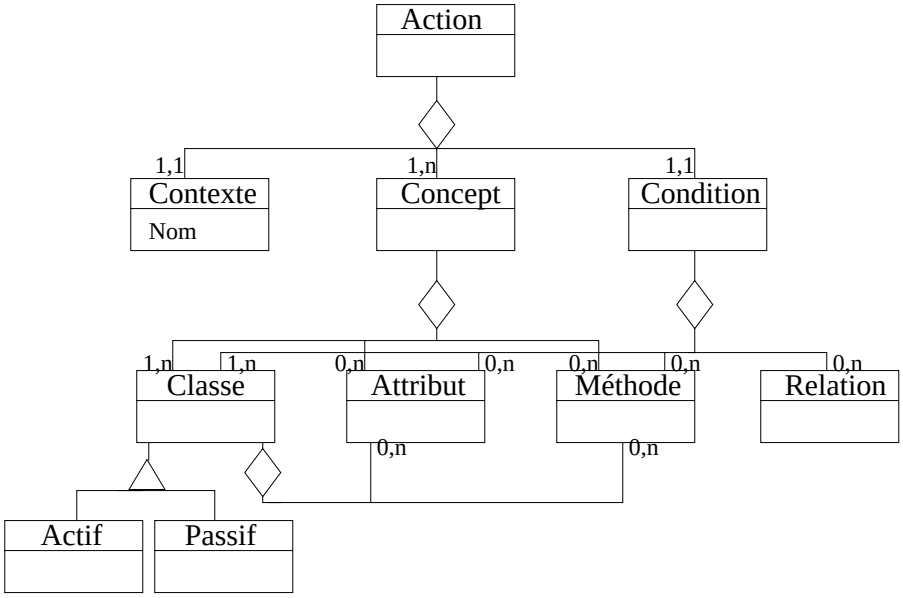
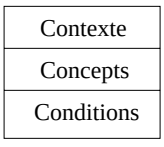


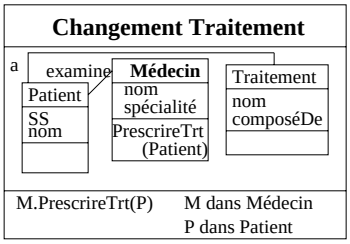
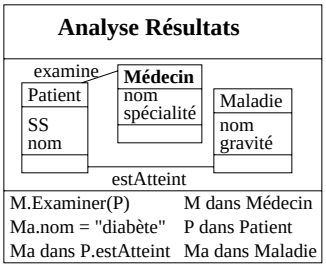
FIG. 5.3 – *Les trois composants de la classe* Action

1. **Contexte** : le contexte est le nom d'une action ; il représente le cadre dans lequel se déroule cette action et est exprimé de manière informelle.
2. **Concepts** : les Concepts sont un ou plusieurs acteurs d'une action. Ces acteurs sont des classes du modèle statique de l'application entrant en jeu dans la réalisation de l'action, éventuellement augmentées des attributs concernés par l'action ou des méthodes appelées pour réaliser l'action.

   L'instanciation de la partie Concepts d'une action est réalisée en utilisant une notation graphique semi-formelle englobante : les acteurs sont représentés à l'intérieur de la partie Concepts d'une action de la même façon qu'un diagramme d'objets dans la méthode OMT [RBP+91] (cf. figure 5.3).

   Enfin, il est nécessaire de distinguer les acteurs **actifs** et les acteurs passifs : un acteur est **actif** (notation en caractères gras) au sens où il participe au déclenchement de l'action. Un acteur passif (notation par défaut) subit l'action sans y participer.
3. **Conditions** : les conditions portent sur les concepts de l'action et doivent être vérifiées pour que l'action se réalise : une condition porte sur la valeur des attributs des classes de la partie Concepts ou montre les méthodes déclenchées pour que l'action se réalise ; d'autre part, une condition établit le lien entre les acteurs de la partie Concepts en instanciant les relations

Rubrique	Signification
Nom	Action.
Classification	Juridiction composée, caractérisation comportementale.
Intention	Détailler les situations de production et de consommation introduites dans le patron Production-Evénement-Consommation.
Motivation	Les situations de production et de consommation sont symétriques et spécialisent par leur nature la classe Action.
Applicabilité	Toute situation de production et toute situation de consommation mise en évidence dans une situation comportementale par le patron Production-Evénement-Consommation.
Participants	En instanciant la classe Action, le patron permet d'instancier les classes Production et Consommation mises en évidence par le patron Production-Evénement-Consommation. L'instanciation de la classe Action implique l'instanciation de classes présentes dans le méta-modèle de cette classe (cf. partie Méta-modèle) et est réalisée en utilisant une représentation graphique englobante (cf. partie Diagramme).
Méta-modèle	
Collaborations	Les classes composant la classe Action sont instanciées avec la classe Action en utilisant une représentation graphique englobante (cf. partie Diagramme).
Diagramme	
Conséquences	Une action comporte un nom informel, une partie semi-formelle illustrant les concepts participant à l'action et une partie plus formelle introduisant les conditions à vérifier pour que l'action ait lieu. Tous les éléments nécessaires pour expliciter la production et la consommation d'un événement sont définis.

Utilisations	Le premier exemple correspond à la consommation de l'événement PrescriptionInterdite déclenché suite à la prescription d'aspirine à une femme enceinte ; le deuxième exemple illustre la production de l'événement a du diabète correspondant à la détection de diabète chez un patient.  

TAB. 5.4 – *Le patron* Action

existant entre les concepts. La partie Conditions est proche de la notion de contrainte de UML (dans UML [Mul97], une contrainte est une relation sémantique quelconque entre éléments de modélisation), mais les conditions prennent en compte en plus le déclenchement des méthodes des classes.

L'instanciation de la partie Conditions est réalisée en utilisant une notation "englobante" : les conditions sont représentées à l'intérieur de la partie Conditions d'une action. Comme dans UML, et pour rester proche d'un niveau utilisateur, nous ne spécifions pas de syntaxe particulière pour les conditions qui peuvent ainsi être exprimées en langage naturel, en pseudo-code, par des expressions de navigation ou par des expressions sémantiques.

5.2.4.2 Du patron Action vers la classe Production

Une Production est une spécialisation d'une Action. Comme toute Action, une Production regroupe un contexte, des concepts et des conditions. Parmi les concepts, un ou plusieurs acteurs peuvent être actifs : nous les appelons **Producteurs**. Par exemple, dans le contexte de prescription de médicaments, le producteur de l'événement *PrescriptionInterdite* est un **Médecin** qui prescrit de l'aspirine à une patiente enceinte.

Dans certains cas cependant, il est nécessaire de réagir non pas à un événement qui est produit, mais plutôt à un événement qui "est". Par exemple, en considérant la situation comportementale *il faut contrôler tous les trois mois le taux de sucre dans le sang des patients atteints de diabète*, c'est bien le fait que le patient

soit atteint de diabète qui nous intéresse et non pas (tout au moins pas dans ce cas) le fait de savoir pourquoi le patient a du diabète. Dans de telles situations, aucun producteur n'est actif.

5.2.4.3 Du patron Action vers la classe Consommation

De la même manière qu'une Production, une Consommation est une spécialisation d'une Action. Elle regroupe donc un contexte, des concepts et des conditions. Certains de ses concepts peuvent être actifs et sont appelés **consommateurs**. Par exemple, dans le contexte de prescription de médicaments, le consommateur de l'événement *PrescriptionInterdite* est le médecin qui change le traitement de la patiente enceinte. Dans certains cas, le ou les consommateurs ne sont pas connus : c'est alors l'application elle-même qui réagit en alertant l'utilisateur ou en corrigeant une situation anormale.

Selon le cas, l'événement est consommé en rétablissant la situation normale, en avertissant l'utilisateur d'une situation anormale ou en déclenchant une action pour résoudre le problème ou pour pouvoir continuer l'activité.

5.2.5 Les patrons RelationProduit et RelationConsomméPar

La dernière étape à réaliser pour représenter complètement une situation comportementale consiste à instancier les relations produit et consomméPar. Cette étape est réalisée par les patrons RelationProduit et RelationConsomméPar représentés selon le formalisme d'E. Gamma respectivement dans les tables 5.5 et 5.6. Dans le contexte d'application de ces patrons, *instancier* n'a pas le sens commun utilisé avec la notion d'objet : instancier signifie ici attribuer un ensemble de caractéristiques appelées attributs aux relations produit et consomméPar.

Le patron Action a montré que les situations de production et de consommation se différenciaient par leur nature. Il est aisé de constater que les relations produit et consomméPar sont différentes pour la même raison : la relation produit lie une production à un événement en spécifiant qu'une situation de production *produit* un ou plusieurs événements ; la relation consomméPar lie un événement à une consommation en spécifiant qu'un événement est *consommé par* une ou plusieurs situations de consommation.

Rubrique	Signification
Nom	RelationProduit.
Classification	Juridiction composée, caractérisation comportementale.
Intention	Affecter des valeurs aux attributs de la relation produit.
Motivation	Instancier la relation produit introduite dans le patron Production-Evénement-Consommation.
Applicabilité	La relation produit d'une situation comportementale.
Participants	La relation produit associe un ou plusieurs événements à une situation de production. Elle possède un attribut délai.
Collaborations	Le délai correspond au délai existant entre le moment où une situation est susceptible de générer un événement et le moment où cet événement se produit réellement.
Conséquences	La relation produit est entièrement instanciée grâce à l'affectation d'une valeur à l'attribut délai.
Utilisations	L'attribut délai peut prendre différentes valeurs parmi lesquelles : – immédiat : l'événement est créé immédiatement ; par exemple l'événement *PrescriptionInterdite* correspondant à la prescription par un médecin, d'aspirine à une femme enceinte, doit être créé immédiatement pour que le médecin puisse changer le traitement ; – au bout d'un temps t : l'événement à prendre en compte ne se produit qu'au bout d'un temps t après la situation de production, par exemple *les résultats d'une analyse sont prêts* trois heures après une prise de sang.

TAB. 5.5 – *Le patron* RelationProduit

Rubrique	Signification
Nom	RelationConsomméPar.
Classification	Juridiction composée, caractérisation comportementale.
Intention	Affecter des valeurs aux attributs de la relation consomméPar.
Motivation	Instancier la relation consomméPar introduite dans le patron Production-Evénement-Consommation.
Applicabilité	La relation consomméPar d'une situation comportementale.
Participants	La relation consomméPar associe une ou plusieurs situations de consommation à un événement. Elle possède deux attributs délai et périodeValidité.
Collaborations	L'attribut délai correspond au délai entre le moment où l'événement apparaît et le moment où il est consommé. L'attribut périodeValidité est nécessaire dans le cas où un même événement ne doit être consommé que pendant une certaine période.
Conséquences	La relation consomméPar est entièrement instanciée grâce à la définition des valeurs des attributs délai et périodeValidité.
Utilisations	L'attribut délai peut prendre différentes valeurs parmi lesquelles : – immédiat : on consomme l'événement immédiatement, par exemple, *en cas d'erreur, le médecin doit immédiatement changer le traitement qu'il a prescrit à une femme enceinte* ; – au bout de n fois : on ne consomme l'événement que s'il s'est déjà produit n-1 fois, par exemple, *si un patient a de la fièvre pendant 15 jours consécutifs, il faut faire des examens spécifiques* ; – plus tard : l'événement sera consommé, mais peu importe quand, par exemple, *après une analyse de sang, la facture doit être envoyée au patient* ; – avant : la consommation doit se faire avant la production réelle de l'événement, mais peu importe quand, par exemple *avant qu'un chirurgien n'opère un malade, il faut contrôler que celui-ci n'est pas allergique aux produits anesthésiants* ; – un temps t avant : la consommation doit se faire un temps t avant la production réelle de l'événement, par exemple *10 minutes avant qu'un chirurgien n'opère un malade, il faut endormir le patient* ; – un temps t après : l'événement n'est consommé qu'au bout d'un temps t après qu'il ait été produit ; par exemple, *deux heures après le réveil d'un patient, contrôler sa tension* ; – toutes les x unités : l'événement sera consommé de façon répétitive, par exemple toutes les 2 heures ou tous les jours ; par exemple, *toutes les deux heures après le réveil d'un patient, contrôler sa tension.*

	L'attribut périodeValidité peut prendre différentes valeurs : – toujours : la consommation de l'événement n'est pas conditionnée ; l'événement est toujours consommé par la situation de consommation ; par exemple dans le cas de *prescription d'aspirine à une femme enceinte* ; – parfois : la consommation de l'événement ne se fait pas toujours, mais uniquement dans certains cas qui ne sont pas nécessairement précisés par rapport à la notion de temps, par exemple *après l'accord d'un rendez-vous, alerter le brancardage si nécessaire* ; – pendant x unités : l'événement doit être consommé uniquement pendant une période de temps déterminée ; par exemple, *pendant 24 heures à partir du réveil du patient, contrôler sa tension* ; – de instant1 à instant2 : l'événement doit être consommé uniquement entre deux instants précis ; par exemple, *du 1er au 31 août, les demandes de rendez-vous doivent être traitées directement par le secrétariat médico-technique*. Si instant1 est vide, l'événement doit être consommé jusqu'à instant2 ; si instant2 est vide, l'événement doit être consommé à partir de instant1 ; par exemple, *à partir de 20h00, transférer les appels au service des urgences*. La combinaison de plusieurs de ces valeurs est bien entendue possible, excepté dans le cas de la valeur toujours.

TAB. 5.6 – *Le patron* RelationConsomméPar

Notons pour conclure que l'affectation d'une valeur au bout de n fois ou toutes les x unités pour l'attribut Délai n'a de sens que si l'événement est mémorisé. En effet, par exemple, un événement mémorisé tel que *la découverte de diabète chez un patient* peut être consommé par exemple tous les deux mois (*pour contrôle de la tension*) ou au bout de 15 fois (*si le patient a du diabète pendant 15 jours consécutifs, il faut lui prescrire un traitement plus fort*). Or, un événement non mémorisé tel que *la prescription d'aspirine à une patiente enceinte* ne pourra pas être consommé au bout de 15 fois ou toutes les 2 heures. Il n'en est pas de même pour les attributs immédiat, plus tard, avant et au bout d'un temps t : on peut en effet supposer qu'un événement non mémorisé peut ne pas être consommé immédiatement, mais par exemple au bout de 2 heures et être ensuite perdu.

Etape	But de l'étape	Patron utilisé
1.	Déterminer une situation comportementale d'une application L'exprimer sous la forme Si... Alors... représentant une situation à laquelle il convient de réagir.	**Situation-Réaction**
2.	Transformer cette situation comportementale en un événement central produit par une ou plusieurs situations de production et consommé par une ou plusieurs situations de consommation.	**Production-Evénement-Consommation**
3.	Détailler l'événement central (en particulier, est-il mémorisé?).	**Evénement**
4.	Détailler chacune des situations de production ayant produit l'événement (ses concepts et ses conditions).	**Action**
5.	Détailler chacune des situations de consommation ayant produit l'événement (ses concepts et ses conditions).	**Action**
6.	Déterminer les caractéristiques des relations existant entre chacune des situations de production et l'événement central.	**RelationProduit**
7.	Déterminer les caractéristiques des relations existant entre l'événement central et chacune des situations de consommation.	**RelationConsomméPar**
8.	Recommencer en 1. jusqu'à avoir traité toutes les situations comportementales du cahier des charges de l'application.	

TAB. 5.7 – *Principes d'une démarche d'utilisation du langage de patrons*

5.2.6 Principes d'une démarche d'utilisation du langage de patrons

La table 5.7 résume la démarche d'utilisation du langage de patrons que nous préconisons dans le but de représenter des situations comportementales présentes dans des applications de domaines différents.

5.2.7 Bilan

Après avoir constaté que les patrons d'analyse existants étaient peu adaptés à la représentation des situations comportementales telles que celles que nous

avons mises en évidence dans le chapitre 2.2.2, nous avons introduit un langage de patrons destiné à aider un concepteur d'applications à représenter de telles situations comportementales. Cette approche favorise la réutilisation de connaissances acquises dès le niveau de l'analyse des besoins au cours du développement d'applications réactives dans plusieurs domaines d'applications. Cet aspect réutilisation de connaissances dès le niveau analyse est le principal but des approches à base de patrons et est aisément acquis.

Afin que cette méthode réponde pleinement à nos besoins, nous avons complété le formalisme d'E. Gamma par une dimension **méta-modèle** dont le but est de clarifier les concepts complexes introduits dans certains patrons.

Enfin, une démarche est introduite par l'utilisation du langage de patrons qui impose un ordre d'utilisation des patrons. Ainsi, la conception d'une application est facilitée et même si cet aspect est encore pauvre, cette structuration des patrons à l'intérieur d'un langage de patrons est un premier pas vers une méthode pour utiliser des patrons.

5.3 Utilisation du langage SCalP

Dans cette section, le langage de patrons SCalP est expérimenté sur un extrait du cahier des charges de l'application médicale. Chacun des patrons du langage de patrons est utilisé dans le but d'aboutir par une démarche systématique à une représentation complète des situations comportementales de ce cahier des charges.

5.3.1 Cahier des charges

Nous expérimentons le langage SCalP sur l'extrait de cahier des charges suivant :

Dans le cadre de la prescription de médicaments, la prescription d'aspirine à une femme enceinte est interdite, tout comme la prescription de suppositoires à un patient avec diarrhée. Lorsqu'un médecin détecte une grossesse chez une patiente, il doit immédiatement pratiquer un traitement particulier composé en premier lieu d'un contrôle de la tension de la patiente. De plus, il doit recontrôler la tension de la patiente tous les trois mois lors d'une visite de contrôle. Le rendez-vous pour le prochain contrôle doit donc être fixé par la secrétaire à chaque visite.

5.3.2 Du cahier des charges vers des situations comportementales

A partir du cahier des charges précédent, nous pouvons mettre en évidence cinq situations comportementales.

1. *Il ne faut pas prescrire d'aspirine à une femme enceinte.*
2. *Il ne faut pas prescrire de suppositoires à un patient avec diarrhée.*
3. *Un traitement spécifique doit être pratiqué sur une femme enceinte, en particulier un contrôle de la tension.*
4. *La tension d'une femme enceinte doit être contrôlée tous les trois mois.*
5. *Un RDV doit être accordé pour la prochaine visite.*

5.3.3 Utilisation du patron Situation-Réaction

La première étape d'utilisation du langage SCalP consiste à utiliser le patron Situation-Réaction pour exprimer les situations comportementales précédentes. Nous obtenons ainsi les situations-réactions données ci-dessous.

1. *Si le médecin prescrit de l'aspirine à une femme enceinte, alors il doit changer le traitement.*
2. *Si le médecin prescrit des suppositoires à un patient atteint de diarrhée, alors il doit changer le traitement.*
3. *Si une patiente est enceinte, alors le médecin doit lui faire passer un examen spécifique, en contrôlant en particulier sa tension.*
4. *Si une patiente est enceinte, alors le médecin doit contrôler sa tension tous les trois mois.*
5. *Si une patiente est enceinte, alors la secrétaire doit fixer un RDV pour sa prochaine visite.*

5.3.4 Utilisation du patron Production-Evénement-Consommation

Le patron Production-Evénement-Consommation est utilisé pour transformer l'ensemble de Situations-Réactions de la section 5.3.3 en un ensemble d'événements produits et consommés (cf. figure 5.4).

L'événement *Prescription Interdite* peut être produit par l'une ou l'autre de deux situations de production : la situation de production *Prescription Aspirine Femme Enceinte* correspondant à la prescription d'aspirine à une patiente enceinte et la situation de production *Prescription Suppositoires Patient avec Diarrhée* correspondant à la prescription de suppositoires à un patient atteint de diarrhée. Ce cas illustre le problème de la synchronisation de situations de production pour la production d'un même événement : ici, c'est soit la prescription d'aspirine à une patiente enceinte, soit la prescription de suppositoires à un patient atteint de diarrhée, qui provoque la création d'un événement *Prescription Interdite*. Cet événement est consommé par la situation de consommation *Changement Traitement* en changeant le traitement.

La situation d'*Analyse de Résultats* produit l'événement *Détection Grossesse* dans le cas où le médecin détecte une grossesse en analysant les résultats des analyses d'une patiente. Cet événement est consommé de deux manières différentes : d'une part, le médecin doit contrôler la tension de la patiente (situation de consommation *Contrôle Tension*) ; d'autre part, la secrétaire doit fixer le prochain RDV (situation de consommation *Accord RDV*). Cette double consommation met en évidence la portée d'un événement : le même événement est consommé par deux situations de consommation.

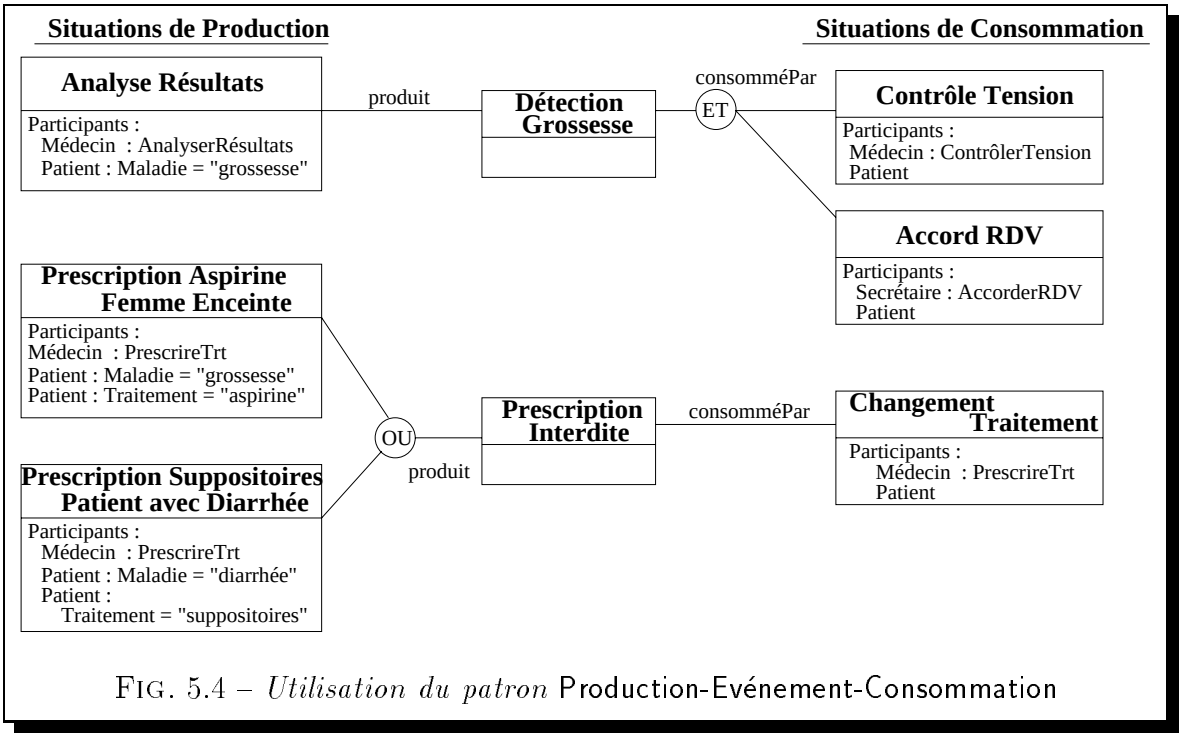


FIG. 5.4 – *Utilisation du patron* Production-Evénement-Consommation

5.3.5 Utilisation du patron Evénement

L'étape suivante consiste à utiliser le patron Evénement pour préciser les événements *Prescription Interdite* et *Détection Grossesse* (cf. figure 5.5). Il n'est pas nécessaire de mémoriser l'événement *PrescriptionInterdite* car celui-ci est ponctuel et toute prescription interdite doit être corrigée par le médecin en changeant le traitement . Par contre, l'événement *DétectionGrossesse* doit être mémorisé car son importance et son caractère durable impliquent des réactions non seulement immédiates, mais aussi tout au long des neuf mois de la grossesse (une grossesse nécessite en particulier des visites de contrôle tous les trois mois ainsi que des précautions spécifiques à prendre pour la prescription de médicaments pendant neuf mois).

FIG. 5.5 – *Utilisation du patron* Evénement

5.3.6 Utilisation du patron Action

Une fois les événements précisés, le patron Action est utilisé pour détailler la façon dont ces derniers sont produits et consommés. Une action peut être soit une situation de production, soit une situation de consommation et est décomposée en trois parties : le contexte, les concepts et les conditions.

Dans notre exemple, les actions à décomposer sont les situations de production *AnalyseRésultats*, *Prescription Aspirine Femme Enceinte* et *Prescription Suppositoires Patient avec Diarrhée* ainsi que les situations de consommation *Contrôle Tension*, *Accord RDV* et *Changement Traitement*. Le résultat de l'utilisation du patron Action pour ces situations est donné dans la figure 5.6 :

– dans la situation de production *AnalyseRésultats*, le producteur est un médecin qui examine une patiente et diagnostique la grossesse de celle-ci[1] ;

1. Il est évident que l'état enceinte n'est pas une maladie : c'est uniquement pour simplifier la lecture des diagrammes que nous nous sommes limités à introduire la classe Maladie pour regrouper divers états des patients.

- dans la situation de production *Prescription Aspirine Femme Enceinte*, le producteur est un médecin qui prescrit un traitement composé d'aspirine à une femme enceinte ;
- dans la situation de production *Prescription Suppositoires Patient avec Diarrhée*, le producteur est un médecin qui prescrit un traitement composé de suppositoires à un patient atteint de diarrhée ;
- dans la situation de consommation *Contrôle Tension*, le consommateur est un médecin qui prend la tension du patient ;
- dans la situation de consommation *Accord RDV*, le consommateur est une secrétaire qui fixe un RDV avec le patient ;
- enfin, dans la situation de consommation *Changement Traitement*, le consommateur est un médecin qui prescrit un nouveau traitement au patient.

5.3.7 Utilisation des patrons RelationProduit et RelationConsomméPar

La dernière étape pour représenter complètement les situations comportementales du cahier des charges de l'application consiste à instancier les relations produit et consomméPar à l'aide des patrons RelationProduit et RelationConsomméPar.

Le patron RelationProduit permet de préciser les relations produit en affectant une valeur à leur attribut délai parmi les deux valeurs immédiat et au bout d'un temps t. Dans notre exemple, nous choisissons de préciser les relations produit de la façon suivante :

- l'événement *Détection Grossesse* est produit immédiatement après la détection d'une grossesse lors de l'analyse des résultats d'un examen sur une patiente : l'attribut délai de la relation produit entre la situation de production *Analyse Résultats* et l'événement *Détection Grossesse* prend la valeur immédiat ;
- de même, l'événement *Prescription Interdite* est produit immédiatement après la prescription d'aspirine à une patiente enceinte : l'attribut délai de la relation produit entre la situation de production *Prescription Aspirine Femme Enceinte* et l'événement *Prescription Interdite* a donc comme valeur immédiat, de même que l'attribut délai de la relation produit entre la

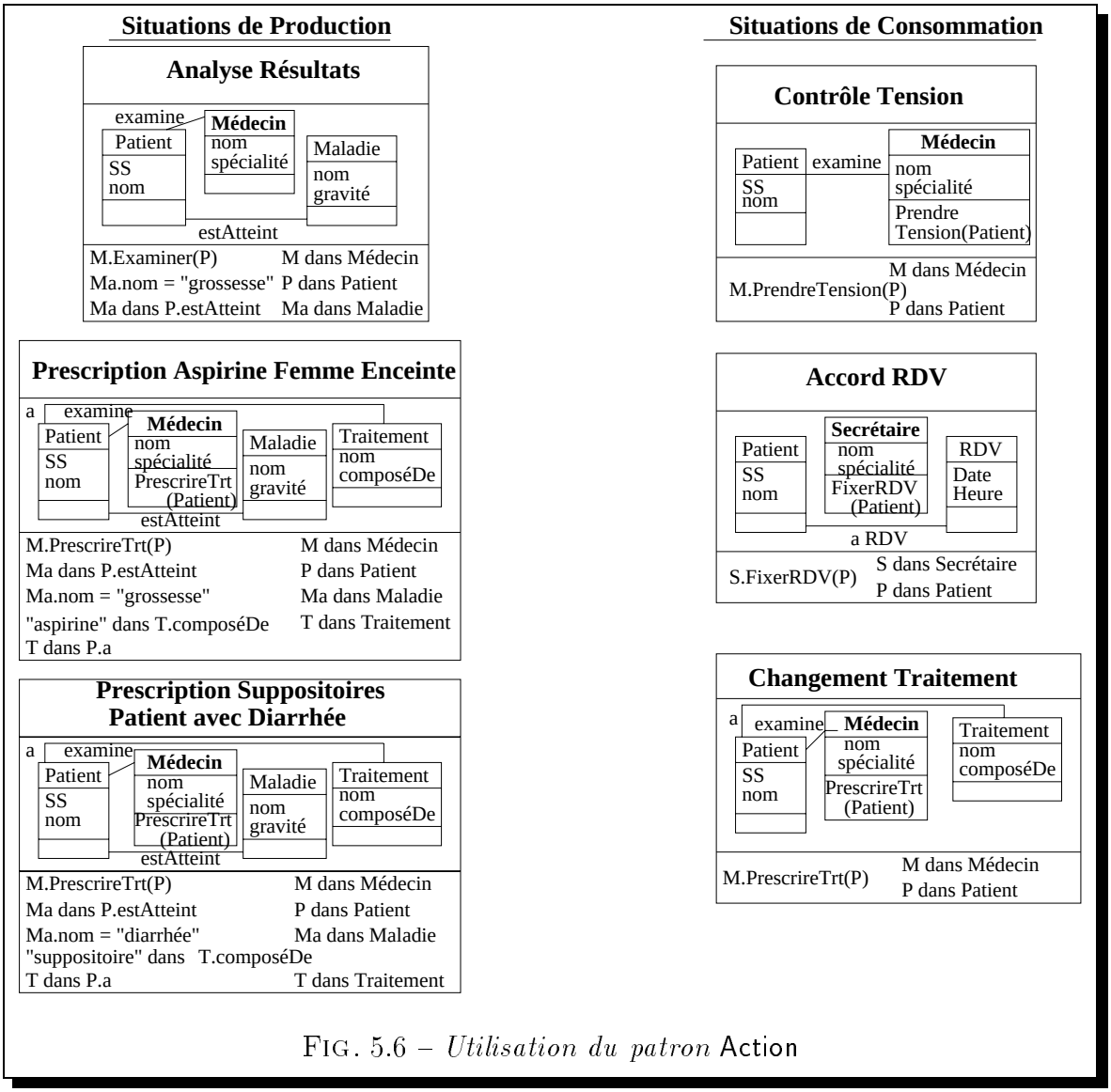


FIG. 5.6 – *Utilisation du patron* Action

situation de production *Prescription Suppositoire Patient Avec Diarrhée* et l'événement *Prescription Interdite*.

Le patron RelationConsomméPar permet de préciser les relations consomméPar en affectant une valeur à leurs attributs délai et périodeValidité. D'après le cahier des charges de l'application et les situations comportementales qui en ressortent, l'événement *Détection Grossesse* est consommé de trois façons différentes :

– d'une part, après la détection d'une grossesse chez une patiente, le médecin doit contrôler immédiatement la tension de la patiente. L'attribut délai de la relation consomméPar entre l'événement *Détection Grossesse* et la situation de consommation *Contrôle Tension* est donc immédiat. La période de

validité de cette consommation est toujours ;

- d'autre part, le médecin doit contrôler régulièrement (tous les 3 mois) la tension de la patiente. Il existe donc une deuxième relation consomméPar entre l'événement *Détection Grossesse* et la situation de consommation *Contrôle Tension*, l'attribut délai de cette relation étant instancié à tous les 3 mois. La période de validité de cette consommation est la période de la grossesse, c'est-à-dire pendant 9 mois ;

- enfin, une secrétaire doit fixer un RDV à la patiente pour sa prochaine visite. L'important ici est qu'un RDV soit accordé à la patiente pour sa prochaine visite, mais cet accord peut être conclu soit à la fin de la visite, soit quelques jours plus tard. L'attribut délai de la relation consomméPar entre l'événement *Détection Grossesse* et la situation de consommation *AccordRDV* est donc plus tard. La période de validité de cette consommation est la période de la grossesse, c'est-à-dire pendant 9 mois.

De même, l'événement *Prescription Interdite* doit être consommé immédiatement par le médecin en changeant le traitement prescrit : l'attribut délai de la relation consomméPar entre l'événement *Prescription Interdite* et la situation de consommation *Changement Traitement* a comme valeur immédiat. La période de validité de cet événement est indéfinie, donc la valeur de l'attribut périodeValidité de la relation consomméPar est toujours.

5.3.8 Bilan

Le résultat de l'utilisation du langage de patrons SCalP pour décrire les situations comportementales du cahier des charges est donné dans la figure 5.7.

Notons que les classes *Médecin*, *Patient*, *Secrétaire*, *Maladie*, *Traitement* et *RDV* correspondent aux objets du domaine d'application alors que les situations de production (*Analyse Résultats*, *Prescription Aspirine Femme Enceinte* et *Prescription Suppositoires Patient avec Diarrhée*) et les situations de consommation (*Contrôle Tension*, *Accord RDV* et *Changement Traitement*) pourraient être comparées à des collaborations génériques en UML [Mul97].

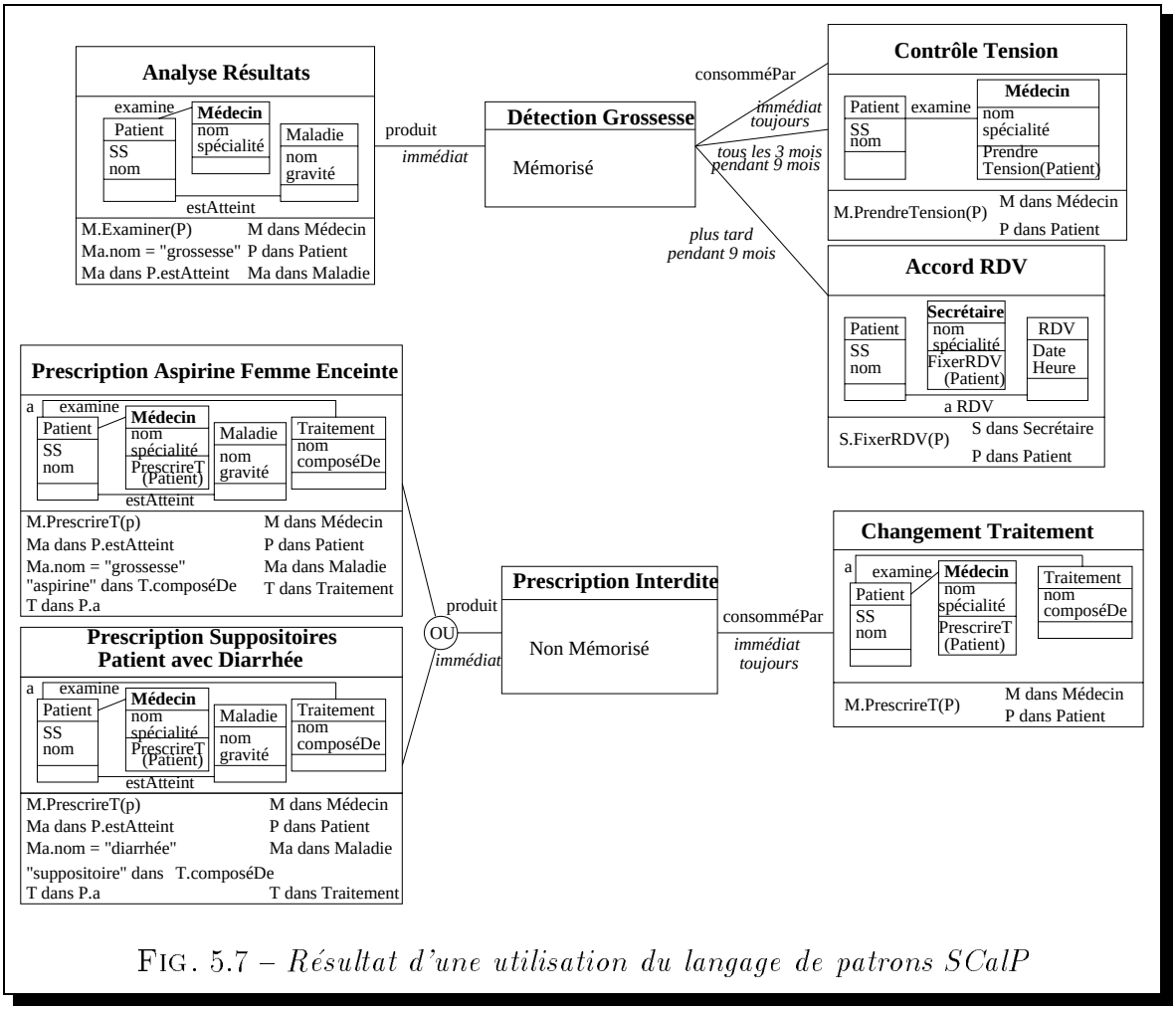


FIG. 5.7 – *Résultat d'une utilisation du langage de patrons SCalP*

5.4 Conclusion : vers une expérimentation...

Nous avons proposé un langage de patrons pour la représentation de situations comportementales communes à plusieurs domaines d'application. Ce langage de patrons se situe au niveau de l'analyse et de la représentation des besoins comportementaux d'une application et favorise la réutilisation de connaissances acquises lors de l'analyse des besoins d'une application pour le développement d'une application dans un autre domaine. Son utilisation pour l'analyse et la représentation de situations comportementales extraites du cahier des charges d'une application médicale nous permet de montrer la faisabilité de notre approche du point de vue de la recherche et de la représentation des besoins d'une application.

Le chapitre suivant montre la faisabilité de notre approche du point de vue du couplage des situations comportementales vers un système cible.

Chapitre 6

Expérimentation dans un cadre de bases de données actives

DANS CETTE PHASE CONCRÈTE D'EXPÉRIMENTATION, nous choisissons de coupler les situations comportementales mises en évidence avec le langage SCalP vers le système de gestion de bases de données (SGBD) actif NAOS. Notre choix du système NAOS comme système cible se justifie par le fait d'une part que NAOS est intégré au SGBD O_2 lui-même conforme à la norme de l'ODMG [Cat94], d'autre part qu'il regroupe des caractéristiques communes à la plupart des autres systèmes de gestion de bases de données actifs. Enfin, NAOS est en grande partie développé dans notre équipe de recherche STORM [STO97].

6.1 Utilisation de SCalP pour la conception d'applications de bases de données actives

Un système de gestion de bases de données actif utilise la notion de règle active pour implanter le comportement réactif des applications [CHR96] [Col96]. C'est le but du système NAOS [Col97] [CC96] présenté dans l'annexe B.

L'implantation d'applications dans un système cible nécessite de prendre en compte des aspects systèmes qui n'apparaissent pas lors des phases d'analyse et de conception. De ce fait, l'implantation avec NAOS engendre des problèmes spécifiques.

6.1.1 Multiples interprétations d'une situation comportementale

L'étude des applications du monde réel présentée dans la section 2.1 montre que les besoins comportementaux des applications réactives peuvent être classés selon cinq catégories : structure, évolution, activité, contrôle et exception. Avec le patron Situation-Réaction, le langage SCalP propose la généralisation de ces types de situations comportementales sous la même forme Si... Alors... afin de permettre l'expression de besoins parfois ambigus. Ainsi, toutes les situations comportementales d'une application représentent des réactions à des situations (habituelles ou inhabituelles).

La situation comportementale une patiente enceinte ne doit pas avoir un traitement composé d'aspirine s'exprime avec le patron Situation-Réaction de la façon suivante : si une patiente est enceinte, alors elle ne doit pas avoir de traitement composé d'aspirine. L'événement central est une incompatibilité dûe au fait qu'une patiente enceinte a un traitement composé d'aspirine (la situation de production). La situation de consommation concerne un changement du traitement de la part du médecin soignant le patient. D'un point de vue conceptuel, cette interprétation englobe tous les cas pouvant conduire à une incompatibilité entre un traitement composé d'aspirine et une grossesse. Si d'un point de vue conceptuel cette généralisation est nécessaire, elle n'est pas satisfaisante au niveau implantation. En effet, dès qu'on souhaite traduire cette interprétation en NAOS, il est nécessaire de savoir plus exactement comment cet événement est atteint. Or, dans NAOS comme dans la majorité des autres systèmes de gestion de bases de données actifs, les événements pris en compte sont en général de bas niveau (création d'un objet dans une classe, modification d'un attribut d'un objet d'une classe, annulation ou validation d'une transaction, début ou fin de l'exécution d'un programme ou d'une méthode, etc.). A partir d'un tel niveau d'abstraction, la situation comportementale ci-dessus peut conduire à diverses interprétations représentées grâce au langage SCalP dans la figure 6.1.

- La première interprétation est celle qui a été faite au niveau conceptuel : si une patiente est enceinte, alors elle ne doit pas avoir un traitement composé d'aspirine. Au niveau implantation, et si l'on suppose que la base de données était dans un état cohérent jusqu'alors, cette interprétation peut être violée soit par l'insertion d'un médicament de type Aspirine dans les traitements d'une patiente alors que celle-ci est enceinte, soit par l'insertion d'une nouvelle maladie dont le nom est Grossesse dans l'ensemble des maladies d'une patiente alors que celle-ci a déjà un traitement composé d'aspirine. L'événement est alors

l'**incompatibilité Traitement-Maladie** et correspond à une règle de **structure** (cf. section 2.2.2).

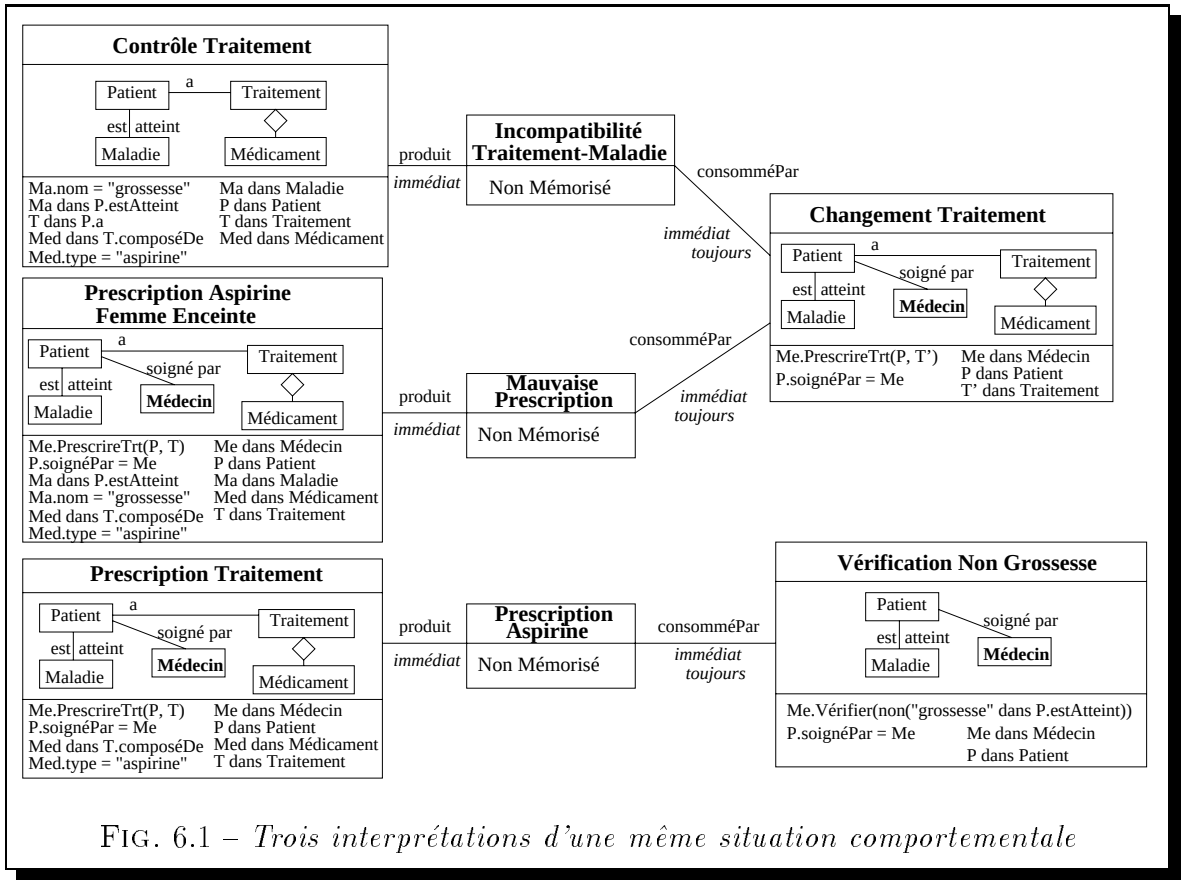


FIG. 6.1 – *Trois interprétations d'une même situation comportementale*

– La deuxième interprétation est la suivante : si un médecin prescrit de l'aspirine à une patiente enceinte, alors il doit changer ce médicament. Dans ce cas, la situation anormale est produite par un médecin lorsque celui-ci prescrit de l'aspirine à une patiente enceinte (appel de la méthode PrescrireTraitement d'un médecin sur une patiente enceinte). L'événement est alors une **mauvaise prescription** et doit être consommé par le médecin en changeant le médicament. Cet événement correspond à une règle d'**exception** (cf. section 2.2.2).

– Enfin, la dernière interprétation possible est la suivante : si un médecin prescrit de l'aspirine à une patiente, alors il doit contrôler que la patiente n'est pas enceinte. L'événement correspondant est alors **la prescription d'aspirine** à une patiente et est produit par le médecin lorsque celui-ci prescrit de l'aspirine à une patiente (appel de la méthode PrescrireTraitement d'un médecin sur une patiente). Cet événement doit être consommé par le médecin qui, selon une règle de **contrôle** (cf. section 2.2.2), vérifie que la patiente n'est pas enceinte.

Cet exemple montre qu'une seule situation comportementale au niveau conceptuel peut correspondre à plusieurs interprétations possibles au niveau implantable. Les événements conceptuels, d'un haut niveau d'abstraction, correspondent à différents types d'événements de plus bas niveau tels ceux pris en compte dans les systèmes de gestion de bases de données actifs.

Dès lors, si la généralisation des différents types de situations (structure, évolution, activité, contrôle et exception) en un type commun Situation-Réaction est nécessaire au niveau conceptuel, elle pose au niveau implantation des redondances. Ainsi, les trois interprétations de la situation comportementale une patiente enceinte ne doit pas avoir de traitement composé d'aspirine visent le même résultat : éviter le fait qu'un objet de la classe Patient de la base de données ait en même temps une maladie dont le nom est Grossesse et un traitement composé d'un médicament dont le type est Aspirine. Si l'on suppose l'application correctement implantée et la base de données cohérente, ces trois interprétations sont redondantes : si la deuxième ou la troisième interprétation est correctement traduite, la première ne se produira jamais ; enfin, c'est au demandeur de l'application de choisir si le contrôle de compatibilité entre une maladie et un médicament est nécessaire avant la prescription du médicament ou non.

6.1.2 Le modèle d'exécution d'un SGBD actif

Le problème soulevé précédemment est d'autant plus important pour une implantation vers unSGBD actif que le **modèle d'exécution** d'un tel système est capital pour le comportement des règles actives définies dans l'application. En effet, le comportement d'une application implantée dans un SGBD actif dépend fortement du modèle d'exécution de celui-ci [Cou96]. Or, en adoptant les principes des approches à base de patrons, nous pouvons penser qu'à un type de situation comportementale correspond un modèle d'exécution privilégié dans NAOS, essentiellement au niveau du mode de couplage et de l'instant de production de l'événement. Cette approche est aussi considérée par l'ACT-NET Consortium [Con96].

La traduction des situations comportementales en NAOS nécessite alors la gestion du "typage" des situations comportementales : structure, évolution, activité, contrôle et exception. Ce bref retour en arrière est nécessaire dans un but de pertinence de l'application implantée, afin qu'un besoin ne soit pas codé de façon redondante. Pour cela, l'approche SCalP spécialise le patron Situation-Réaction afin que le demandeur de l'application décide quel est plus précisément la nature de chaque situation comportementale.

Rubrique	Signification
Nom	Structure.
Hérite de	Patron Situation-Réaction.
Applicabilité	Les situations comportementales ayant comme but la gestion de la structure des objets.
Utilisations	Si la température d'un patient est modifiée, alors elle ne doit pas être inférieure à 36 degrés ni supérieure à 40 degrés. Si une patiente est enceinte, alors elle ne doit pas avoir de traitement composé d'aspirine.

TAB. 6.1 – *Le patron* Structure

6.1.3 Spécialisation du patron Situation-Réaction

Les problèmes de spécialisation, d'adaptation et de composition de patrons ne sont encore que peu traités dans la littérature. Nous proposons ici une spécialisation du patron Situation-Réaction par rapport à la **nature** de la situation visée par le patron (cf. figure 6.2). Le patron Situation-Réaction est ainsi spécialisé en cinq patrons : Structure (cf. tableau 6.1), Evolution (cf. tableau 6.2), Activité (cf. tableau 6.3), Contrôle (cf. tableau 6.4) et Exception (cf. tableau 6.5). La spécialisation porte essentiellement sur la rubrique Applicabilité et est matérialisée dans la nouvelle rubrique Hérite de.

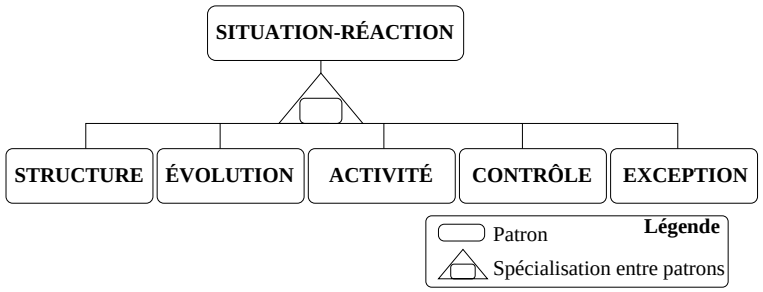


FIG. 6.2 – *Spécialisation du patron* Situation-Réaction *pour un couplage vers NAOS*

Rubrique	Signification
Nom	Evolution.
Hérite de	Patron Situation-Réaction.
Applicabilité	Les situations comportementales ayant comme but le contrôle de l'évolution des objets.
Utilisations	Si le prix d'un médicament est augmenté, alors l'augmentation maximale autorisée est de 10 % de l'ancien prix. Si la température d'un patient augmente de plus de 1 degré en 2 heures, alors avertir le médecin traitant du patient.

TAB. 6.2 – *Le patron* Evolution

Rubrique	Signification
Nom	Activité.
Hérite de	Patron Situation-Réaction.
Applicabilité	Les situations comportementales ayant comme but la gestion de l'activité de l'application.
Utilisations	Si une demande d'analyse arrive à l'UMT, alors traiter l'analyse. Si un rendez-vous est confirmé, alors un rendez-vous pour le brancardage doit être fixé.

TAB. 6.3 – *Le patron* Activité

Rubrique	Signification
Nom	Contrôle.
Hérite de	Patron Situation-Réaction.
Applicabilité	Les situations comportementales ayant comme but le contrôle du bon fonctionnement de l'application.
Utilisations	Si le médecin prescrit de l'aspirine à une patiente, alors il doit vérifier que la patiente n'est pas enceinte. Si le médecin doit opérer un patient, alors il doit vérifier que celui-ci n'est pas allergique aux produits anesthésiants.

TAB. 6.4 – *Le patron* Contrôle

Rubrique	Signification
Nom	Exception.
Hérite de	Patron Situation-Réaction.
Applicabilité	Les situations comportementales ayant comme but la réaction à des situations inhabituelles ou à des erreurs.
Utilisations	Si la température d'un patient est supérieure à 39,5 degrés, alors il faut surveiller attentivement ce patient. Si les résultats d'un acte d'un patient sont "Haute Gravité", alors il faut alerter le médecin du patient.

TAB. 6.5 – *Le patron* Exception

6.2 NAOS et les situations comportementales

Cette section présente l'expérimentation de notre approche dans le cadre de la conception de bases de données actives avec le système NAOS.

6.2.1 Traduction de situations comportementales en NAOS

NAOS étant intégré au SGBD O_2, nous présentons ici la traduction en O_2/NAOS d'un ensemble de situations comportementales appartenant à un extrait de l'application médico-technique dont le schéma O_2 est donné dans l'annexe C.

Le résultat de l'utilisation du langage SCalP sur ces situations comportementales est représenté dans les figures 6.3 à 6.11. Afin d'apporter plus de clarté à ces figures, nous ne notons dans la partie Concepts que les noms des classes concernées et les relations entre ces classes. Les attributs et les méthodes de ces classes sont représentés avec les autres classes dans la figure C.1, annexe C.

Pour chaque situation comportementale, nous rappelons l'interprétation qui en a été faite selon l'utilisation des patrons Structure, Evolution, Activité, Contrôle et Exception. En particulier, nous montrons comment un même énoncé de situation comportementale peut donner lieu à plusieurs interprétations selon l'utilisation de l'une ou de l'autre des spécialisations du patron Situation-Réaction.

Enfin chacune des situations comportementales est traduite en O_2/NAOS. Parfois, des remarques sur des restrictions d'utilisation de NAOS et sur les bénéfices potentiels de l'apport de nouvelles fonctionnalités sont données. On notera en particulier comment deux interprétations possibles d'une même situation comportementale donnent lieu à des traductions différentes en NAOS.

Le lecteur souhaitant des explications sur la signification d'une règle active écrite en NAOS pourra se reporter à l'annexe B.

1. Evénement *Incompatibilité Traitement-Maladie* (cf. figure 6.3)

   Cet événement correspond à la situation comportementale Une patiente enceinte ne doit pas avoir de traitement composé d'aspirine qui peut avoir plusieurs interprétations. Nous analysons deux de ces interprétations afin de montrer d'une part combien le résultat obtenu pour l'expression sous la forme Si... Alors... dépend du patron utilisé, d'autre part combien l'implantation en NAOS diverge.

   L'événement Incompatibilité Traitement-Maladie est le résultat de l'utilisation du patron Structure sur cette situation comportementale.

   Si une patiente est enceinte
   alors elle ne doit pas avoir de traitement composé d'aspirine.

   Cette situation comportementale exprime une réaction à engendrer suite à une situation d'incompatibilité entre un traitement et une maladie. Cette situation d'incompatibilité est générée de deux manières différentes : soit une maladie de type "Grossesse" est insérée dans l'ensemble des maladies d'une patiente alors que celle-ci possède un traitement composé d'aspirine, soit un médicament de type "Aspirine" est inséré dans les traitements d'une patiente alors que celle-ci est enceinte. Cette constatation nous amène tout naturellement à traduire cette situation-réaction par une règle active déclenchée sur un événement composite (insertion d'une nouvelle maladie dans la collection des maladies d'un patient ou insertion d'un nouveau médicament dans le traitement d'un patient). Les deux cas n'ont cependant pas la même importance du point de vue de la réaction à engendrer : si annuler l'insertion d'une maladie de type "Grossesse" n'a aucun sens, l'annulation de la prescription du médicament de type "Aspirine" est inévitable dans le second cas.

   Afin d'apporter plus de **sémantique** et plus de clarté aux règles actives définies, nous préferons donc utiliser deux règles actives déclenchées sur des événements indépendants plutôt qu'une règle active déclenchée sur un événement composite. Les deux règles actives InsertionGrossesse et PrescriptionAspirine sont ainsi

définies. La règle InsertionGrossesse est déclenchée après l'insertion d'une nouvelle maladie dans la collection des maladies d'un patient (clause on after insert Patient→MaladiesContractées), vérifie que l'une des maladies insérées est une grossesse alors que la patiente possède un traitement composé d'aspirine (clause if) et supprime le médicament de type "Aspirine" de la liste des médicaments de la patiente (clause do). La règle active PrescriptionAspirine réagit avant l'insertion d'un nouveau médicament dans le traitement d'une patiente (clause on before insert Patient→TraitementSuivi), vérifie que l'un des médicaments insérés est de type "Aspirine" alors que la patiente est enceinte (clause if) et affiche un message pour avertir l'utilisateur et remplacer la prescription du médicament de type "Aspirine" (clause do instead).

Règles NAOS :

```
rule InsertionGrossesse
coupling immediate
on after insert Patient→MaladiesContractées with P
if ((exists m in delta(P) : m→Nom = "Grossesse")
  and (range of lm is o2 set(Medicament)
                select M
                from M in P→TraitementSuivi
                where M→Type = "Aspirine"))
do { display ("Il faut supprimer tous les médicaments composés d'aspirine
            dans le traitement de cette patiente");
   P→TraitementSuivi -= lm ; }

rule PrescriptionAspirine
coupling immediate
on before insert Patient→TraitementSuivi with P
if ((exists m in current(P)→MaladiesContractées : m→Nom = "Grossesse")
  and (exists me in delta(P) : me→Type = "Aspirine"))
do instead { display ("Attention : vous avez prescrit de l'aspirine
                    alors que cette femme est enceinte"); }
```

Cette recherche d'une meilleure sémantique met en exergue l'utilité des masques d'événements proposés par des systèmes comme Samos [GGD91], Snoop [CAM93] ou encore Ode [GJ91] [GJS92]. Utiliser des masques dans une règle active permet de donner plus de sémantique aux événements déclencheurs de la règle et

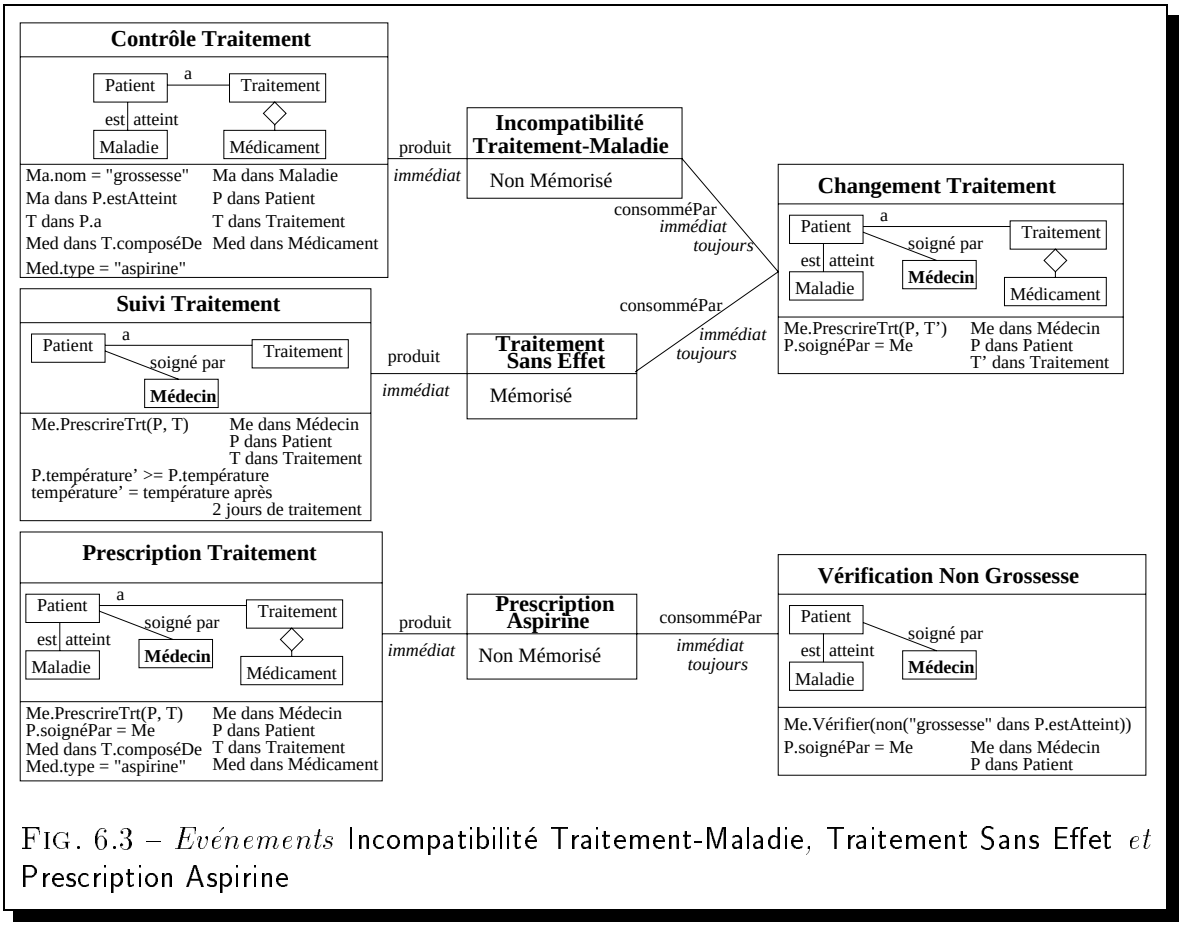


FIG. 6.3 – *Evénements* Incompatibilité Traitement-Maladie, Traitement Sans Effet *et* Prescription Aspirine

par conséquent à la règle elle-même. Ainsi, en intégrant cette notion de masque à NAOS, la règle active InsertionGrossesse serait déclenchée non pas après l'insertion de n'importe quelle maladie dans l'ensemble des maladies de n'importe quel patient, mais après l'insertion d'*une maladie de type "Grossesse"* chez *une patiente*. D'un point de vue utilisateur, les masques améliorent la sémantique en permettant de conditionner le déclenchement de la règle à un événement plus proche du niveau utilisateur. Notons cependant que l'exécution effective de la règle ne dépend pas de l'utilisation des masques, celle-ci étant liée, dans un système n'intégrant pas cette notion, à l'évaluation de la partie Condition des règles.

Malgré son utilité et parce que NAOS ne l'intègre pas, nous utilisons cette notion de masques uniquement lorsqu'elle introduit un aspect autre que l'apport de sémantique.

2. Evénement *Prescription Aspirine* (cf. figure 6.3)

   Cet événement correspond à l'application du patron Contrôle à la situation comportementale *Une patiente enceinte ne doit pas avoir de traitement composé*

d'aspirine.

Cette règle de contrôle correspond à la règle de structure ci-dessus, mais son but n'est pas le même : ici, on souhaite contrôler qu'une patiente à laquelle un médecin prescrit un traitement composé d'aspirine n'est pas enceinte, autrement dit : *avant de prescrire un traitement composé d'aspirine à une patiente, il faut vérifier que la patiente n'est pas enceinte* :

Si le médecin prescrit de l'aspirine à une patiente
alors il doit vérifier que la patiente n'est pas enceinte.

Nous traduisons tout naturellement cette règle de contrôle par une règle active immédiate qui réagit avant l'insertion d'un médicament dans le traitement d'une patiente (clause on before insert Patient→TraitementSuivi) et vérifie que ce médicament est de type "Aspirine" (clause if). Si tel est le cas, c'est alors l'action de la règle (clause do) qui recherche si l'une des maladies de la patiente à laquelle l'aspirine est prescrit est une grossesse ; le cas échéant, la règle affiche un message et supprime l'insertion de ce médicament du traitement de la patiente.

Règle NAOS :

```
rule ControleTraitement
coupling immediate
on before insert Patient→TraitementSuivi with P
if (exists Me in delta(P) : Me →Type = "Aspirine")
do { o2 list(Maladie) lm;
     o2query (lm, "select M
                   from M in $1
                   where M→Nom = "Grossesse"", P→MaladiesContractees) ;
     if (lm <> list()) {
       Display ("Attention : la patiente est enceinte") ;
       delta(P)→TraitementSuivi -= Me ; } ; }
```

3. Evénement *Température Non Conforme* (cf. figure 6.4)

Cet événement correspond à l'utilisation du patron Structure sur la situation comportementale *La température d'un patient doit être comprise entre 36 et 40 degrés.*

Si la température d'un patient est modifiée
alors elle ne doit pas être inférieure à 36 degrés ni supérieure à 40 degrés.

La traduction de cette règle de structure se fait classiquement en NAOS par la règle active TemperatureNonConforme qui réagit immédiatement après la modification de l'attribut Température d'un objet de la classe Patient (clause on after update Patient→Temperature), compare la nouvelle température aux bornes requises (clause if) et avertit le médecin si la température n'est pas correcte (clause do). La transaction n'est pas annulée car il est possible que la température soit effectivement en dehors des bornes "normales".

Règle NAOS :

rule TemperatureNonConforme
coupling immediate
on after update Patient→Temperature **with** P
if (P→Temperature > 40)
  or (P→Temperature < 36)
do { display ("Attention : température non conforme aux normales ! "); }

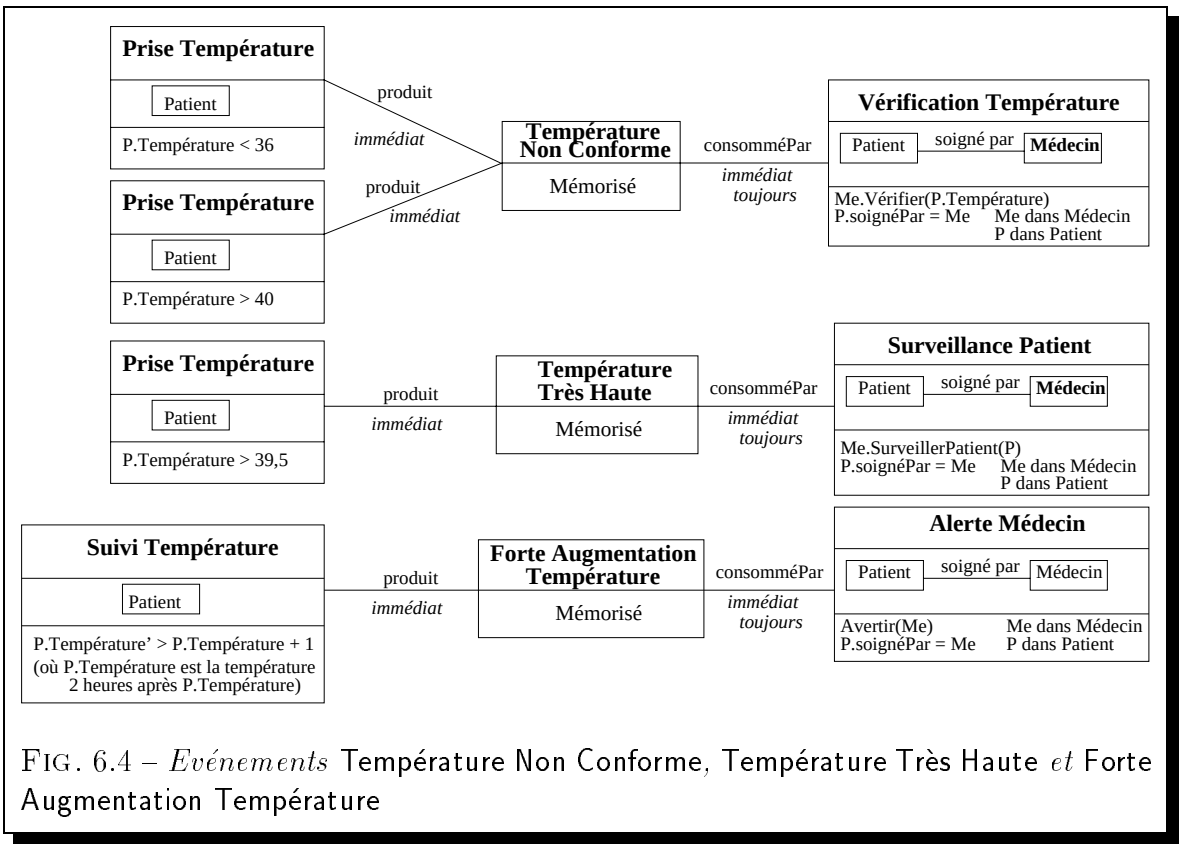


FIG. 6.4 – *Evénements* Température Non Conforme, Température Très Haute *et* Forte Augmentation Température

4. Evénement *Forte Augmentation Temperature* (cf. figure 6.4)

   Cet événement correspond à l'application du patron Evolution sur la situation comportementale *La température d'un patient ne doit pas augmenter de plus de 1 degré toutes les deux heures.*

   Si la température d'un patient augmente de plus de 1 degré en 2 heures,
   alors avertir le médecin traitant du patient.

   Cette règle d'évolution exprime un traitement qui doit être effectué de façon asynchrone par rapport aux autres traitements de l'application. Dans notre contexte, une exécution asynchrone caractérise le déclenchement d'une tâche par une autre tâche ; l'exécution de la tâche déclenchée est indépendante de la tâche déclenchante et peut se faire en parallèle ou à tout autre moment. La description d'une telle exécution n'est pas possible à l'heure actuelle dans O_2 ni dans tout autre système de gestion de bases de données où les traitements sont effectués séquentiellement et où la notion de temps est peu prise en compte.

   Par rapport à ces deux problèmes, NAOS offre la notion de règle active réagissant à des événements temporels : ici, la règle est déclenchée toutes les deux heures après la modification de la température d'un patient (clause on temporal event every 2 hour after update Patient→Temperature) et vérifie que la température n'a pas augmenté de plus de 1 degré (clause if). Notons cependant qu'aucune delta-structure n'est associée à un événement temporel dans NAOS et qu'il n'est donc pas possible de savoir quel est l'environnement d'exécution de la règle, autrement dit quel est le patient dont la température a été modifiée deux heures plus tôt.

   S'il était possible de retrouver l'environnement d'exécution de l'événement temporel relativement à celui de l'événement relatif, la règle active correspondante serait la suivante :

   Règle NAOS :

   rule ForteAugmentationTemperature
   coupling immediate
   on temporal event
   every 2 **hour**
   after update Patient→Temperature **with** P
   if (**current**(P)→Temperature > **old**(P)→Temperature + 1)
   do { display ("Attention : la température du patient a fortement augmenté ;
   prévenez son médecin !") ; }

5. Evénement *Traitement Sans Effet* (cf. figure 6.3)

   Cet événement correspond à l'application du patron Activité sur la situation comportementale *Un traitement n'ayant pas d'effet sous 2 jours doit être changé.*

   Si un traitement est prescrit à un patient
   alors deux jours après, vérifier la température du patient et si sa température est la même qu'au moment de la prescription du traitement, changer le traitement.

   De la même manière que pour la règle d'évolution présentée ci-dessus, cette règle d'activité impose un traitement asynchrone de l'application traduisible par une règle active avec toutefois les problèmes quant à la delta-structure. De plus, la température d'un patient est généralement modifiée plusieurs fois en deux jours. Il faut donc que la valeur de la température du patient au moment de la prescription du traitement soit mémorisée. Cela nécessite de connaître la valeur de l'objet à la fois au moment où l'événement relatif est déclenché et au moment où l'événement temporel se produit (deux jours après l'événement relatif). La solution que nous proposons à ce problème consiste à mettre à jour un attribut TemperaturePrescriptionTraitement au moment de la prescription du traitement et à comparer (clause if) à cette valeur la valeur de l'attribut Temperature du patient deux jours après la prescription du traitement, au moment où la règle TraitementSansEffet est effectivement déclenchée (clause on temporal event 2 day after insert Patient→TraitementSuivi).

   Règle NAOS :

   rule TraitementSansEffet
   coupling immediate
   on temporal event
   2 **day after insert** Patient→TraitementSuivi **with** P
   if (P→Temperature >= P→TemperaturePrescriptionTraitement)
   do { P→MedecinTraitant→PrescrireTraitement(P); }

   Notons enfin qu'une telle règle soulève un important problème au niveau transactionnel. En effet, elle implique que la transaction dans laquelle elle est définie soit une transaction longue (de durée supérieure à deux jours) validée uniquement après l'exécution de la règle, donc deux jours plus tard. Or dans un tel contexte médico-technique, il est évident que de nombreuses procédures devront être exécutées et validées indépendamment de cette attente de deux jours. Un mécanisme de transactions séparées est donc nécessaire pour l'implantation de l'application médico-technique.

6. Evénement *Augmentation Trop Importante* (cf. figure 6.5)

Cet événement correspond à l'utilisation du patron Evolution sur la situation comportementale *L'augmentation du prix d'un médicament ne doit pas être supérieure à 10 % du prix initial.* Cette règle d'évolution peut être interprétée de deux façons différentes :

– Si le prix d'un médicament est augmenté
alors plafonner l'augmentation à 10 % de l'ancien prix.

Cette règle d'évolution peut être traduite naturellement en une règle active immédiate qui réagit à la modification du prix d'un médicament et compare la valeur de cet attribut après la modification.

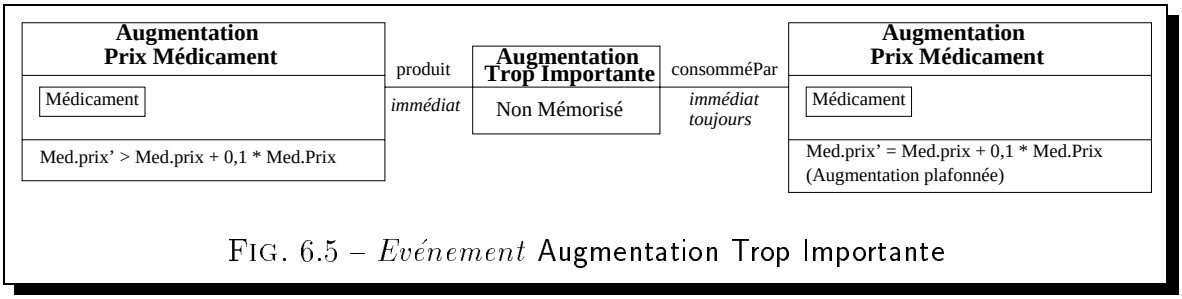


FIG. 6.5 – *Evénement* Augmentation Trop Importante

– Si le prix d'un *même* médicament est augmenté
alors plafonner l'augmentation à 10 % de l'ancien prix.

Cette règle d'évolution implique de considérer toutes les augmentations du prix d'un médicament particulier pendant une transaction. Elle ne peut donc être traduite par une règle active immédiate car s'intéresser à l'augmentation du prix d'un même médicament pendant une transaction suppose de comparer le prix en début de la transaction et le prix en fin de transaction. Il est donc nécessaire d'utiliser une règle différée qui prendra en compte l'effet net d'une séquence d'opérations[1]. La règle active différée (clause coupling) AugmentationPrixTropImportante est ainsi définie ; elle réagit à la modification de l'attribut Prix d'un médicament (clause on after update Medicament→Prix) et compare la valeur obtenue après la dernière modification de l'attribut Prix d'un médicament à sa valeur initiale en début de transaction (clause if).

1. L'effet net d'une séquence d'opérations est le bilan de ces opérations qui perdure à la fin de cette séquence (cf. annexe B).

Règle NAOS :

rule AugmentationPrixTropImportante
coupling deferred
on after update Medicament→Prix **with** Me
if (Me→Prix > **old**(Me)→Prix * 1,10)
do { Me→Prix = **old**(Me)→Prix * 1,10 ; }

Cette règle différée est finalement souhaitable car elle permet aussi de prendre en compte la première situation comportementale en réagissant en fin de transaction à *toutes* les modifications des prix de médicaments survenues pendant la transaction.

7. Evénement *Détection Diabète* (cf. figure 6.6)

Cet événement correspond à l'application du patron Activité sur la situation comportementale *Tous les jours pendant un mois, prendre le taux de glycémie d'un patient chez qui on a détecté un diabète.*

Si on détecte du diabète chez un patient
alors tous les jours pendant un mois, prendre son taux de glycémie.

Comme précédemment, cette activité asynchrone peut facilement être gérée par une règle active immédiate déclenchée toutes les 24 heures après la détection d'un diabète chez un patient, c'est-à-dire l'insertion d'une maladie de nom "Diabète" dans l'ensemble des maladies d'un patient (clause on temporal event every 24 hour after insert Patient→MaladiesContractees). Outre la gestion de la delta-structure, un nouveau problème est dû au fait que le contrôle de la glycémie ne doit se faire que pendant un mois après la détection du diabète : il faut donc faire en sorte que la règle SuiviDiagnostic ne s'effectue que pendant une période de temps déterminée qui dépend ici du moment où est détecté le diabète. Aucune telle notion de période de validité n'est associée à une règle dans NAOS, si ce n'est que la validité d'un événement rend implicite celle d'une règle active : une règle déclenchée sur un événement apparaissant du 1^{er} janvier au 28 février ne pourra être effectivement déclenchée que du 1^{er} janvier au 28 février. Il convient donc soit de proposer une notion explicite de période de validité d'une règle, soit d'étendre la validité d'un événement relativement à l'occurrence d'un autre événement (ici l'insertion d'une nouvelle maladie de nom "Diabète" traduite par OccurringDate + 1 month).

Règle NAOS :

rule SuiviDiagnostic
coupling immediate
on temporal event
every 24 **hour**
until OccurringDate + 1 **month**
after insert Patient→MaladiesContractees **with** P
if (**exists** Me in P→MaladiesContractees : Me→Nom = "Diabète")
do { display("Prendre la glycémie du patient", P→Nom) ; }

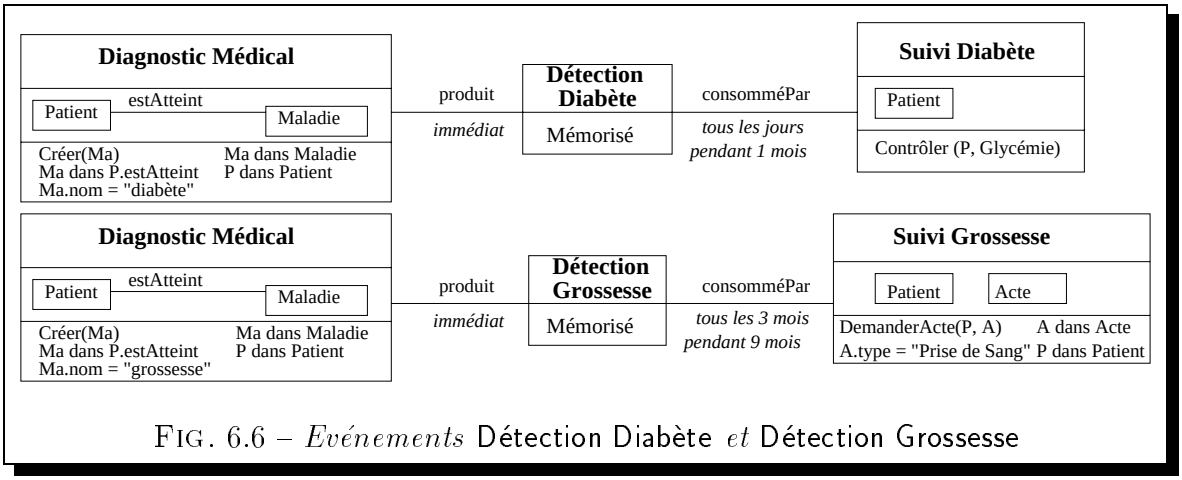


FIG. 6.6 – *Evénements* Détection Diabète *et* Détection Grossesse

8. Evénement *Détection Grossesse* (cf. figure 6.6)

Cet événement correspond à l'application du patron Activité sur la situation comportementale *Tous les trois mois pendant une grossesse, prescrire une prise de sang à la patiente.*

Si une patiente est enceinte
alors tous les trois mois pendant sa grossesse, lui prescrire une prise de sang.

Ici encore, il est nécessaire de gérer la période de validité pendant laquelle la règle doit être déclenchée. Comme précédemment, la validité d'une règle peut être obtenue implicitement grâce à l'occurrence de l'événement correspondant pendant une période donnée. Or ici, cette période correspond à la période de grossesse de la patiente sur laquelle la règle est déclenchée et n'est pas connue à priori. De récents travaux de recherche [Ron97] introduisent des types d'événements dynamiques pour répondre à ce problème : ceux-ci expriment des événements utilisant des valeurs temporelles stockées dans la base de données. Il devient alors possible

d'exprimer le fait que des événements temporels apparaissent relativement à une date stockée dans la base de données ou pendant une période propre à chaque objet d'une même classe : ici, en supposant qu'un attribut PeriodeGrossesse stocke pour chaque patiente sa période de grossesse, la règle active SuiviGrossesse sera déclenchée tous les 3 mois pendant la période de grossesse de chaque patiente enceinte (clause on temporal event every 3 month during Patient→PeriodeGrossesse).

Règle NAOS :

rule SuiviGrossesse
coupling immediate
on temporal event
every 3 **month**
during Patient→PeriodeGrossesse **with** P
do { Analyse PS;
PS→Type→Type = "Prise de sang" ;
P→MedecinTraitant→DemanderActe(P, PS) ; }

9. Evénement *Opération* (cf. figure 6.7)

Cet événement correspond d'une part à l'application du patron Contrôle sur la situation comportementale *Avant d'opérer un patient, vérifier que celui-ci n'est pas allergique aux produits anesthésiants*, d'autre part à l'application du patron Activité sur la situation comportementale *Dix minutes avant le début d'une opération, endormir le patient.*

– L'application du patron Contrôle sur la première situation comportementale donne lieu à la situation-réaction suivante :

Si le médecin doit opérer un patient
alors il doit vérifier que celui-ci n'est pas allergique aux produits anesthésiants.

Le but de cette règle est de contrôler qu'un patient n'est pas allergique aux produits anesthésiants avant de l'opérer. La règle active ci-dessous doit donc être déclenchée avant le début de la méthode EndormirPatient (clause on before method-begin Medecin→EndormirPatient).

Règle NAOS :

rule ControleTraitement
coupling immediate

on before method-begin Medecin→EndormirPatient(P) **with** M
do { VerifierAllergies(**arg**(M)→P) ; }

où **arg**(M)→P représente le patient paramètre effectif de la méthode EndormirPatient qui a déclenché la règle.

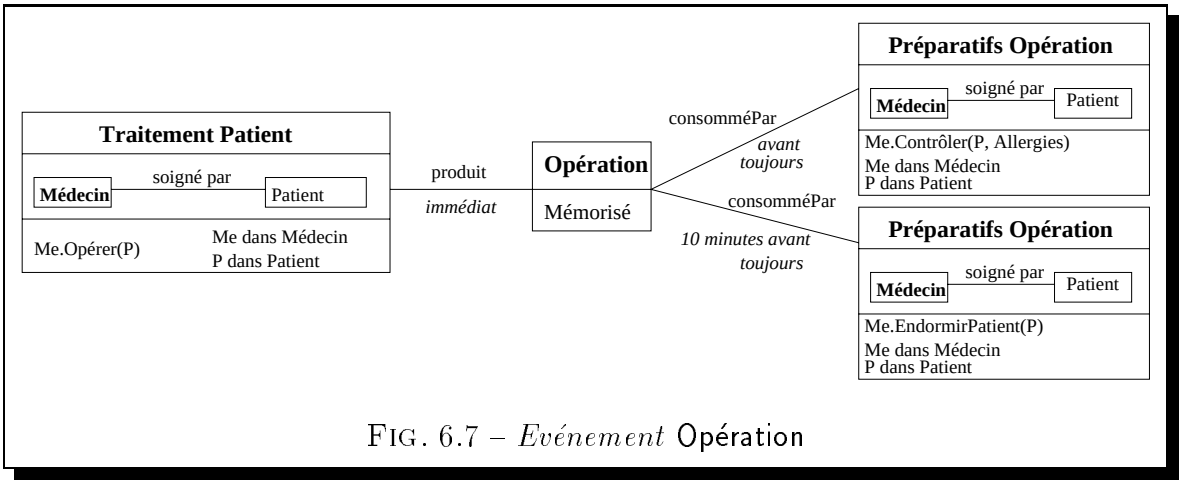


FIG. 6.7 – *Evénement* Opération

– L'application du patron Activité sur la deuxième situation comportementale donne lieu à la situation-réaction suivante :

Si une opération doit être pratiquée chez un patient
alors dix minutes avant le début de l'opération, endormir le patient.

Actuellement, cette règle qui représente une activité normale devant être effectuée avant le début de l'opération d'un patient, ne peut être implantée en O_2 : la gestion purement séquentielle des instructions ne permet pas d'exprimer le moment où le patient doit être endormi par rapport à celui où il doit être opéré. En NAOS, le fait que les règles ne soient exécutées qu'après l'occurrence d'un événement ne permet pas de fixer aisément l'exécution d'une action dix minutes avant cet événement. Le seul moyen pour implanter une telle situation comportementale consiste à déduire la date à laquelle le patient doit être endormi à partir de celle de son opération stockée dans la base, et à définir une règle active déclenchée exactement à cette date. Cette solution n'autorise néanmoins aucune modification de la date de l'opération.

Les travaux proposés dans [Ron97] permettent de traiter de telles situations où il est nécessaire de supporter des événements relatifs spécifiant un instant avant un autre. En intégrant leur notion de types d'événements dynamiques, NAOS pourrait alors gérer cette situation en déclarant la règle active suivante :

Règle NAOS :

rule OperationPatient
coupling immediate
on temporal event
at Patient→DateOperation - 10 **minute with** P
do { P→MedecinTraitant→EndormirPatient(P) ; }

10. Evénement *Absence Médecin* (cf. figure 6.8)

Cet événement correspond à l'application du patron Exception sur la situation comportementale *Si un médecin est malade, alors il faut recalculer le planning des visites de façon à ce que ses malades puissent être vus par d'autres médecins.*

Si un médecin est absent,
alors il faut répartir les visites de ses patients sur les autres médecins.

Pour traduire cette situation exceptionnelle où un médecin est absent sans que celà ait été prévu, il est nécessaire d'utiliser une règle active réagissant à l'événement utilisateur AbsenceMedecin (clause on user event AbsenceMedecin(MedecinAbsent : Medecin)). Cette règle est immédiate et recherche les patients du médecin pour les répartir sur les autres médecins.

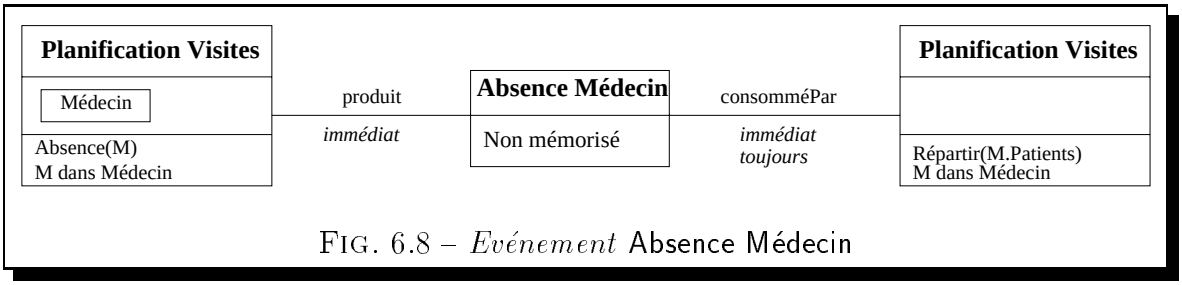


FIG. 6.8 – *Evénement* Absence Médecin

Règle NAOS :

rule AbsenceMedecin
coupling immediate
on user event AbsenceMedecin(MedecinAbsent : Medecin)
if range of lp **is o2 set**(Patient)
select P
from P **in** LesPatients
where P→MedecinTraitant = MedecinAbsent
do { display ("Répartir les patients suivants", lp) ; }

11. Evénement *Demande Acte* (cf. figure 6.9)

Cet événement correspond à l'application du patron Activité sur deux situations comportementales : *L'unité médico-technique (UMT) doit réagir à une demande d'acte en fixant un rendez-vous au patient pour un examen ou en traitant des analyses* et *Un prescripteur demande un acte à l'UMT ; l'UMT doit alors lui retourner les informations correspondantes (indications, contre-indications, etc.).* L'application de ce patron donne lieu à 3 règles d'activité :

Si une demande d'analyse arrive à l'UMT,
alors l'UMT doit traiter l'analyse.

Si une demande d'examen arrive à l'UMT,
alors l'UMT doit fixer un rendez-vous au patient pour l'examen.

Si un prescripteur demande un acte à l'UMT,
alors l'UMT doit retourner des informations au prestataire.

Ces règles d'activité correspondent à des traitements séquentiels synchrones traduisibles en O_2 de façon tout à fait naturelle dans le corps du programme TraiterDemandeActe de l'application médico-technique. Le but de ce programme est le traitement d'une demande d'acte par l'unité médico-technique.

Programme O_2 :

```
program TraiterDemandeActe(LActe: Acte)
{ Prescripteur pr;
  string p, d, r;

  ...
  if (LActe→Type→Type = "Analyse") /* L'acte est une analyse */
    TraiterAnalyses(LActe);
  else /* L'acte est un examen */
    AccorderRdv(LActe→EntiteConcernee, LActe);

  ...
  pr = LActe→Prescripteur;
  p = LActe→Type→Preparation;
  d = LActe→Type→Deroulement;
  r = LActe→Type→Remarques;
  self→EnvoyerInformations(pr, p, d, r);
... }
```

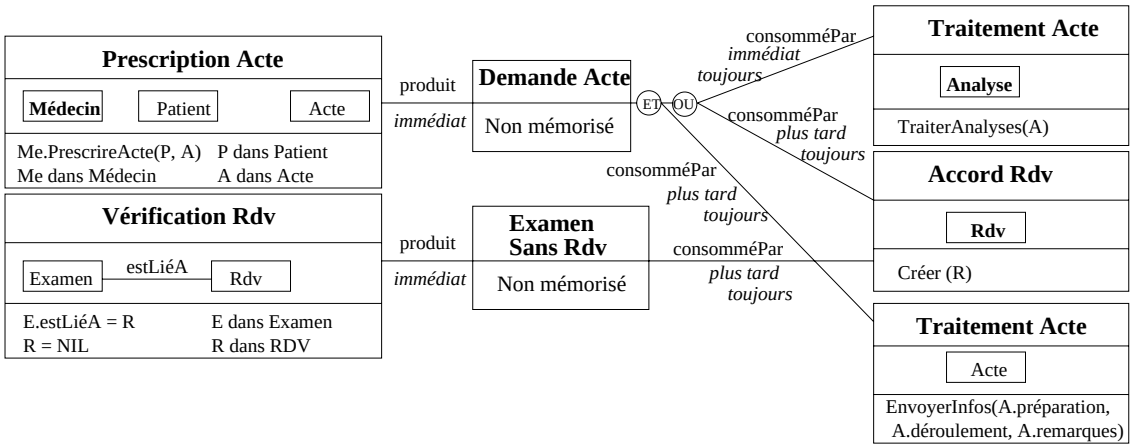


FIG. 6.9 – *Evénements* Demande Acte *et* Examen Sans Rdv

12. Evénement *Examen Sans Rdv* (cf. figure 6.9)

Cet événement correspond à l'utilisation du patron Structure sur la situation comportementale *Pour être exécuté, tout examen doit avoir un rendez-vous.*

Si un examen est sans rendez-vous
alors fixer un rendez-vous pour cet examen.

En réalité, un rendez-vous peut n'être accordé qu'un certain temps après l'enregistrement d'une demande d'examen. Le fait qu'un examen soit sans rendez-vous est donc tolérable pendant un certain temps. Par conséquent, nous définissons la règle active différée ExamenSansRdv afin de permettre l'affectation d'un rendez-vous associé à l'examen à la fin de la transaction. Dans NAOS, une règle différée a une sémantique ensembliste: elle accumule les événements déclenchants au cours de la transaction déclenchante et traite l'ensemble des événements à la fin de la transaction. Tous les examens créés pendant la transaction déclenchante sont donc traités en même temps en fin de transaction (clause on after create Examen). La règle construit ainsi l'ensemble des examens créés pendant la transaction qui n'ont encore aucun rendez-vous associé (clause if) ; s'il existe un ou plusieurs tels examens, le programme d'accord d'un rendez-vous est automatiquement exécuté pour chacun de ces examens (clause do).

Règle NAOS :

rule ExamenSansRdv
coupling deferred
on after create Examen **with** E

if range of le **is o2 set**(Examen)
select X
from X **in** E
where X→RdvExamen = NIL
do { **for** (X **in** le)
AccorderRdv(X→EntiteConcernee, X) ; }

Le mode de traitement ensembliste d'une règle différée résulte d'un choix de NAOS. La possibilité qu'une règle différée ait un mode de traitement par instance rendrait la règle active beaucoup plus simple, même si celle-ci était alors exécutée plusieurs fois :

Règle NAOS :

rule ExamenSansRdv
coupling deferred
on after create Examen **with** E
if (E→RdvExamen = NIL)
do { AccorderRdv(E→EntiteConcernee, E) ; }

Ceci met en évidence que la sémantique d'un mode de couplage différé ou immédiat n'est pas la même que celle que nous lui donnons au niveau conceptuel : un mode de couplage différé correspond au niveau conceptuel au fait de ne pas consommer immédiatement l'événement (délai de la relation consomméPar à plus tard) et ainsi de tolérer une incohérence temporaire pendant une période de temps déterminée.

13. Evénement *Nouveau Rdv* (cf. figure 6.10)

Cet événement correspond à l'application du patron Activité sur trois situations comportementales : *Le brancardage est planifié lors d'une procédure de demande de rendez-vous* ; *Chaque fin de journée, éditer l'état prévisionnel des demandes de rendez-vous pour le lendemain* ; *Chaque fin de journée, éditer la liste des rendez-vous accordés le jour même.*

- La première situation comportementale peut être traduite tout naturellement dans le corps du programme AccorderRdv dont le but est l'accord d'un rendez-vous à un patient pour un examen.
- L'application du patron Activité sur la deuxième situation comportementale donne lieu à une règle d'activité typique d'un traitement asynchrone et donc

facilement traduisible dans NAOS par une règle temporelle déclenchée tous les jours à 18h00 (clause on temporal event at **/**/** : 18:00).

Si la fin de journée arrive
alors éditer l'état prévisionnel des demandes de rendez-vous pour le lendemain.

Règle NAOS :

rule PrevisionsRdvLendemain
coupling immediate
on temporal event
at **/**/** : 18:00
if range of lb **is o2 set**(RdvBrancard)
select R
from R **in** LesRdvBrancards
where R→Date = currentdate + 1 day
do { display(lb); }

où *currentdate* représente la date du jour courant.

– L'application du patron Activité sur la troisième situation comportementale donne lieu à la situation-réaction suivante :

Si la fin de journée arrive
alors éditer la liste des rendez-vous accordés le jour même.

O_2 n'offre aucun support pour savoir si le rendez-vous a été accordé le jour même, excepté en stockant la date d'accord du rendez-vous. En NAOS, il est possible de déclarer une règle active différée déclenchée à la fin du programme AccorderRDV. Ainsi, grâce à la sémantique ensembliste d'une règle différée, tous les rendez-vous accordés pendant l'exécution de ce programme pourraient être pris en compte et affichés en même temps. Cependant dans un contexte hospitalier réel, ce programme est appelé plusieurs fois au cours d'une même journée dans un environnement distribué et multi-transactionnel. Or, les événements ne sont détectés dans NAOS que dans un environnement mono-transactionnel. Ce cas ne peut donc être pris en considération et il devient fondamental non seulement de détecter des événements dans un environnement multi-transactionnel, mais aussi d'exécuter des règles dans un environnement réparti. Des travaux sont en cours dans notre équipe pour tenter d'apporter des solutions à ce problème [LCD97].

Pour contourner ces limites, nous proposons une solution basée sur l'utili-

sation de deux règles actives. La règle active NouveauRdv est déclenchée à la fin de chaque transaction au cours de laquelle un ou plusieurs nouveaux rendez-vous sont créés (clause on after create Rdv) et met à jour la racine de persistance LesRdvAujourdhui pour stocker tous les rendez-vous accordés le jour même (clause do). La règle active RdvAccordesAujourdhui est déclenchée tous les jours à 18 heures (clause on temporal event at **/**/** : 18:00) et affiche les rendez-vous de cette racine de persistance (clause do).

Règles NAOS :

rule NouveauRdv
coupling deferred
on after create Rdv **with** R
do { LesRdvAujourdhui += R ; }

rule RdvAccordesAujourdhui
coupling immediate
on temporal event
at **/**/** : 18:00
do { Afficher(LesRdvAujourdhui) ;
LesRdvAujourdhui = NIL ; }

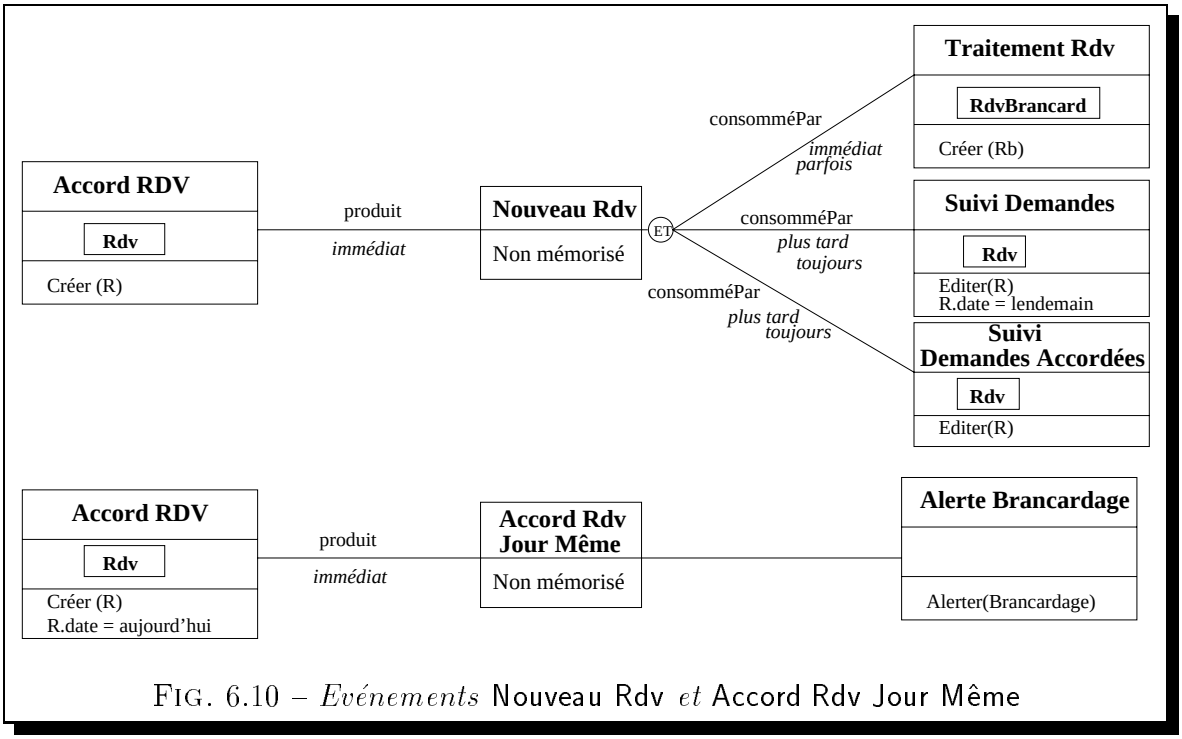


FIG. 6.10 – *Evénements* Nouveau Rdv *et* Accord Rdv Jour Même

14. Evénement *Accord Rdv Jour Même* (cf. figure 6.10)

Cet événement correspond à l'application du patron Exception sur la situation comportementale *Si un rendez-vous est accordé le même jour que la demande, alors le brancardage doit être alerté.*

Si on accorde un rendez-vous pour le jour même
alors il faut prévenir le brancardage.

Cette règle d'exception gère les cas où un rendez-vous est accordé le même jour que la demande et pourrait très facilement être implantée dans le programme AccorderRdv grâce à un traitement conditionnel classique. Cependant, l'accord d'un rendez-vous le même jour que la demande est exceptionnel. Une règle active déclenchée seulement si la date du rendez-vous concorde avec la date courante permettrait donc de réduire le coût de traitement d'un tel événement. En intégrant la notion de masque à NAOS, la règle active RdvAujourdhui ne serait déclenchée que sur l'occurrence d'un événement d'accord de rendez-vous le jour-même (clause on after update Rdv→Date with R { R→Date = currentdate }) et optimiserait considérablement la réaction à un tel événement. Sans masque, la vérification de la correspondance des dates se fait dans la partie Condition de la règle, mais celle-ci est activée pour toutes les modifications de l'attribut Date de la classe Rdv sans aucune optimisation possible.

Règle NAOS :

rule RdvAujourdhui
coupling immediate
on after update Rdv→Date **with** R { R→Date = currentdate }
do { SendMail(Brancardage) ; }

Les événements suivants peuvent être traduits de la même façon :

- *Résultats Graves* (cf. figure 6.11), correspondant à l'application du patron Exception sur la situation comportementale *Si une anomalie grave est détectée dans les résultats d'un acte d'un patient, alors le médecin du patient doit être alerté.*
- *Température Très Haute* (cf. figure 6.4), correspondant à l'application du patron Exception sur la situation comportementale *Si la température d'un patient est supérieure à 39,5 degrés, alors surveiller attentivement le patient.*

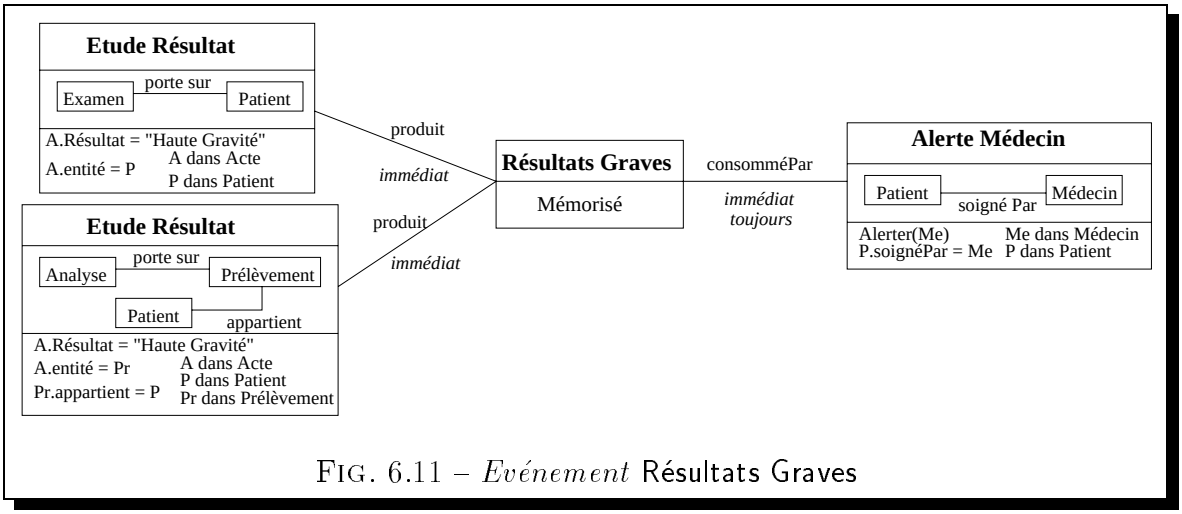


FIG. 6.11 – *Evénement* Résultats Graves

6.2.2 Bilan de la traduction

Les divers exemples présentés ci-dessus nous permettent d'évaluer certaines conclusions quant à la traduction privilégiée de chaque type de situation comportementale en NAOS. Nous entendons par traduction privilégiée celle qui est la plus naturelle pour implanter la situation comportementale, tant au niveau du choix entre un mode de traitement séquentiel classique et une règle active qu'au niveau du choix du modèle d'exécution d'une règle active. D'autre part, nous pouvons établir une synthèse des techniques les plus utiles et les plus facilement utilisables dans NAOS et apporter des propositions d'extensions à ce système actif pour implanter les situations comportementales avant d'analyser des résultats concernant les systèmes de gestion de bases de données actifs en général.

6.2.2.1 A propos de la génération de code avec SCalP

La première conclusion que nous pouvons établir concerne la ou les traductions privilégiées adoptées selon le type de situation comportementale. Cette traduction n'est possible qu'après la phase finale de l'utilisation du langage de patrons SCalP lorsque toutes les informations caractéristiques d'une situation comportementale sont saisies. Ainsi, les patrons du langage SCalP permettent de coupler les situations comportementales du monde réel en un code O_2/NAOS privilégié (cf. figure 6.12) que nous résumons ci-dessous.

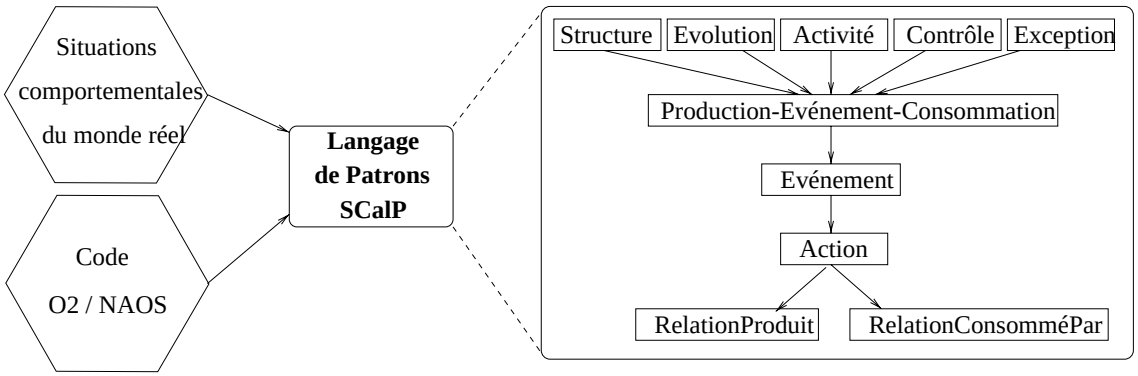


FIG. 6.12 – *Résumé du couplage "Situations comportementales - NAOS"*

1. **Règles de structure**

   Le but d'une règle de structure étant la maintenance de la structure des entités de l'application et des relations entre ces entités dans une base de données par définition cohérente, la traduction privilégiée d'une règle de structure est une règle active immédiate. L'événement auquel cette règle réagit est la modification d'une entité ou d'un attribut d'une entité de l'application. L'instant d'occurrence de l'événement est après car d'un point de vue utilisateur, il est plus naturel de réagir à une violation de contrainte après que celle-ci soit apparue. La condition de la règle est la non vérification de la contrainte que doivent respecter les entités de l'application. L'action est alors de divers types. Dans certain cas, un message d'erreur est affiché si une incohérence est tolérée (comme dans le cas de la température d'un patient égale à 40,5 degrés). Dans d'autres cas, la transaction est annulée de façon à revenir à un état cohérent lorsqu'aucune incohérence n'est permise (par exemple lors de la prescription d'aspirine à une patiente enceinte). Enfin, une action de compensation peut être déclenchée pour retrouver une situation normale sans annuler tout ce qui a été fait pendant la transaction (comme la suppression d'aspirine à une patiente que l'on diagnostique enceinte).

2. **Règles d'évolution**

   Le but d'une règle d'évolution est la gestion de l'évolution de la valeur des attributs d'une ou de plusieurs entités de l'application. Deux types d'évolution sont possibles : pendant une transaction (comparaison des valeurs en début et en fin de transaction) et relativement à la notion de temps (comparaison de valeurs à 2 jours ou à 3 mois d'intervalle).

Dans le cas d'une évolution pendant une transaction, la traduction la plus satisfaisante et la plus générale est une règle active différée réagissant après un événement de modification d'un attribut d'une classe. La condition compare la nouvelle valeur à l'ancienne et lorsque la contrainte entre ces deux valeurs n'est pas vérifiée, l'action peut soit simplement afficher un message à l'utilisateur, soit modifier les valeurs des attributs de l'entité avec respect de la contrainte.

Une évolution non liée à une transaction nécessite l'utilisation d'une règle active réagissant à un événement temporel déclenché un certain temps après la modification de l'attribut.

3. **Règles d'activité**

   La traduction d'une situation-réaction de type activité nécessite de savoir si la réaction doit être effectuée séquentiellement ou de façon asynchrone par rapport à la situation.

   Dans le premier cas, la traduction d'une règle d'activité est possible en programmant tout naturellement la réaction à la suite de la situation dans le corps d'une méthode, d'un programme ou d'une application. Cette conclusion confirme les critiques apportées à des méthodes telles IFO2 qui traduisent de telles situations sous forme de règles actives alors qu'une programmation séquentielle classique est plus naturelle.

   Dans le second cas, l'asynchronisme est dû au fait que la réaction doit être effectuée un certain temps après la situation. Une règle active immédiate réagissant à un événement temporel relatif apporte une solution à un tel besoin non exprimable facilement dans O_2.

4. **Règles de contrôle**

   Une règle de contrôle met en exergue une contrainte à vérifier avant qu'une situation ne se produise ou qu'une action ne s'exécute. Autrement dit, une proaction doit être réalisée afin de vérifier qu'une contrainte est respectée ou qu'un traitement est effectué avant qu'une action ne soit exécutée.

   Les règles actives NAOS réagissant avant l'occurrence réelle d'un événement tel la modification d'un attribut ou le déclenchement d'une méthode, sont intéressantes pour résoudre de telles situations-réactions et constituent la cible privilégiée de traduction des règles de contrôle. Tout naturellement, la partie Condition d'une telle règle active est inexistante : la vérification d'une contrainte, but même d'une règle de contrôle, se fait dans la partie Action de la règle active correspondante.

Situation	Traduction privilégiée	Mode de couplage	Type de l'événement
Structure	Règle active	Immédiat	ON AFTER ...
Evolution	Règle active	Différée	ON AFTER ...
	Règle active	Immédiat	Evénement temporel relatif
Activité	Corps d'une méthode, d'un programme, d'une application		
	Règle active	Immédiat	Evénement temporel relatif
Contrôle	Règle active	Immédiat	ON BEFORE ...
Exception	Règle active	Immédiat	ON AFTER ... Utilisation de masques

TAB. 6.6 – *Résumé du couplage en O_2 et NAOS*

5. **Règles d'exception**

Une règle d'exception est destinée à prendre en compte des situations apparaissant exceptionnellement dans l'application. Conceptuellement, cela sous-entend que l'événement ait effectivement eut lieu. Il est donc naturel d'utiliser une règle active réagissant après l'occurrence d'un événement. De plus, le traitement de l'exception devant être pris en compte immédiatement, une telle règle bénéficie du mode de couplage immédiat. Cependant, le caractère exceptionnel d'une règle d'exception la rend intéressante si elle permet de minimiser et d'optimiser le coût de traitement d'une situation exceptionnelle, ce qui n'est possible que par l'utilisation de masques permettant le déclenchement de la règle seulement dans les cas correspondants.

En conclusion, à un type de situation comportementale correspond une traduction privilégiée en O_2/NAOS résumée dans le tableau 6.6. Dans le cas de traduction en règles actives NAOS, un mode d'exécution privilégié peut aussi être adopté au niveau du mode de couplage et de l'instant d'occurrence de l'événement.

6.2.2.2 A propos de NAOS

Le but de cette section est d'analyser l'utilité des techniques proposées par NAOS. D'une part, nous nous attardons sur les techniques les plus facilement et les plus naturellement utilisables. D'autre part, nous présentons les limitations de NAOS par rapport à ce dont nous aurions aimé bénéficier pour implanter les situations comportementales des applications réactives de façon plus satisfaisante.

Utilité de NAOS - La plupart des règles actives que nous avons utilisées pour traduire les situations comportementales sont immédiates et orientées instances. Ceci est vrai non seulement pour l'application médico-technique, mais aussi pour toutes les autres applications étudiées et est dû au fait que les règles actives immédiates sont compréhensibles naturellement. D'autre part, elles couvrent la majorité des types de situations comportementales mis en évidence, même si pour les règles d'activité, une expression séquentielle classique est plus naturelle. Les règles différées ne sont réellement utiles que pour contrôler l'évolution des entités de l'application ou lorsque la consommation peut s'effectuer à un moment ultérieur en tolérant éventuellement une incohérence temporaire des données. Cependant, les règles différées sont très difficiles à utiliser car leur sémantique ensembliste pose des problèmes d'interprétation et complique l'écriture d'une règle alors qu'une sémantique orientée instance serait parfois utile. Par opposition au mode immédiat, le mode différé signifie à un niveau conceptuel que la consommation de l'événement peut se faire à un moment ultérieur ou qu'une incohérence temporaire de la base peut être tolérée.

Restriction d'utilisation de NAOS - L'implantation d'applications réactives avec le prototype actuel NAOS nécessite des adaptations de nos spécifications alors qu'en ayant bénéficié de certaines nouvelles fonctionnalités, nous aurions pu implanter plus facilement certaines situations comportementales. En particulier, nous avons constaté :

- **l'absence d'environnement d'exécution pour les événements temporels**. Plus largement ce problème met en évidence, pour un événement temporel relatif, la nécessité de garder une trace de l'environnement d'exécution de l'événement relaté ;

- **la difficulté de retrouver le contexte lié au chemin d'un événement lors de la traversée d'objets**. Ce problème se pose par exemple lorsque l'on souhaite retrouver après la modification d'un médecin, l'ensemble des prélèvements sur lesquels ce médecin a prescrit des analyses (un prélèvement ayant un propriétaire (un patient) qui a lui-même un médecin traitant). Dans ce cas, il est nécessaire de gérer l'événement

  on after update Prelevement→Proprietaire→MedecinTraitant

  afin de conserver la trace du chemin complet ayant abouti à un médecin à partir d'un prélèvement. Ce problème difficile à résoudre est d'autant plus important qu'O_2 n'offre pas de gestion de liens inverses qui permettraient de retrouver plus

facilement les prélèvements à partir du médecin : ainsi, il serait possible de déclarer une règle réagissant à l'événement on after update Medecin et de retrouver les prélèvements sur lesquels le médecin a prescrit des analyses. L'utilisation et l'implantation des règles actives seraient grandement facilitées, même si les liens inverses modifient la modélisation et la conception du schéma de l'application. Notons que cet aspect "lien inverse" fait maintenant partie de la norme ODMG [Cat94] ;

- **l'obligation d'un mode de traitement ensembliste avec une règle différée** ; ce paramètre du modèle d'exécution de NAOS complique l'écriture de certaines règles pour implanter des situations comportementales alors que des règles différées orientées instances seraient plus satisfaisantes pour traduire certaines situations comportementales. Pour résoudre de tels problèmes, de récents travaux [Cou96] visent à paramétrer le modèle d'exécution d'un SGBD actif en fonction des applications ;

- **l'absence d'une période de validité d'une règle active**, même si ce problème peut être contourné en associant une période de validité à l'événement déclenchant la règle. Dans ce cadre, il conviendrait d'ailleurs d'élargir la notion de période de validité des événements temporels à tous les types d'événements et de prendre en compte des périodes de validité dynamiques dépendant des données de la base, ce que permettent les types d'événements dynamiques [Ron97].

6.2.2.3 A propos des systèmes de gestion de bases de données actifs

Au delà du système NAOS se pose le problème de l'utilisation des systèmes de gestion de bases de données actifs en général. NAOS étant représentatif des autres systèmes de gestion de bases de données actifs, il est vraisemblable que ses restrictions d'utilisation pour l'implantation des situations comportementales seront constatées dans les autres systèmes et devront être résolues afin d'améliorer l'utilisation de cette technologie. Nous manquons d'expérience dans les autres systèmes pour affirmer complètement cette supposition néanmoins raisonnable.

De plus, certains aspects sont essentiels pour une meilleure utilisation des systèmes de gestion de bases de données actifs.

- Prise en compte des **environnements distribués** et des **transactions multiples** apparaissant dans des contextes tels qu'un centre hospitalier : les systèmes

de gestion de bases de données actifs doivent être adaptés à un environnement distribué multi-transactionnel en permettant non seulement la détection d'événements dans des transactions multiples, mais aussi l'exécution de règles réparties.

- Intégration d'**événements dynamiques** tels ceux proposés dans [Ron97] pour exprimer des événements relatifs à une date stockée dans la base.

- Association de **masques** aux événements pour d'une part améliorer l'expression de l'événement et obtenir une sémantique proche d'un utilisateur, d'autre part optimiser et réduire le coût du traitement de certaines situations exceptionnelles.

- Prise en compte du mode d'exécution Détaché pour l'exécution des règles actives, comme proposé dans [CM95]. Ce mode implique que la partie déclenchée et la partie déclenchante s'exécutent dans deux transactions différentes et est pertinent pour prendre en compte les situations comportementales exécutées de manière asynchrone.

6.3 Le prototype SCalP : un outil pour la conception de situations comportementales en NAOS

Le prototype SCalP a été développé selon deux buts essentiels :

- l'intégration du concept de patron pour la représentation des besoins et plus généralement l'utilisation de ce concept pour la modélisation d'un système d'information. Naturellement, nous nous attachons à représenter des situations comportementales à l'aide du langage de patrons SCalP ;

- la génération de règles actives en NAOS, avec la volonté principale de montrer la pertinence et l'utilisabilité des concepts introduits par NAOS pour la représentation de situations du monde réel.

6.3.1 Architecture du prototype

Le prototype SCalP a été développé en utilisant les méta-outils GraphTalk [Par93] et LEdit [Par94] présentés dans l'annexe D. Il permet la représentation de situations comportementales à l'aide du langage de patrons SCalP. Cette version du prototype regroupe les patrons Production-Evénement-Consommation, Evénement, Action,

RelationProduit et RelationConsomméPar en une seule étape. L'utilisation des patrons Structure, Evolution, Activité, Contrôle et Exception n'est pas prise en compte par l'outil et est laissée à la charge de l'utilisateur.

L'architecture du prototype SCalP est donnée dans la figure 6.13. Le prototype est composé de deux modules principaux : le module SCalP Représentation pour la représentation de situations comportementales à l'aide du langage de patrons et le module SCalP Génération pour la génération de règles actives NAOS. Ces deux modules communiquent via une interface de programmation (l'API de GraphTalk) en C.

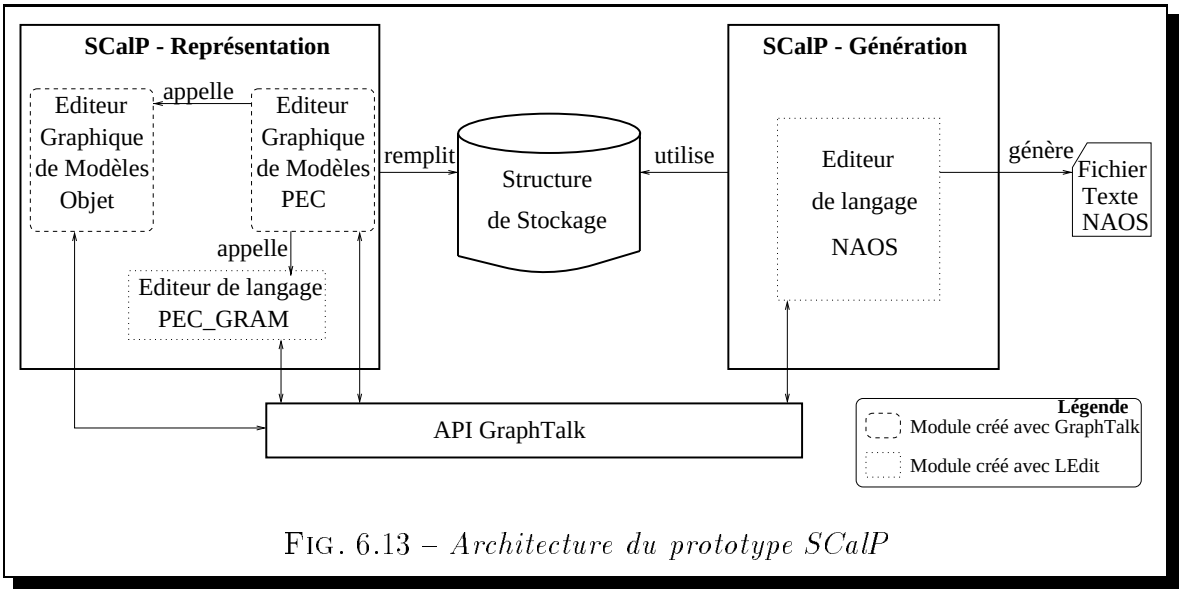


FIG. 6.13 – *Architecture du prototype SCalP*

6.3.1.1 Module SCalP Représentation

Ce module destiné à la représentation de situations comportementales à l'aide du langage SCalP, a comme composant principal un éditeur graphique de modèles de type Production-Evénement-Consommation développé avec GraphTalk et permettant la création d'instances du patron Production-Evénement-Consommation. Cet éditeur appelle un autre éditeur graphique développé avec GraphTalk : l'éditeur graphique de modèles objet, afin de représenter la partie Concepts mise en évidence par le patron Action sous forme de modèles objets traditionnels. Enfin, il appelle un éditeur de langage développé avec LEdit et destiné à représenter la partie Conditions mise en évidence par le patron Action. En effet, si au niveau conceptuel, les conditions peuvent être exprimées sans utiliser une syntaxe particulière, l'utilisation du prototype nécessite l'emploi d'une grammaire formelle suffisamment simple et proche de la langue naturelle pour être compréhensible par le demandeur de l'application et son concepteur. Cette

- Valeur d'un attribut selon le format InstanceClasse.NomAttribut = ValeurAttribut, par exemple Ma.nom = "Diabète" où Ma est une instance de Maladie.
- Appel de méthode d'une classe selon le format InstanceClasse.NomMéthode(Paramètres), par exemple M.PrescrireAnalyses(P) où M est une instance de Médecin.
- Instance d'une association mono-valuée entre deux classes selon le format InstanceClasse.NomAssociation = InstanceClasse, par exemple P.a = M où M et P sont des instances respectivement de Médecin et de Patient si l'on suppose qu'un patient a un seul médecin dans un hôpital.
- Instance d'une association multi-valuée entre deux classes selon le format InstanceClasse dans NomClasse, par exemple Me dans Médecin.
- Appartenance d'une instance à une classe selon le format InstanceClasse dans InstanceClasse.NomAssociation, par exemple Ma dans P.estAtteint où Ma et P sont des instances respectivement de Maladie et de Patient.

TAB. 6.7 – *Types de conditions possibles*

grammaire (appelée *PEC GRAM*) permet de conditionner la réalisation d'une action en fonction des classes utilisées dans la partie Concepts et d'apposer des conditions de cinq types différents (cf. table 6.7).

Les informations acquises par chacun des trois éditeurs permettent au module SCalP Représentation de remplir une structure de stockage utilisée par le module SCalP Génération.

6.3.1.2 Module SCalP Génération

Le module SCalP Génération destiné à la génération de règles actives NAOS, consiste en un éditeur de langage NAOS développé avec LEdit. Il utilise la structure de stockage précédente pour génèrer un fichier texte composé des règles actives NAOS correspondant aux situations comportementales représentées par le module SCalP Représentation.

Le fichier texte généré peut ensuite être utilisé comme n'importe quel fichier texte dans NAOS (cf. figure 6.14). L'étape d'analyse n'est cependant pas nécessaire puisque l'éditeur développé avec LEdit assure une syntaxe des règles conforme à la grammaire NAOS présentée selon le format LEdit dans l'annexe E.

6.3.2 Réalisation du prototype avec GraphTalk

La réalisation du prototype est basée sur le développement en GraphTalk de l'éditeur de modèles Production-Evénement-Consommation.

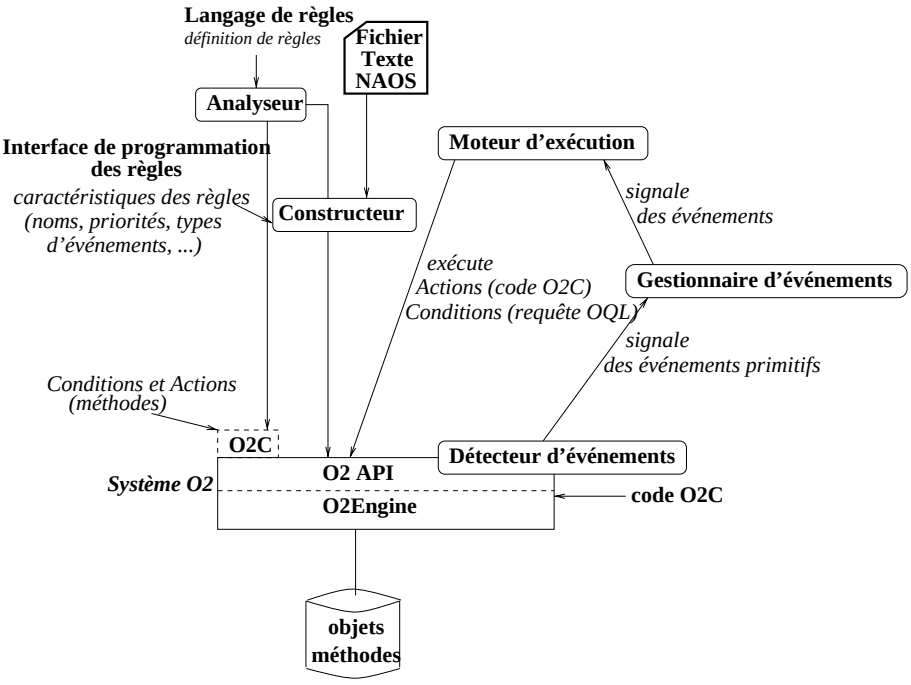


FIG. 6.14 – *Architecture de NAOS*

6.3.2.1 Spécification sémantique

Les concepts essentiels du langage de patrons sont introduits en construisant un graphe Patron (cf. figure 6.15). A ce graphe sont rattachées les instances de la classe Objet **Evénement**, **Production**, **Consommation** et **Action**. Les productions et les consommations sont des actions. De plus, une production est liée à un événement par le lien **Produit** et un événement est lié à une consommation par le lien **ConsomméPar**. Notons que les instances **Concepts** et **Conditions** qui n'ont d'existence que par rapport à une action ne sont pas rattachées au graphe Patron.

6.3.2.2 Affectation des propriétés

Conformément au langage de patrons, des propriétés sont affectées aux concepts introduits (cf. figure 6.16).

- Un événement possède la propriété **Mémorisation** de type menu à deux valeurs possibles : mémorisé ou non mémorisé (cf. fenêtre Objet - Mémorisation).
- La relation Produit (respectivement ConsomméPar) possède la propriété de type Texte **DélaiP** (respectivement **DélaiC** et **PériodeValidité**).

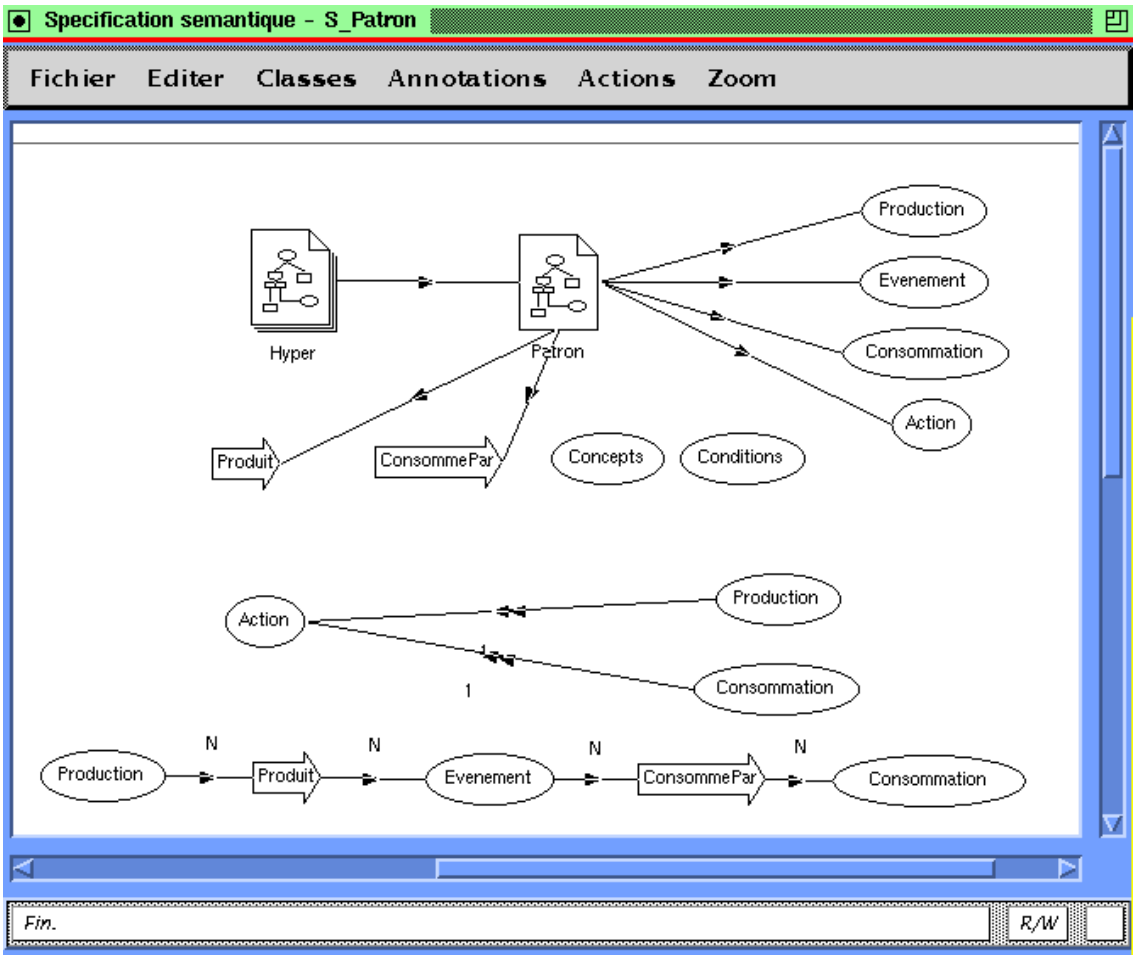


FIG. 6.15 – *Spécification sémantique - Graphe Patron - SCalP*

– Une action possède les propriétés **Concepts** et **Conditions** de type Objets. L'objet Concepts est associé au graphe **Objet** par l'intermédiaire de la propriété **Concepts Modèle** de type Graphes. L'objet Conditions est associé à la grammaire **PEC GRAM** par l'intermédiaire de la propriété **ConditionsPEC** de type Texte structuré (cf. fenêtre Objet - ConditionsPEC).

– Enfin, la grammaire **NAOS** est attachée au graphe Patron grâce à la propriété de type texte structuré **RèglesNAOS** (cf. fenêtre Objet - ReglesNAOS).

6.3.2.3 Spécification des formes

Par convention, un événement et une action sont représentés sous forme d'un rectangle avec leur nom spécifié à l'intérieur (cf. figure 6.17). Les parties Concepts et

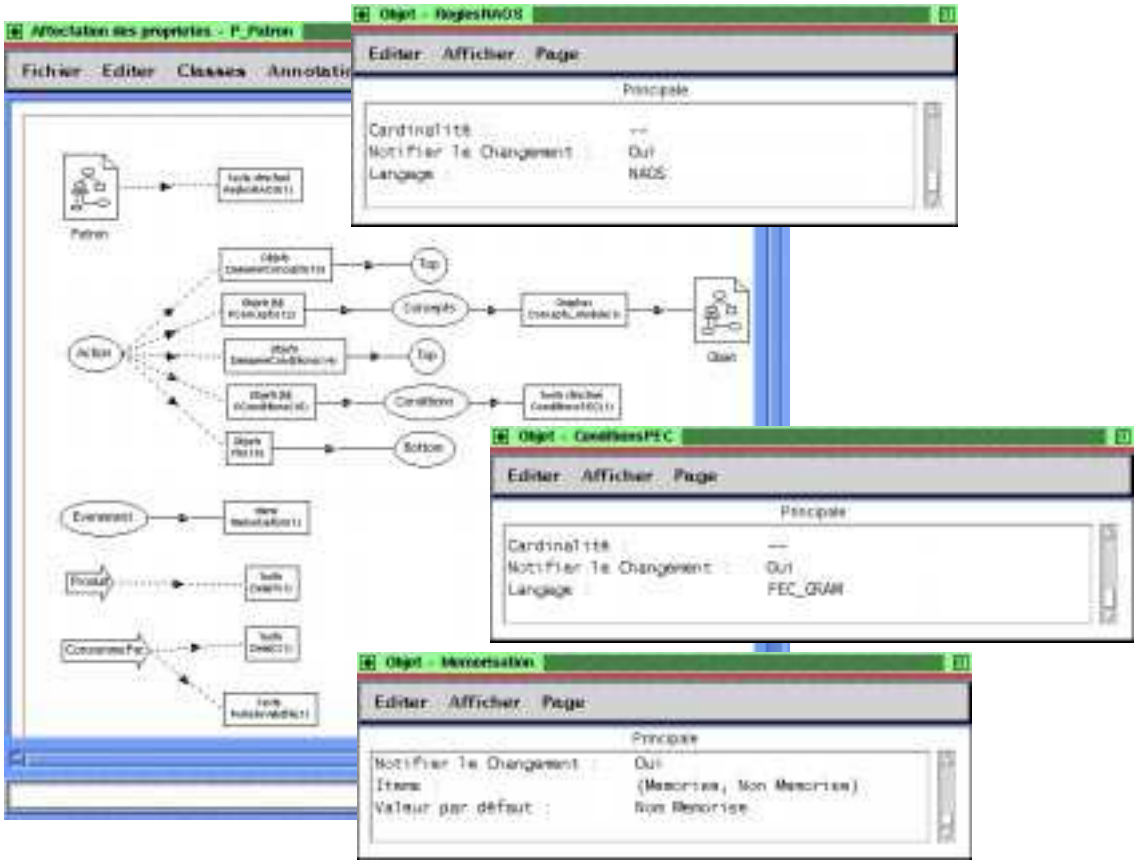


FIG. 6.16 – *Affectation des propriétés - Graphe Patron - SCalP*

Conditions rattachées à une action ont une forme rectangulaire où sont indiqués respectivement les mots CONCEPTS et CONDITIONS ; ceci permet de reconnaître une action d'un événement et d'accéder facilement d'une part au modèle objet correspondant à la partie Concepts, d'autre part à la grammaire PEC GRAM correspondant à la partie Conditions.

6.3.2.4 Spécification des fenêtres

Des menus spécifiques à l'utilisation souhaitée pour le prototype SCalP sont spécifiés (cf. figure 6.18). Ainsi, le sous-menu standard Classes du graphe Patron est remplacé par le sous-menu spécifique **Créer** regroupant quatre actions : l'action **ProductionEvénementConsommation** permet de créer une instance du patron Production-Evénement-Consommation, c'est-à-dire un événement lié à une production et à une consommation ; les actions **Production**, **Evénement** et **Consommation** permettent

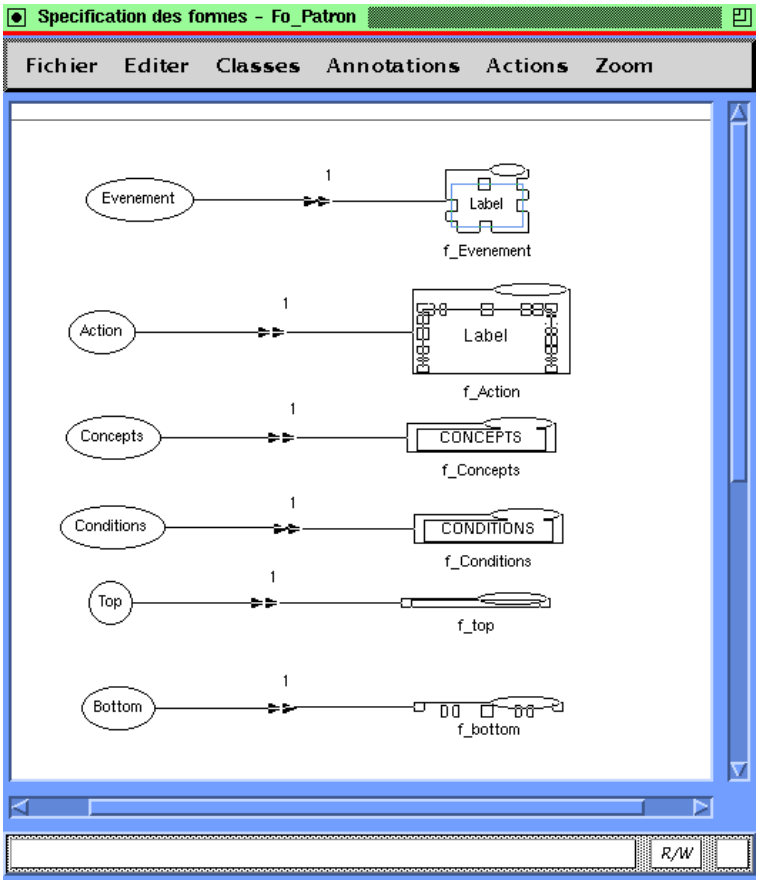


FIG. 6.17 – *Spécification des formes - Graphe Patron - SCalP*

respectivement de créer une instance des objets Production, Evénement et Consommation. De la même façon, le sous-menu **Générer** est attaché au graphe Patron et permet d'appeler grâce à une requête GQL, la grammaire NAOS remplie.

6.3.3 Utilisation du prototype

Comme tout atelier réalisé avec GraphTalk, c'est une instance du prototype développé qui est utilisable après compilation. L'utilisation d'une instance de l'atelier généré suit les étapes proposées par le langage de patrons : création d'un événement lié à une ou plusieurs productions et à une ou plusieurs consommations, mémorisation de l'événement, détail des parties Concepts et Conditions des productions et des consommations, enfin détail des relations Produit et ConsomméPar. Notons que la première étape suit elle-même deux phases : tout d'abord, création d'une instance du patron Production-Evénement-Consommation impliquant la création en même temps

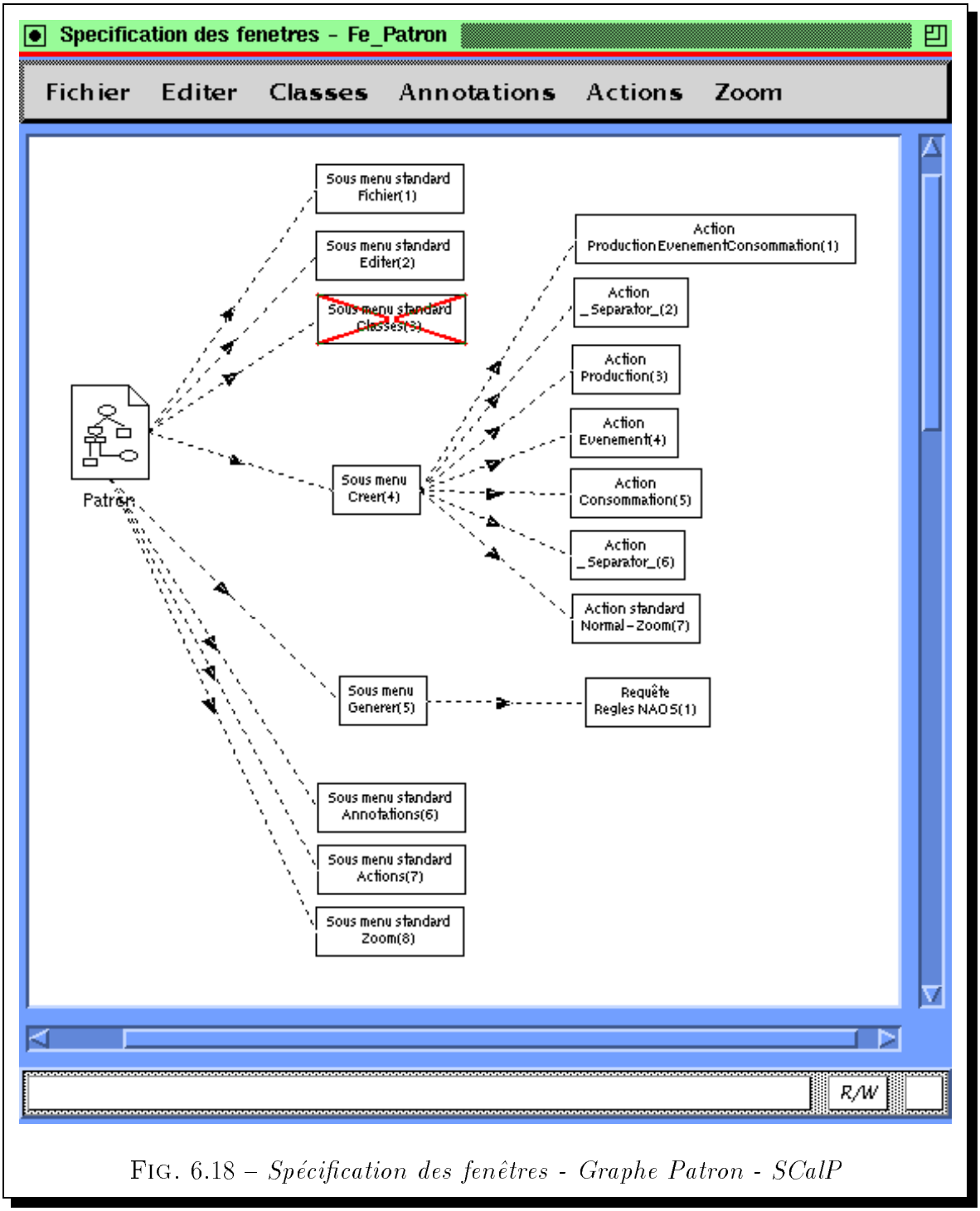


FIG. 6.18 – *Spécification des fenêtres - Graphe Patron - SCalP*

d'une production, d'un événement et d'une consommation, puis ajout éventuel de nouvelles productions ou de nouvelles consommations. La figure 6.19 montre une utilisation possible d'une instance de l'atelier généré pour la représentation des situations comportementales mises en évidence dans le cahier des charges de la section 5.3.2.

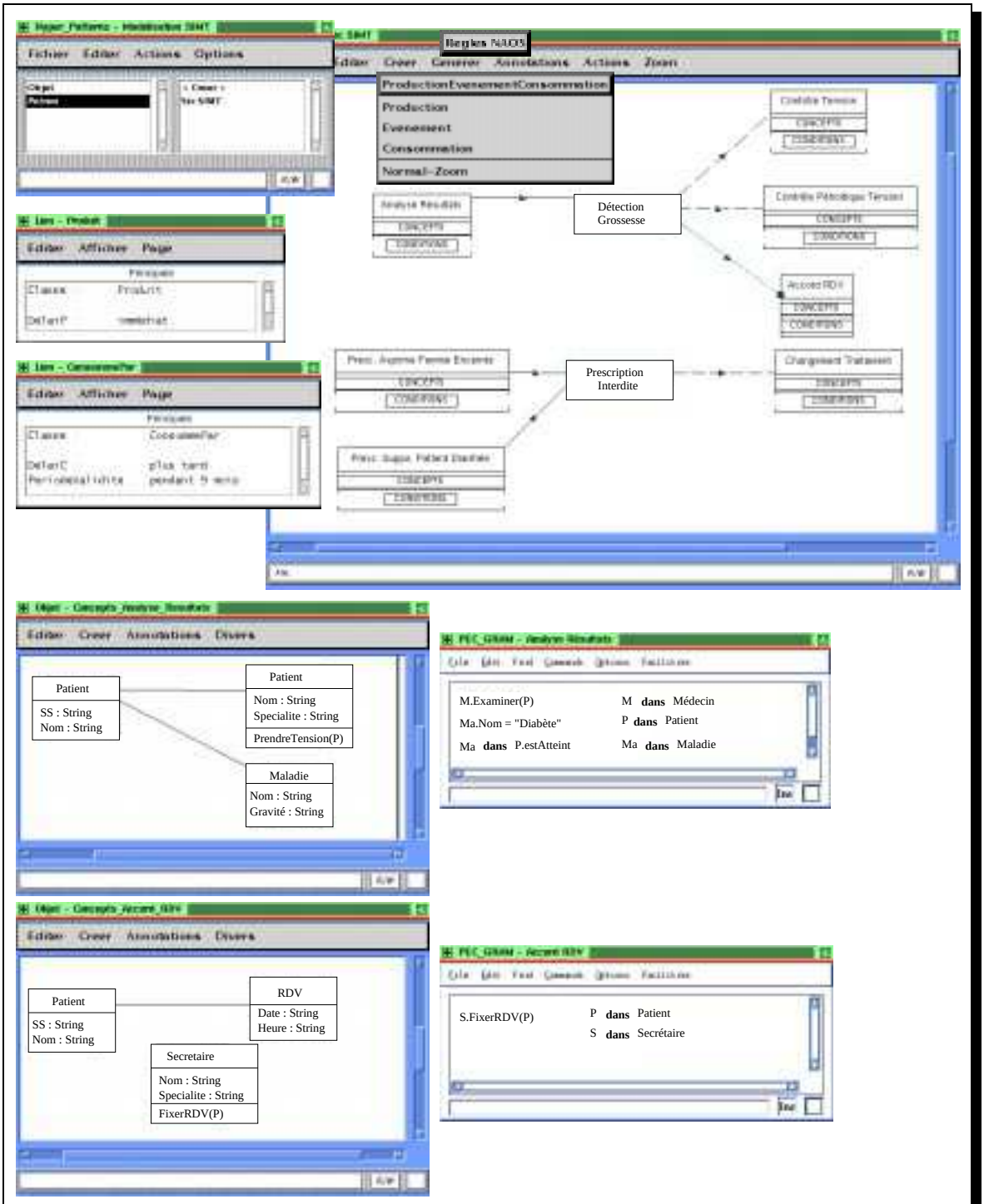


FIG. 6.19 – *Un exemple d'utilisation du prototype SCalP*

6.4 Conclusion : vers d'autres systèmes...

L'approche SCalP permet la représentation de situations comportementales présentes dans les applications réactives de différents domaines. Plusieurs systèmes sont

adaptés pour implanter de telles situations et l'expérimentation que nous avons réalisée dans le cadre des systèmes de gestion de bases de données actifs conduit à plusieurs remarques.

D'une part, la faisabilité de notre approche est établie : l'approche SCalP permet à la fois de représenter les situations comportementales du monde réel et de coupler celles-ci aux technologies offertes par les systèmes de gestion de bases de données actifs. Il est cependant nécessaire d'adapter l'approche pour prendre en compte des éléments résultant d'une différence d'interprétation de mêmes concepts entre le niveau conceptuel et le niveau implantable. Cette différence implique en particulier qu'un événement au niveau conceptuel est traduisible de différentes façons au niveau implantable.

D'autre part, l'utilisabilité des systèmes de gestion de bases de données actifs pour représenter des situations comportementales du monde réel est démontrée. L'expérimentation a été menée sur un système actif particulier, le système NAOS. Ce système est représentatif des autres systèmes de gestion de bases de données actifs et nous permet de penser que cette technologie offre de gros avantages pour programmer des situations comportementales complexes difficilement gérables avec un mode de programmation séquentiel classique.

Enfin, si les règles actives apportent des facilités d'implantation et une importante réduction du coût de traitement des situations comportementales, elles doivent néanmoins être utilisées avec parcimonie : dans certains cas, une programmation séquentielle classique est plus naturelle à comprendre pour un utilisateur qu'une règle active. De plus, certains concepts proposés pour augmenter les capacités d'un système actif sont difficiles à utiliser : en particulier, les événements composites correspondant à des disjonctions d'événements primitifs ne sont utiles que si les événements ont des sémantiques proches et si la réaction à chacun de ces événement est la même. S'ils sont utiles dans de nombreux cas, les événements composites compliquent parfois fondamentalement la compréhension des règles et doivent être utilisés avec méthode. D'une façon générale, une méthodologie d'utilisation des règles actives dans tout SGBD actif est souhaitable, en particulier en ce qui concerne l'endroit où les règles doivent être définies.

En conclusion, l'expérimentation que nous avons réalisée sur le système NAOS promet des perspectives intéressantes non seulement avec d'autres systèmes de gestion de bases de données actifs, mais aussi avec d'autres types de systèmes (systèmes de gestion de bases de données déductifs, systèmes experts, etc.).

Chapitre 7

Conclusion et perspectives

L'OBJECTIF GLOBAL DE CE TRAVAIL était de faciliter l'expression et l'implantation des aspects réactifs des applications par des techniques plus efficaces et mieux adaptées aux perceptions des utilisateurs. Nous résumons comment cet objectif a été atteint par les propositions concrètes de cette thèse et plus généralement par les apports des approches à base de patrons. Enfin, nous donnons quelques perspectives à notre travail.

7.1 Bilan et contributions

Notre travail de recherche a été initialisé par l'étude d'applications du monde réel dont le cahier des charges comportait explicitement des besoins comportementaux exprimables en particulier par des questions du genre *"que faire si...?"*. Les réponses à ce type de questions mettent en exergue une très forte notion de comportement et de dynamique de l'application pour exprimer des réactions à des situations réelles. Les approches traditionnelles de représentation du comportement sont peu adaptées pour prendre en compte de telles **situations comportementales** ; aussi avons-nous centré notre thèse sur une adaptation des principes d'une nouvelle approche de l'ingénierie des systèmes : l'**approche à base de patrons**.

7.1.1 Principaux résultats

Cette thèse propose tout d'abord une **classification de situations comportementales** communes à des applications de domaines différents. Cette classification qui peut servir de guide pour l'analyse d'applications de nature réactive, met en évi-

dence cinq types de besoins comportementaux dont les buts sont le maintien de la structure des entités de l'application, le contrôle de l'évolution des objets, la gestion de l'activité de l'application, le contrôle du bon fonctionnement de l'application et la réaction à des situations exceptionnelles ou à des erreurs. Ces situations comportementales engendrent un comportement de l'application soit parce qu'elles doivent toujours être vérifiées, soit parce qu'elles représentent des réactions en réponse à des situations habituelles ou inhabituelles.

Nous proposons ensuite une abstraction de ces situations comportementales selon un modèle unifié de **situations-réactions** et nous montrons la pertinence de la technique des patrons dans le cadre de ce modèle unifié. Concrètement, nous développons un langage (le langage SCalP) adapté aux situations comportementales et composé d'un **ensemble de patrons** basés sur des notions d'événements produits et consommés. Ce langage est utilisable dès le niveau de l'expression des besoins et inclut des éléments d'une **démarche organisée** par l'utilisation successive des patrons proposés. Notre démarche et nos patrons permettent de pallier certaines des limites des techniques existantes de représentation du comportement en introduisant à la fois des aspects statiques montrant des relations entre des classes du système d'information et des aspects dynamiques décrivant des comportements d'entités de l'application. De plus, ils permettent de prendre en compte tant des comportements locaux des objets que le comportement global de l'application.

Les quelques expériences d'utilisation de nos patrons et de notre langage nous permettent de confirmer que ces techniques sont assez facilement appréhendables par un utilisateur pour exprimer des situations-réactions d'une manière naturelle. A ce titre, nous pouvons dire que nos propositions se situent à un niveau conceptuel et sont proches de la perception d'un utilisateur.

Nous conduisons finalement une expérimentation de nos propositions dans un cadre de bases de données actives en couplant le langage SCalP avec le prototype de gestion de bases de données actif NAOS. Ce couplage nécessite une spécialisation d'implantation du patron **Situation-Réaction** selon les cinq types de situations comportementales pré-identifiés (structure, évolution, activité, contrôle et exception). Cette expérimentation en cours de développement permet néanmoins de conclure quant à l'utilisation des technologies de SGBD actifs pour prendre en compte les aspects réactifs des applications dans des domaines différents. Nous pouvons ainsi constater que cette technologie permet d'implanter plus facilement les situations comportementales que celle des systèmes traditionnels. Cependant, les systèmes de gestion de bases de données actifs ne sont pas encore stabilisés et leurs évolutions (répartition des règles,

généralisation des masques d'événements, etc.) faciliteront la prise en compte des situations comportementales de manière à la fois plus performante et plus naturelle.

En conclusion, l'approche que nous proposons permet d'augmenter la réutilisation dans le développement d'une nouvelle application, tant au niveau de l'**analyse des besoins** qu'à celui de l'**implantation technique**. Naturellement, cette affirmation reste à confirmer par le développement complet d'une application.

7.1.2 Evaluation générale de la technique des patrons

Nos définitions générales de patrons dans le cadre de la représentation de situations comportementales et de la conception de bases de données actives nous permettent de mettre en évidence les principaux intérêts d'une approche à base de patrons.

L'intégration d'une approche à base de patrons dans une méthode de conception orientée objet classique comme OMT ou UML constitue tout d'abord une forme d'organisation de la modélisation à chaque étape de la conception. Ainsi, les patrons aident à l'identification des classes d'objets et des associations pertinentes d'un système d'information et organisent le modèle du système d'information en sous-modèles tant lors de l'analyse que lors de la conception ou de l'implantation d'une application.

De plus, les patrons sont des mini-systèmes complets utilisables comme support pédagogique réellement intéressant. En effet, ils représentent des besoins d'applications et sont modélisés tant sur le plan statique que sur le plan dynamique. Ces patrons, exprimés sous la forme de diagrammes OMT ou UML, offrent des solutions dans un ou plusieurs systèmes cibles. Enfin, ils sont illustrés par de nombreux exemples et leur utilisation est constamment justifiée.

La technique des patrons constitue enfin une base pour une nouvelle approche de l'ingénierie des systèmes qui, par opposition à une approche descendante classique, procède par "couplage" entre des besoins du monde réel et des solutions logicielles éprouvées dans un système cible. Cette nouvelle approche a l'avantage de traduire presque systématiquement les mêmes besoins du monde réel avec les mêmes solutions dans un système cible. Nous manquons cependant d'études et d'expériences pour évaluer l'applicabilité professionnelle de cette approche et sa combinaison avec une approche plus traditionnelle.

Par notre travail, nous avons pu évaluer l'intérêt essentiel d'une telle approche quant à son adéquation à la réduction de la **complexité des systèmes**. Cette complexité est mieux maîtrisée grâce à la définition et à la réutilisation de composants

éprouvés de granularité supérieure aux composants de base des méthodes de conception et des systèmes cibles.

7.2 Perspectives

Les propositions et expérimentations exposées dans cette thèse marquent une étape dans notre projet de recherche que nous souhaitons poursuivre à court terme et à long terme sur plusieurs aspects.

Dans un premier temps, nous souhaitons conduire une évaluation du prototype qui devra en particulier permettre d'analyser comment un utilisateur non expérimenté appréhende cette nouvelle notion de patrons. Il est évident que cette évaluation conduira à une amélioration du prototype tant au niveau des patrons proposés qu'au niveau du langage SCalP.

Dans le cadre de cette évaluation, d'autres aspects sont à étudier :

- la représentation des situations comportementales à l'aide du langage SCalP doit être effectuée sur de nouvelles applications du monde réel afin de mieux valider l'utilisation de ce langage ;

- le couplage vers NAOS doit être approfondi afin que les premiers résultats obtenus quant à la traduction privilégiée des types de situations comportementales soient étendus au niveau de tous les composants du modèle d'exécution des règles actives. Il est souvent reproché aux systèmes de gestion de bases de données actifs leur difficulté à être compréhensible : déclenchement de règles en cascade, conflit possible entre plusieurs règles, complexité des modèles d'exécution associés, etc. L'aide apportée par l'approche SCalP dans l'organisation et le choix de la nature des situations comportementales de l'application devrait permettre d'une part de réduire les possibilités de conflit entre les situations comportementales (donc entre les règles actives correspondantes) et d'autre part de faciliter la compréhension d'un ensemble de règles ;

- le couplage des situations comportementales vers d'autres types de systèmes cibles doit être étudié afin de démontrer le caractère générique de l'approche. Ainsi, l'implantation des situations comportementales dans tout type de système intégrant la notion de règle de la forme Si Tête, Alors Corps est naturelle. Nous pensons en particulier aux systèmes de gestion de bases de données déductifs ;

– enfin et surtout, il est nécessaire de combiner l'utilisation de l'approche SCalP avec d'autres patrons. En effet, les patrons de SCalP ne représentent qu'un aspect dynamique des besoins d'une application qui doit être intégré avec les autres aspects d'un système d'information (aspects statiques, fonctionnels, organisationnels, etc.). Cette intégration avec d'autres patrons permettra de modéliser et d'implanter complètement une application.

Cette dernière perspective d'intégration du langage SCalP ouvre des perspectives plus générales et à plus long terme sur les approches à base de patrons.

Les approches à base de patrons ont pour finalité principale le couplage entre des besoins du monde réel et des solutions logicielles. A l'heure actuelle, il n'existe pas encore de réel couplage entre ces deux mondes. Il est donc nécessaire de définir exactement la notion de couplage entre besoins du monde réel et solutions logicielles en définissant des patrons et des règles de traçabilité pour ces deux mondes. Ce couplage peut être étudié d'un point de vue cognitif quant à son implication vis-à-vis de la perception qu'a un utilisateur des applications.

Les langages de patrons sont aujourd'hui limités à des catalogues de patrons sans être de véritables langages. Les mécanismes offerts sont du type "recettes de cuisine" et nous ne savons pas répondre à des questions telles que comment sélectionner, spécialiser, composer ou instancier des patrons ? comment intégrer des patrons dans une approche descendante classique ? etc. Trois voies doivent être approfondies pour répondre à de telles questions.

– D'une part, il est possible de maintenir une distinction permanente entre le concept de patron et le concept de classe. Il est alors indispensable d'établir clairement les relations possibles entre patrons et entre patrons et classes.

– D'autre part, on peut considèrer que les patrons sont des classes (complexes) et il faut alors appliquer aux patrons les techniques de l'ingénierie objet. Une telle solution plus "élégante" de réification des patrons en classes permet de tirer bénéfice des mécanismes objets reconnus (instanciation, spécialisation, composition, etc.) pour l'expression des patrons.

– Enfin, on peut aussi, comme dans UML [Mul97], considérer un patron comme une collaboration générique, un regroupement remarquable de classes collaborant à la réalisation d'un ou de plusieurs services particuliers.

Quelle que soit la voie retenue, il est nécessaire de conduire des études sur des

mécanismes pour adapter, spécialiser ou composer des patrons et de mieux intégrer ces patrons dans une méthode. Cette intégration peut d'une part être vue comme une extension des méthodes du type OMT et d'autre part, s'appuyer sur le principe du cycle de vie en fontaine [HSE90] en considérant qu'une définition de patrons ne peut être tenue pour achevée que si son acceptation dans une bibliothèque a été validée. Ce dernier aspect impose en particulier la double condition : généralisation suffisante et agrégation suffisante. Une telle bibliothèque de patrons, pour être exploitable, implique nécessairement l'utilisation et peut-être la définition d'un système de recherche d'informations facilitant la recherche de patrons pour répondre à un problème donné. Un tel système est une condition forte pour la promotion des patrons comme éléments réutilisables. Il reste à imaginer, spécifier, développer et évaluer le type d'indexation adapté à des patrons ainsi que le langage de requêtes et les paramètres à privilégier dans une fonction de correspondance. Mais ceci nous engage fortement dans un autre domaine de recherche sur lequel il serait nécessaire de s'investir pleinement.

Bibliographie

[AC93] M. Adiba et C. Collet. *Objets et Bases de Données: le SGBD O_2*. Hermès, 1993.

[ACC+93] M. Adiba, C. Collet, T. Coupaye, P. Habraken, J. Machado, H. Martin et C. Roncancio. Triggers Systems: Different Approaches. Rapport Survey Aristote SUR007, Laboratoire de Génie Informatique, Grenoble, 1993.

[ACM96] ACM. Software Patterns. *Communications of the ACM*, 39(10), Octobre 1996.

[AEFdC90] E. Andureau, P. Enjalbert et L. Farinas del Cerro. *Logique Temporelle, Sémantique et Validation de Programmes Parallèles*. Masson, 1990.

[AIS+77] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King et S. Angel. *A Pattern Language*. Oxford University Press, New York, 1977.

[AJLGLP94] L. Al-Jadir, A. Le Grand, M. Léonard et O. Parchet. Contribution to the evolution of information systems. Dans *Proceedings of IFIP Working Conference on Methods and Associated Tools for the Information Systems Lifecycle (CRIS'94)*, Maastricht, Pays-Bas, Septembre 1994.

[Ale79] C. Alexander. *The Timeless Way of Building*. Oxford University Press, New York, 1979.

[App97] B. Appleton. Patterns and Software: Essential Concepts and Terminology. http://www.enteract.com/ bradapp/, Mai 1997.

[Bas94] M. Basso. Manufacturing Applications : First Description. Rapport IDEA.DE.21T.005, Projet Esprit IDEA - TXT Ingegneria Informatica, Milan, Italie, Novembre 1994.

[BC87] K. Beck et W. Cunningham. Using Pattern Languages for Object-Oriented Programs. Dans Norman MEYROWITZ, éditeur, *OOPSLA'87*, Octobre 1987.

[BFGLG96] M. Bonjour, G. Falquet, J. Guyot et A. Le Grand. *Java : de l'esprit à la méthode.* International Thomson Publishing France, 1996.

[Bid93] N. Bidoit. *Bases de Données Déductives - Présentation de DATALOG.* Armand Colin, 1993.

[BJ94] K. Beck et R. Johnson. Patterns Generate Architecture. Dans *ECOOP'94*, Juillet 1994.

[BJR96] G. Booch, I. Jacobson et J. Rumbaugh. *The Unified Modeling Language ; Documentation Set ; Version 0.91 Addendum ; UML Update.* Rational Sotware Corporation, 1996.

[BK93] D. Badouel et A. Khaled. *La programmation C et C++.* Hermès, 1993.

[BLR92] R.J. Brachman, H.J. Levesque et R. Reiter, éditeurs. *Knowledge Representation.* MIT/Elsevier, 1992.

[BM94] M. Basso et G. Monteleone. New Application Domains : First Investigation Report. Rapport IDEA.DE.21T.003, Projet Esprit IDEA - TXT Ingegneria Informatica, Milan, Italie, Novembre 1994.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad et M. Stal. *Pattern-Oriented Software Architecture - A System of Patterns.* Wiley and Sons Ltd., 1996.

[Boe86] B.W. Boehm. A Spiral Model of Software Development and Enhancement. *ACM Sigsoft*, 11(4), 1986.

[Boo91] G. Booch. *Object Oriented Design with Applications.* The Benjamin/Cummings Publishing Company, Inc, 1991.

[BP93] F. Bodart et Y. Pigneur. *Conception Assistée des Systèmes d'Information - Méthode, Modèles, Outils.* Masson, 1993.

[BR95] G. Booch et J. Rumbaugh. *Unified Method for Object-Oriented Development ; Documentation Set ; Version 0.8.* Rational Sotware Corporation, 1995.

[Bra83] G.W. Brams. *Réseaux de Petri : théorie et pratique.* Masson, 1983.

[Bra88] I. Bratko. *Programmation en PROLOG pour l'intelligence artificielle.* InterEditions, 1988.

[BRJ97a] G. Booch, J. Rumbaugh et I. Jacobson. UML Notation Guide - Version 1.0. Rapport, Rational Software Corporation, Janvier 1997. http://www.rational.com.

[BRJ97b] G. Booch, J. Rumbaugh et I. Jacobson. UML Semantics - Version 1.0. Rapport, Rational Software Corporation, Janvier 1997. http://www.rational.com.

[Bru93] J. Brunet. *Analyse Conceptuelle Orientée Objet.* Thèse de Doctorat, Université Paris VI, Mars 1993.

[Béz95] J. Bézivin. Technologie objet et ingénierie des besoins : une réconciliation nécessaire. *L'OBJET : Logiciel, bases de données, réseaux*, 1(1), 1995.

[CAM93] S. Chakravarthy, E. Anwar et L. Maugis. Design and Implementation of Active Capability for an Object-Oriented Database. Rapport UF-CIS-TR-93-007, University of Florida, Gainesville, USA, Janvier 1993.

[Cas94] R. Casallas. Using Triggers in a Software Configuration Manager. Rapport, LGI-IMAG, Grenoble, 1994.

[Cat94] R.G.G. Cattel. *The Object Database Standard : ODMG 93.* Morgan Kaufman Publishers, 1994.

[CC95] C. Collet et T. Coupaye. The NAOS system. Dans *The 1995 ACM SIGMOD International Conference on Management of Data (Exhibit program)*, San Jose, USA, Mai 1995.

[CC96] C. Collet et T. Coupaye. Primitive and Composite Events in NAOS. Dans *Actes des 12emes Journées Bases de Données Avancées*, Cassis, Août 1996.

[CCC+96] C. Cauvet, R. Cicchetti, C. Collet, A. Doucet, B. Faure, M-C. Fauvet, A. Front, S. Gançarski, A. Gaudillère, J-P. Giraudin, G. Jomier, L. Lakhal, A. Meyer, S. Monties, N. Novelli, M-V. Ould-Mohammed, P. Poncelet, F. Semmak, P-C. Scholl et M. Teisseire. Modélisation du comportement et contrôle de l'évolution d'une application persistante. Rapport,

Rapport final, Action de Soutien Programmée du GDR BD regroupant les équipes du CRI, LAFORIA, LAMSADE, LIM, LIMOS, LIRMM et LSR-IMAG, Septembre 1996.

[CCS94] C. Collet, T. Coupaye et T. Svensen. NAOS Efficient and Modular Reactive Capabilities in an Object-Oriented Database System. *Actes de la 20eme Conférence Internationale VLDB - Santiago*, Septembre 1994.

[CD87] J.M. Chatain et A. Duchaussoy. *Systèmes Experts - Méthodes et Outils.* Eyrolles, 1987.

[CF97] S. Ceri et P. Fraternali. *Designing Database Applications with Objects and Rules - The IDEA Methodology.* Addison-Wesley, 1997. ISBN 0-201-403692-1.

[CHR96] C. Collet, P. Habraken et C. Roncancio. Règles actives dans les SGBD. *Ingénierie des Systèmes d'Information (ISI)*, 4(3), 1996.

[CM93] S. Ceri et R. Manthey. First Specification of Chimera (CM and CL). Rapport IDEA.DD.2P.004.02, Projet Esprit IDEA, Mai 1993.

[CM95] C. Collet et J. Machado. Optimization of active rules with parallelism. Dans *International Workshop on Active and Real-Time Database Systems (ARTDB-95)*, Skovde - Suède, Juin 1995.

[CM96] D. Coad, P. with North et M. Mayfield. *Object Models - Strategies, Patterns and Applications.* Yourdon Press Computing Series, 1996.

[Coa92] P. Coad. Object-Oriented Patterns. *Communications of the ACM*, 35(9), Septembre 1992.

[Col96] C. Collet. Bases de Données Actives : des systèmes relationnels aux systèmes à objets. Rapport RR 965-I-LSR 4, LSR-IMAG, Grenoble, Novembre 1996. http:
www-lsr.imag.fr.

[Col97] C. Collet. *Naos*, chapitre du livre Active Rules for Databases. Springer Verlag, Norman W. Paton édition, Octobre 1997.

[Con92] IDEA Consortium. Technical Annex. Rapport, Projet Esprit IDEA (EP6333), Mars 1992.

[Con96] The ACT-NET Consortium. The Active Database Management System Manifesto: A Rulebase of ADBMS Features. *SIGMOD Record*, 25(3), Septembre 1996.

[Cop96] J. Coplien. *Patterns - The Patterns White Paper*. 1996. ISBN 1-884842-50-X.

[Cou96] T. Coupaye. *Un modèle d'exécution paramétrique pour systèmes de bases de données actifs*. Thèse de Doctorat, Université Joseph Fourier, Novembre 1996.

[CPFJF+97] M. Chabre-Peccoud, J.C. Freire-Junior, A. Front, J-P. Giraudin et R. Guetari. *Ingénierie Objet - Concepts, techniques et méthodes*, chapitre 1 - Objets, langages et méthodes. InterEditions, 1997.

[CPT96] R. Cicchetti, P. Poncelet et M. Teisseire. Modélisation et Vérification Comportementales. Dans *Actes du Congrès INFORSID'96*, Bordeaux, Juin 1996.

[CS95] J. Coplien et D. Schmidt, éditeurs. *Pattern Languages of Program Design*. Addison-Wesley, 1995.

[CY90] P. Coad et E. Yourdon. *Object Oriented Analysis*. Yourdon Press, Prentice-Hall, 1990.

[DBB+88] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, D. Mac Carthy, A. Rosental et S. Sarin. The HIPAC Project: Combining Active Databases and Timing Constraints. *SIGMOD Record*, 17(1):51–69, 1988.

[Dep83] Department of Defense. *ADA Reference Manual*, ANSI/MIS-STD 1815 édition, 1983.

[DN66] O.J. Dahl et K. Nygaard. SIMULA, an Algol-based simulation language. *Communications of the ACM*, 9(9):671–678, September 1966.

[DR94] P. Dechamboux et C. Roncancio. PEPLOM$_d$: an Object Oriented Database Programming Language extended with Deductive Capabilities. Dans Springer-Verlag, éditeur, *LNCS*, volume 856, pages 2–14, Athènes, Septembre 1994. Database and Expert Systems - DEXA.

[EC94] J. Estublier et R. Casallas. The ADELE Configuration Manager. Rapport, LGI - IMAG, Grenoble, 1994.

[Far85] H. Farreny. *Les Systèmes Experts - Principes et Exemples.* Editions Cepadues, 1985.

[FCS96] M-C. Fauvet, J-F. Canavaggio et P-C. Scholl. Expressions de Requêtes Temporelles dans un SGBD à Objets. Dans *Actes des 12emes Journées Bases de Données Avancées*, Cassis, Août 1996.

[FFre] J.C. Freire et A. Front. Les Méthodes d'Analyse et de Conception Orientées Objets : Approche Qualitative. Rapport de recherche, LSR-IMAG, Groupe STORM, A paraître.

[FGF97] J-C. Freire, J-P. Giraudin et A. Front. Atelier MODSI : un Outil de Méta-Modélisation et de Multi-Modélisation. Dans *Atelier AFADL (Approches formelles dans l'assistance au développement de logiciels)*, Toulouse, Mai 1997.

[FGL95] A. Front, J-P. Giraudin et M. Lopez. Passive and Active Rules in Conceptual Models for Application Design. Dans *International Conference on Information System Concepts - ISCO3 - Towards a Consolidation of Views*, Marburg, Allemagne, Mars 1995. International Federation of Information Processing (IFIP), Working Group WG 8.1.

[FGVLV94] O. Friesen, G. Gauthier-Villars, A. Lefebvre et L. Vieille. Applications of deductive object-oriented databases using Datalog Extended Language. Dans R. Ramakrishnan, éditeur, *Applications of Logic Databases.* Kluwer, 1994.

[FJ97] J.C. Freire-Junior. *Ingénierie des Systèmes d'Information: Une Approche de Multi-Modélisation et de Méta-Modélisation.* Thèse de Doctorat, Université Joseph Fourier, Grenoble, Juillet 1997.

[Fow97] M. Fowler. *Analysis Patterns: reusable object models.* Addison-Wesley, Reading MA, 1997.

[FRG96] A. Front, C. Roncancio et J-P. Giraudin. Behavioral Situations and Active Databases Systems. Dans ACM, éditeur, *Workshop on Databases: Active and Real-Time (Concepts meet Practice), organisé dans le cadre*

de la 5eme Conférence Internationale CIKM96 (Conference on Information and Knowledge Management), Rockville, Maryland, USA, Novembre 1996.

[Fro86] R. Frost. *Introduction to Knowledge Based Systems.* Editions Collins Professional and Technical Books, 1986.

[Fro94] A. Front. Règles de Déduction et Règles Actives dans un Modèle de Systèmes d'Information et dans un Langage de Programmation pour Bases de Données. Rapport de DEA, DEA Informatique, ENSIMAG-INPG, Juin 1994.

[Fro95a] A. Front. Critères de Comparaison de Méthodes d'Analyse et de Conception Orientées Objets. Rapport Intermédiaire, ASP "Modélisation du comportement et contrôle de l'évolution d'une application persistante", Juillet 1995.

[Fro95b] A. Front. Des Modèles Conceptuels Adaptés aux Règles de Déduction et aux Règles Actives. Dans *8emes Journées Internationales "Le Génie Logiciel et ses Applications"*, Paris - La Défense, Novembre 1995.

[Fro95c] A. Front. Des Règles dans un Modèle Conceptuel : l'exemple du projet Esprit IDEA. Dans J-P. Giraudin, éditeur, *Actes des Premières Journées Jeunes Chercheurs en Modélisation et Environnements de Développement*, Grenoble, Janvier 1995. GDR-PRC Bases de Données, Pôle Modélisation et Environnements de Développement.

[Fro95d] A. Front. Ingénierie de Bases de Données Déductives et Actives. Dans A. Laribi, éditeur, *Recueil des articles présentés au Séminaire Jeunes Doctorants*, Archamps, Décembre 1995. DEA MATIS, Editions Systèmes et Information.

[Fro95e] A. Front. Système d'Information Médico-Technique. Rapport Intermédiaire, ASP "Modélisation du comportement et contrôle de l'évolution d'une application persistante", Décembre 1995.

[Fro97] A. Front. Définition de patrons pour la conception de bases de données actives. Dans *Actes du XV^{eme} congrès INFORSID'97*, Toulouse, Juin 1997.

[GD93] A.J. Gonzales et D.D. Dankel. *The Engineering of Knowledge-Based Systems - Theory and Practice.* Prentice Hall - Englewood Cliffs, 1993.

[GGD91] S. Gatziu, A. Geppert et K.R. Dittrich. Integrating Active Concepts into an Object-Oriented Database System. Dans *3rd International Workshop on Database Programming Language: Bulk Types and Persistent Data*, pages 399–415, Nafplion, 1991. Morgan Kaufmann.

[GHJV93] E. Gamma, R. Helm, R. Johnson et J. Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design. Dans Oscar Nierstrasz, éditeur, *LNCS*. Springer, Kaiserlautern (Allemagne), Juillet 1993.

[GHJV94] E. Gamma, R. Helm, R. Johnson et J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Publishing Compagny, 1994.

[Gir95] J-P. Giraudin. Evolution de la modélisation des systèmes d'information. Dans *8emes Journées Internationales "Le Génie Logiciel et ses Applications"*, Paris - La Défense, Novembre 1995.

[GJ91] N. Gehani et H.V. Jagadish. Ode as an Active Database: Constraints and Triggers. Dans *Actes de la 17eme Conférence Internationale VLDB*, pages 327–336, Barcelone - Espagne, Septembre 1991.

[GJS92] N. Gehani, H.V. Jagadish et O. Shmueli. Event Specification in an Active Object-Oriented Database. Dans *ACM SIGMOD International Conference on Management of Data*, pages 81–90, San Diego - USA, 1992.

[GKPC85] F. Giannesini, H. Kanoui, R. Pasero et M. Van Caneghem. *PROLOG.* InterEditions, 1985.

[GMSB96] M.C. Gaudel, B. Marre, F. Schlienger et G. Bernot. *Précis de Génie Logiciel.* Masson, 1996.

[GQVG94] A. Gonzales-Quel, M. Villegas et S. Gonzales. IDEA Technology Assessment based on Workflow Applications. The Royal Life Application. Rapport IDEA.DE.21.S.001.01, Projet Esprit IDEA - Sema Group, Madrid, Espagne, Novembre 1994.

[GR83] A. Goldberg et D. Robson. *SmallTalk-80 : the language and its implementation.* Addison-Wesley Publishing Company, 1983.

[Han92] E. Hanson. Rule Condition Testing and Action Execution in Ariel. Dans *ACM-SIGMOD*, pages 281–290, Juin 1992.

[Har87] D. Harel. Statecharts : A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, Juin 1987.

[HSE90] B. Henderson-Sellers et J.M. Edwards. The object-oriented life cycles. *Communications of the ACM*, pages 142–159, Septembre 1990.

[JCJO92] I. Jacobson, M. Christerson, P. Johnsson et G. Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, 1992.

[Joh92] R.E. Johnson. Documenting Frameworks using Patterns. Dans Andreas PAEPCK, éditeur, *OOPSLA'92*, October 1992.

[Kar95] Even-André Karlsson, éditeur. *Software Reuse: A Holistic Approach.* John Wiley & Sons - Wiley series in Software-Based Systems, 1995.

[Lai97] M. Lai. *UML - La notation unifiée de modélisation objet - Applications en Java.* InterEditions, 1997.

[Lal90] R. Lalement. *Logique, Réduction, Résolution.* Masson, 1990.

[LCD97] O. Lobry, C. Collet et P. Dechamboux. The Virtuose Distributed Object Store. Dans *Actes des Journées O2 Technology - 13eme Congrès BDA*, Grenoble, Septembre 1997.

[Lis81] B.H. Liskov. *CLU Reference Manual*, ANSI/MIS-STD 1815 édition, 1981.

[Lop96] M. Lopez. IDEA Final Report. Rapport, Projet Esprit IDEA (EP6333), Grenoble, Juin 1996.

[Mad96] M. Madjid. Conception de BD actives à travers le paradigme d'objet actif. Dans *Actes des 3emes Journées des Jeunes Chercheurs en Bases de Données*, Paris, Novembre 1996. GDR-PRC Bases de Données.

[MNM87] B. Meyer, J-M. Nelson et M. Matsuo. Eiffel : object-oriented design for software engineering. Dans *Proceedings of ESOP'87 European Software Engineering Conference*, pages 237–245. AFCET, 1987.

[MO96] J. Martin et J.J. Odell. *Object Oriented Methods : Pragmatic Considerations*. Prentice Hall - Englewood Cliffs, New Jersey, USA, 1996.

[Moi77] J.L. Le Moigne. *La théorie du Système Général*. PUF, Paris, 1977.

[Mon94a] G. Monteleone. A Command and Control System for a Military Application. Rapport IDEA.WP.21T.004, Projet Esprit IDEA - TXT Ingegneria Informatica, Novembre 1994.

[Mon94b] G. Monteleone. Energy Management Applications : First Description. Rapport IDEA.DE.21T.002, Projet Esprit IDEA - TXT Ingegneria Informatica, Novembre 1994.

[Mon96] L. Mondemé. Informatique de gestion : les sept couches de la réutilisation. *Revue Génie Logiciel*, 42:40–44, Décembre 1996.

[Mor96] J-M. Morel. Expériences de réutilisation avec la méthode REBOOT. *Revue Génie Logiciel*, 42:45–50, Décembre 1996.

[MPP96] P. Maret, L. Poullet et J.M. Pinon. Des modèles conceptuels pour capitaliser la connaissance au sein d'une organisation. *Ingénierie des systèmes d'information*, 4(4):491–540, 1996.

[Mul97] P-A. Muller. *Modélisation Objet avec UML*. Eyrolles, 1997.

[NECH92] D. Nanci, B. Espinasse, B. Cohen et H. Heckenroth. *Ingénierie des Systèmes d'Information avec Merise - Vers une Deuxième Génération*. Sybex, 1992.

[Ner92] J.M. Nerson. Applying Object-Oriented Analysis and Design. *Communications of the ACM*, 35(9):63–74, Septembre 1992.

[Ode93a] J. Odell. Specifiyng Requirements Using Rules. *Journal of Object-Oriented Programming*, 6(2):20–24, Mai 1993.

[Ode93b] J. Odell. Using Business Rules with Diagrams. *Journal of Object-Oriented Programming*, 6(4):10–16, Juillet/Août 1993.

[Ous97] M. Oussalah, éditeur. *Ingénierie Objet - Concepts, techniques et méthodes*. InterEditions, Mai 1997.

[Pag94] D. Pagonis. *Construire un système d'information hospitalier intégré*. Thèse de Doctorat, Université Joseph Fourier, 1994.

[Par93] Parallax Sotware Technologies. *GraphTalk 2.5 - Métamodélisation - Manuel de Référence et Interface de Programmation*, Décembre 1993.

[Par94] Parallax Sotware Technologies. *LEdit - Interface de programmation - Version 1.20*, Avril 1994.

[Pig96] Y. Pigneur. A Framework for New Information Systems. Dans F. Bodart & al., éditeur, *The Future of Information Systems: Challenges and Pitfalls*, pages 61–102, Namur, Belgique, Octobre 1996.

[Pre94] W. Pree. *Design patterns for Object-Oriented Development.* Addison Wesley, 1994.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy et W. Lorensen. *Object-Oriented Modeling and Design.* Yourdon Press, Prentice-Hall, 1991.

[RdZ94] A. Rogister et I. de Zegher. Preliminary Study on OPADE reengineering using IDEA methodology and concepts. Rapport IDEA.WP.21.M.001.1, Projet Esprit IDEA - BIM, Novembre 1994.

[RFB88] C. Rolland, O. Foucault et G. Benci. *Conception des systèmes d'information - La méthode REMORA.* Eyrolles, 1988.

[Rob93] P. Robert. *Le nouveau Petit Robert, Dictionnaire de la Langue Française.* Dictionnaires Le Robert, Paris, 1993.

[Rol93] C. Rolland. Adapter les Méthodes à l'Objet : Challenges et Embûches. Dans *Journées Méthodes d'Analyse et de Conception Orientées Objet des Systèmes d'Information*, Paris, Novembre 1993. AFCET.

[Rol97] C. Rolland. A Primer for Method Engineering. Dans Association INFORSID, éditeur, *Actes du congrès INFORSID'97*, Toulouse, Juin 1997. ISBN : 2-906-855-13-8.

[Ron97] C. Roncancio. Toward duration-based, constrained and dynamic event types. Dans *Proceedings of ARTDB'97*, Septembre 1997.

[Roy70] W.W. Royce. Managing the Development of Large Systems : concepts and techniques. Dans *WESCON*, volume 14, pages 1–9, Août 1970.

[RTBG97] D. Rieu, M. Tollenaere, F. Bounaas et J-P. Giraudin. Patrons d'objets pour les SGDT : le projet POSEIDON. Dans *2eme Congrès International*

Franco-Québecois - Le Génie Industriel dans un Monde sans Frontières, Albi, Septembre 1997.

[SHP89] M. Stonebraker, M. Hearst et S. Potomianos. A Commentary on the POSTGRES Rules System. *SIGMOD-RECORD*, 18(3):5–11, Septembre 1989.

[SKdM92] E. Simon, J. Kiernan et C. de Maindreville. Supporting Deductive and Active Rules on Top of a Relationnal DBMS. Dans In Yuan, éditeur, *Actes de la 18eme Conférence Internationale VLDB*, pages 315–326, Vancouver, Canada, Août 1992.

[SS91] R. Spencer-Smith. *Logic and Prolog.* Harvester Wheatsheaf, 1991.

[STO97] Equipe STORM. Serveur d'Objets Multimédias Actifs Temporels. Rapport d'activité - Equipe STORM - Laboratoire LSR-IMAG, Juillet 1997. http://www-lsr.imag.fr/storm.html.

[Tea94] GOODSTEP Team. The GOODSTEP Project : General Object-Oriented Database for Software Engineering Processes. Dans *Proceedings of the Asia-Pacific Software Engineering Conference*, pages 10–19, Tokyo, Japan, 1994. IEEE Computer Society Press.

[Tei94] M. Teisseire. *Le Modèle IFO2 : de la modélisation comportementale à la dérivation.* Thèse de Doctorat, Université Aix-Marseille II, 1994.

[TRC83] H. Tardieu, A. Rochfeld et R. Coletti. *La Méthode Merise : tome 1 : principes et outils.* Editions d'organisation, Paris, 1983.

[TSI85] TSI. Numéro Spécial Réseaux de Petri. *Techniques et Sciences Informatiques*, 4(1), Janvier/Février 1985.

[VCK96] J. Vlissides, J. Coplien et N. Kerth, éditeurs. *Pattern Languages of Program Design 2.* Addison-Wesley, 1996.

[WF90] J. Widom et S. Finkelstein. Set-Oriented Production Rules in Relationnal Database Systems. Dans *ACM-SIGMOD, International Conference on Management of Data*, pages 259–270, Atlantic City - New Jersey - USA, Mai 1990. ACM-Press.

[Wir83] N. Wirth. *Programming in Modula-2.* Springer-Verlag, 1983.

Index

Annexe A

Un exemple de langage de patrons

Cette annexe présente un exemple de langage de patrons : le langage *Evénements Récurrents pour Calendriers* [Fow97] dont le but est de décrire la manière de détecter et de traiter des **événements récurrents** dans une application. Des exemples d'événements récurrents dans une application sont par exemple : *Tous les mardis et tous les jeudis, le chirurgien Dupont opère des patients*, ou encore *tous les premiers mercredis du mois à 12h00, la sirène des pompiers est testée.* Six patrons y sont introduits dans deux buts différents : aider le concepteur d'applications à appréhender et à modéliser le problème des événements récurrents et lui offrir des éléments de code (ici en Java [BFGLG96]) pour implanter une solution à ce problème.

A.1 Calendrier

Tout agent réagit généralement à plusieurs événements récurrents. Par exemple, *un chirurgien opère ses patients tous les mardis et tous les jeudis. Tous les mercredis et tous les vendredis, il consulte des patients externes. De plus, il effectue des visites auprès de ses patients encore hospitalisés tous les matins. Enfin, il rencontre ses collègues nationaux dans des réunions tous les premiers lundis du mois.*

Une première manière de représenter ces événements consiste à créer des associations entre le chirurgien et des instances d'une classe Date pour chaque événement (opération, visite, consultation, réunion, etc.). Cette solution implique de nombreuses associations (autant qu'il y a d'événements récurrents) et nécessite d'un point de vue implantation de changer l'interface d'un objet chaque fois qu'un nouvel événement récurrent apparaît. Martin Fowler propose pour pallier à ce problème le patron **Ca-**

lendrier (cf. figure A.1) composé de trois classes Personne, Calendrier et Date. Une personne possède un calendrier (représentant son emploi du temps), et un calendrier est associé à une ou plusieurs dates par le principe de classe associative introduit par exemple dans OMT [RBP+91]. La classe associative définit une nouvelle classe Evénement et caractérise le fait que tout calendrier comporte une date pour chaque instance d'un événement.

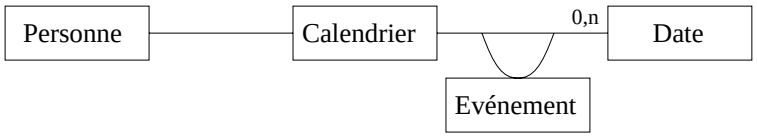


FIG. A.1 – *Le patron* Calendrier *[Fow97]*

A.2 Interface d'un calendrier

Le patron **Interface d'un calendrier** apporte des solutions pour déterminer les questions importantes auxquelles un calendrier doit pouvoir répondre. Il est ici important de se préoccuper non pas de la structure interne d'un calendrier, mais de ce que l'on souhaite que celui-ci fasse. Ainsi, Martin Fowler propose qu'un calendrier puisse par exemple répondre aux questions suivantes : *quels jours le chirurgien opère-t-il ce mois-ci ?* (la réponse serait alors un ensemble de dates comprises entre le début et la fin de ce mois), *quelle est la date de sa prochaine réunion ?* ou encore *le chirurgien a-t-il quelque chose de prévu le lundi 12 octobre ?*. Un exemple de code est proposé pour implanter les méthodes introduites pour répondre aux trois questions précédentes. La signature de ces méthodes est donnée dans la suite en utilisant le langage de programmation Java :

```
class Calendrier {
    public boolean ALieu(String evtArg, Date uneDate);
    public Vector dates(String evtArg, PeriodeDate pendant);
    public Date ProchaineOccurrence(String evtArg, Date uneDate); } ;
```

A.3 Eléments d'un calendrier

L'emploi du temps d'une personne contient en général plusieurs événements récurrents. Un calendrier comporte donc plusieurs éléments, dont chacun associe un événement à une expression temporelle qui détermine les dates appropriées (cf. figure A.2). Ainsi, le fait que le chirurgien ait des opérations les mardis et jeudis et des réunions tous les premiers lundis du mois est représenté par deux éléments du calendrier : l'un associé à l'événement Opération et à l'expression temporelle tous les mardis et tous les jeudis, l'autre associé à l'événement Réunion et à l'expression temporelle Tous les premiers lundis du mois.

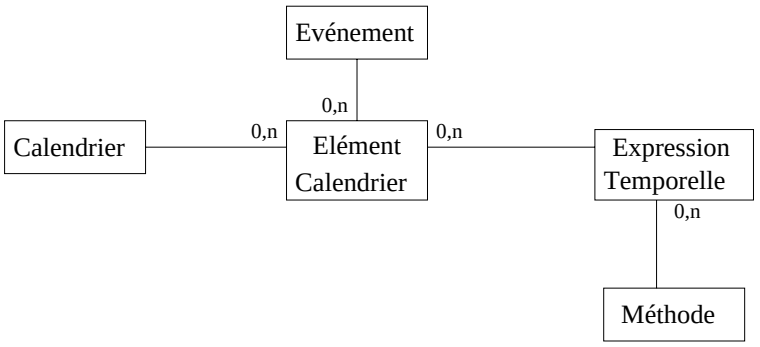


FIG. A.2 – *Le patron* Eléments du Calendrier *[Fow97]*

Le comportement dynamique du système engendré par l'appel de la méthode ALieu définie grâce au patron Interface du calendrier est représenté par le diagramme d'interaction donné dans la figure A.3. La méthode ALieu est invoquée avec deux paramètres : un événement (Opération) et une date (1er avril) pour déterminer si un événement du type Opération a lieu le 1er avril. Le traitement de cette méthode se déroule comme suit.

- Le calendrier recevant l'appel de la méthode ALieu délègue le message à ses éléments.
- Chaque élément vérifie l'événement passé en paramètre (l'opération) et demande l'expression temporelle qui lui correspond.
- L'expression temporelle appelle sa méthode inclut(date) qui détermine si la date passée en paramètre est incluse dans l'expression temporelle (par exemple, le 1er avril est-il inclus dans l'expression temporelle : Tous les mardis et jeudis de l'année en cours ?).

– Si l'expression temporelle inclut la date passée en paramètre, alors l'élément répond vrai au calendrier.

– Si l'un des éléments répond vrai, alors le calendrier répond vrai ; sinon, il répond faux.

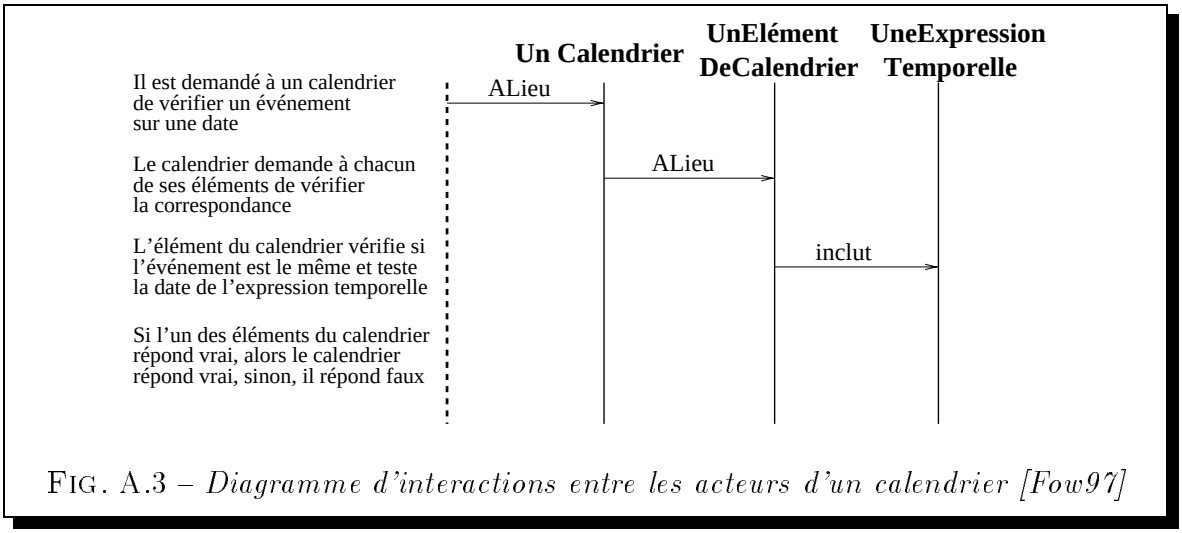


FIG. A.3 – *Diagramme d'interactions entre les acteurs d'un calendrier [Fow97]*

De plus, le patron Eléments d'un calendrier propose une implantation de l'opération ALieu :

```
class Calendrier {
    public boolean ALieu(String evtArg, Date uneDate) {
        ElementCalendrier chaqueEC;
        Enumeration e = elements.elements();
        while (e.aEncoreElements()) {
            chaqueEC = (ElementCalendrier)e.prochainElement();
            if (chaqueEC.ALieu(evtArg, uneDate))
              return true; }
        return false; } ;


class ElementCalendrier {
    public boolean ALieu(String evtArg, Date uneDate) {
        if (evt == evtArg)
          return ExpressionTemporelle.inclut(uneDate);
        else return false; } ;
```

A ce niveau du langage, l'événement est créé et associé à une expression temporelle. Reste maintenant à former l'expression temporelle. L'auteur propose quelques types d'expressions telles que "Un Jour Tous les Mois" ou "Une Période d'Année". Les classes implantant ces expressions temporelles sont codées et doivent être paramétrables afin de couvrir le plus de problèmes possibles.

A.4 Un Jour Tous les Mois

Le patron Un jour Tous les Mois permet de représenter des événements tels que *le premier lundi du mois*. Une telle phrase comporte deux variables : d'une part le jour de la semaine, d'autre part sa position dans le mois. La classe Expression Temporelle se spécialise donc en une classe UnJourTousLesMois possèdant deux attributs Jour_Semaine et Compteur (cf. figure A.4).

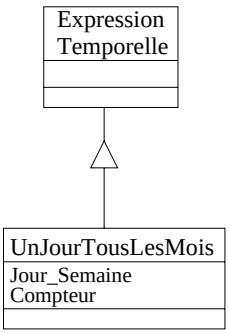


FIG. A.4 – *Expression temporelle* Un Jour Tous les Mois *[Fow97]*

Le code proposé pour implanter cette classe est le suivant :

```
abstract class ExpressionTemporelle {
        public abstract boolean inclut (Date laDate); } ;

class JourDansMois extends ExpressionTemporelle {
        private int compteur;
        private int IndexJour;
        public  JourDansMois (int IndexJour, int compteur) {
                this.IndexJour = IndexJour;
                this.compteur = compteur; } ;
```

```
public  boolean inclut (Date uneDate) {
        return JourCorrespond (uneDate) && SemaineCorrespond(uneDate); } ;

private boolean JourCorrespond (Date uneDate) {
        return uneDate.ObtenirJours() == IndexJour; } ;

private boolean SemaineCorrespond (Date uneDate) {
        if (compteur > 0)
          return DebutSemaineCorrespond(uneDate);
        else
          return FinSemaineCorrespond(uneDate); } ;

private boolean DebutSemaineCorrespond(Date uneDate) {
        int JoursJusqueFinMois = JourRestantsDansMois(uneDate) + 1;
        return SemaineDansMois(JoursJusqueFinMois) == Math.abs(compteur); } ;

private int SemaineDansMois(int NombreJour) {
        return ((NombreJour - 1) /7) +1; }
} ;
```

La classe Java Date utilise les chiffres 0 à 6 pour coder les jours de la semaine de dimanche à samedi. Ainsi, *le chirurgien a une réunion tous les premiers lundis du mois* sera représenté par une expression temporelle avec comme jour de la semaine le lundi (codé 1 dans la classe Date de Java) et comme compteur de semaine dans le mois 1 (*1er lundi du mois*). L'expression temporelle correspondante sera donc : *JourDansMois(1,1)*.

A.5 Périodes d'Année

Dans le même contexte que précédemment, il est parfois nécessaire de représenter des événements qui apparaissent périodiquement chaque année. L'auteur propose pour cela d'utiliser un autre sous-type d'expression temporelle : Périodes d'Année. Une période d'année est délimitée par un jour de départ et un jour de fin (cf. figure A.5). Le lecteur intéressé par le code proposé pour implanter cette classe pourra se référer à [Fow97].

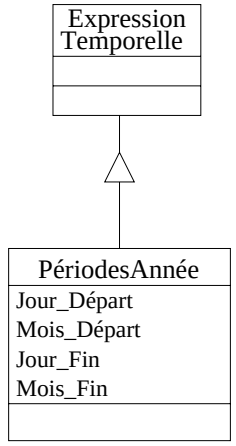


FIG. A.5 – *Expression temporelle* Périodes d'Année *[Fow97]*

A.6 Expressions Ensemblistes

Les expressions temporelles précédemment introduites représentent des événements récurrents. Cependant une personne, par exemple notre chirurgien, est souvent confrontée à plusieurs événements récurrents. Le patron Expressions ensemblistes propose une manière de représenter des expressions ensemblistes en introduisant les trois classes union, intersection et différence. Le code proposé pour implanter ces classes est donné dans [Fow97].

Annexe B

Le système de gestion de bases de données actif NAOS

Un système de gestion de bases de données **actif** est capable de détecter des situations ou des faits pertinents et de réagir sans intervention de l'utilisateur en exécutant une action si une condition est vérifiée. Dans ce but, le système NAOS (O_2 Native Active Object System) [Col97] [CC96] permet de définir des règles actives, puis d'exécuter ces règles dans le cadre d'applications O_2. Il est développé depuis septembre 1992 au sein de l'équipe STORM [STO97] du laboratoire LSR-IMAG (Université de Grenoble) et a été en grande partie spécifié et implanté dans le cadre du projet Esprit III GOODSTEP [Tea94].

NAOS étend le SGBD O_2 [AC93] avec des règles actives de la forme **Evénement-Condition-Action**. Les règles appartiennent au schéma de la base de données et sont définies au même niveau que les classes ou les applications (cf. figure B.1).

B.1 Un exemple de règle active NAOS

Considérons l'exemple de schéma O_2 donné dans la figure B.2 pour illustrer l'utilisation d'une règle. La classe Acte représente des actes médicaux ayant un numéro, un type, une spécialité, une date d'exécution et un résultat global stocké sous forme de caractères. La classe Entite représente des entités (patients ou prélèvements) sur lesquelles on effectue des actes. Une entité est caractérisée par un numéro et par l'ensemble des actes effectués. L'objet nommé LesEntites représente l'ensemble des entités gérées par l'hôpital et est stocké dans la base de données. La règle ResultatHauteGravite de la

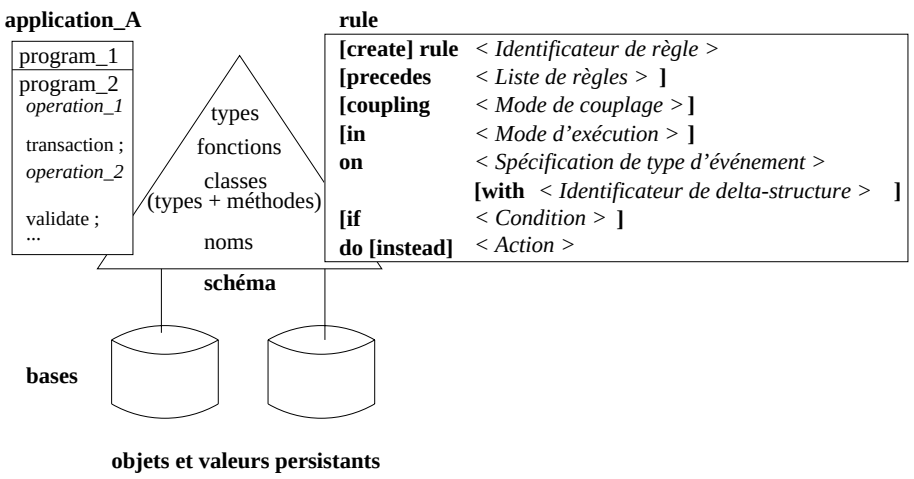


FIG. B.1 – *Intégration des règles NAOS dans le SGBD O_2*

figure B.3 est définie sur ce schéma pour avertir le médecin lorsque les résultats d'un acte sont de haute gravité.

```
class Acte public type
        tuple (numacte: string,
               typeacte: CatalogueActe,
               spécialite: string,
               date: Date,
               resGlobal: string) ;

class Entite public type
        tuple (numero: string,
               lesactes: set(Acte)) ;

name LesEntites : set(Entite);
```

FIG. B.2 – *Exemple de schéma O_2*

B.2 Structure générale de définition d'une règle

La signification de chacun des composants de la structure générale de définition en NAOS d'une règle active donnée dans la figure B.1 est la suivante :

```
rule ResultatHauteGravite
coupling immediate
on after update Entite→lesactes with e
if (exists a in (e→lesactes - old(e→lesactes))
   : a→resGlobal = "Haute gravité")
do { display ("Attention : Résultats alarmants"); }
```

FIG. B.3 – *Une règle active NAOS*

Identificateur de règle - Chaque règle est identifiée par un nom de règle unique dans le schéma auquel elle appartient (ex : ResultatHauteGravite).

Liste de règles - Il est possible d'affecter des priorités relatives aux règles au moyen d'une relation de précédence, ceci afin de résoudre des conflits dûs à l'exécution simultanée de plusieurs règles en réaction au même événement. Si des conflits subsistent, les règles sont exécutées selon leur ordre de définition. Dans l'exemple B.3, aucune relation de précédence n'est spécifiée.

Mode de couplage - Dans NAOS, les parties Condition et Action d'une règle s'exécutent dans la transaction qui a produit l'événement déclenchant (appelée transaction déclenchante). Si la condition est vraie, l'action est donc exécutée dans la transaction déclenchante. Le mode de couplage précise le moment où les parties Condition et Action de la règle s'exécutent par rapport à la transaction déclenchante. Il existe deux types de règles dans NAOS : les règles **immédiates** (mode de couplage immediate) et les règles **différées** (mode de couplage deferred). La condition d'une règle immédiate est évaluée juste après la détection de son événement déclenchant ; si la condition est vraie, l'action est alors exécutée. La condition d'une règle différée est évaluée après la dernière opération de la transaction déclenchante, mais avant sa validation. Les règles immédiates ont un mode de traitement des événements par instance. Les règles différées ont un mode de traitement ensembliste : elles accumulent les événements déclenchants au cours de la transaction déclenchante et traitent l'ensemble des événements en même temps. Enfin, toutes les règles consomment les événements. Dans l'exemple, la règle ResultatHauteGravite est immédiate.

Mode d'exécution - Le mode d'exécution d'une règle immédiate spécifie si la règle doit être déclenchée par un événement qui apparaît dans une transaction en lecture seule (in r-trans), en lecture-écriture (in rw-trans) ou en écriture seule (in w-trans). Le mode d'exécution d'une règle différée est systématiquement lecture-écriture et peut donc être omis.

Type d'événement - Un type d'événement décrit une situation particulière qui peut être reconnue par le système. Un événement peut être primitif ou composite. Un événement composite est construit récursivement à partir d'événements primitifs ou composites connectés par des opérateurs de conjonction, disjonction, disjonction exclusive, négation, séquence et séquence stricte. Il existe cinq types d'événements primitifs, dont les trois premiers sont associés à un instant de production de l'événement qui peut être before ou after (l'événement sera produit respectivement avant ou après l'occurrence du type d'événement concerné) :

- *les événements de manipulation d'entités* : création et destruction d'objets, modification d'objets ou de valeurs, insertion et suppression dans des collections, appel de méthodes ;
- *les événements transactionnels* : début, fin, validation ou annulation d'une transaction ;
- *les événements applicatifs* : début ou fin de l'exécution d'une application ;
- *les événements utilisateurs* déclenchés explicitement par l'utilisateur ;
- *les événements temporels* liés au temps, absolus (ex : *le 1er janvier 2000 à 0h00*), relatifs (ex : *3 heures après une prise de sang*) ou périodiques (ex : *tous les 3 mois*).

Dans l'exemple, la règle ResultatHauteGravite réagit à un événement de manipulation d'entité : elle est exécutée après chaque mise à jour de l'attribut lesactes d'un objet de la classe Entite.

Identificateur de delta-structure - Les données relatives aux événements sont stockées dans des *delta-structures* qui représentent l'environnement de déclenchement d'un événement, lequel peut être utile pour spécifier les parties Condition et Action de la règle. La delta-structure d'une règle immédiate est un élément réfèrant soit l'entité

concernée par l'opération produisant l'événement, soit la valeur composante insérée, supprimée ou mise à jour ou les paramètres effectifs et/ou le résultat d'un appel de méthode ou de programme : dans notre exemple, la delta-structure est l'objet de la classe Entite dont l'attribut lesactes est modifié : on appelle cet objet e. La delta-structure d'une règle différée est une collection qui reflète les changements survenus sur un ensemble d'entités.

Condition - La condition est une formule composée de prédicats sur les objets et les valeurs, donc sur l'état de la base de données. Elle est exprimée à l'aide du langage de requêtes OQL et est vraie si la requête correspondante est non vide. Dans ce cas, la partie Action est exécutée, sinon l'exécution de la règle est terminée. Dans notre exemple, la condition est vraie si l'un des actes ajouté dans l'attribut lesactes d'une entité a pour résultat global *haute gravité*.

Action - L'action est un bloc d'instructions écrites en O_2C qui peut référencer les données du contexte de l'événement ainsi que le résultat de la condition. En général, les actions sont des messages aux objets, des opérations de la base de données ou des appels de procédure. Dans notre exemple, la partie Action de la règle ResultatHauteGravite consiste en l'affichage d'un message d'alerte.

Clause instead - L'une des utilisations possibles d'une règle est destinée à l'annulation de l'opération déclenchante, ce qui n'a de sens que lorsque la règle est immédiate et l'instant de production de l'événement de type before. Quand une règle comprenant une clause instead est déclenchée, l'opération déclenchante est annulée et les actions de la clause instead sont exécutées. Ce n'est pas le cas pour notre règle ResultatHauteGravite où l'affichage du message est exécuté en plus de l'insertion du nouveau résultat.

Exécution des règles - L'exécution d'un ensemble de règles est gérée par le **modèle d'exécution** d'un système de règles qui spécifie quand et comment sont exécutées des règles actives déclenchées par des événements produits par l'exécution d'une transaction. Le modèle d'exécution de NAOS est composé de plusieurs dimensions [CC95], dont le mode de couplage, l'exécution de règles multiples et la delta-structure déjà traitées. Les autres dimensions sont données ci-dessous.

- **Exécution de règles en cascade** : lors de l'exécution de la partie Action d'une règle, une opération peut déclencher d'autres règles (ou elle-même). On parle

alors de règles en cascade. L'exécution des règles est basée sur la notion de cycles d'exécution décrivant l'exécution d'une séquence d'opérations. Dans NAOS, les règles immédiates sont exécutées en profondeur d'abord, les règles différées en largeur d'abord.

– **Effet net** : l'effet net d'une séquence d'opérations est le bilan de ces opérations qui perdure à la fin de cette séquence. Par exemple, si une séquence d'opérations crée deux entités `a` et `b`, puis détruit `a`, l'effet net sera la création de `b` pour le déclenchement et la construction des environnements d'exécution. Dans NAOS, toutes les règles prennent en compte l'effet net.

Annexe C

Modèle objet et classes O_2 d'un extrait du SIMT

C.1 Modèle statique de l'application SIMT

La figure C.1 donne le modèle statique de l'application médico-technique simplifiée.

C.2 Classes et racines de persistance O_2 de l'application SIMT simplifiée

```
class Personne public type
                tuple (NumSS : integer,
                       Nom: string,
                       Prenom: string),
                       Sexe: char) ;

class Medecin inherit Personne public
      methods PrescrireTraitement(Patient, Traitement) ;
              EndormirPatient(Patient) ;
              DemanderActe(Patient, Acte) ;

class Entite public type
              tuple (NumEntite: integer) ;
```

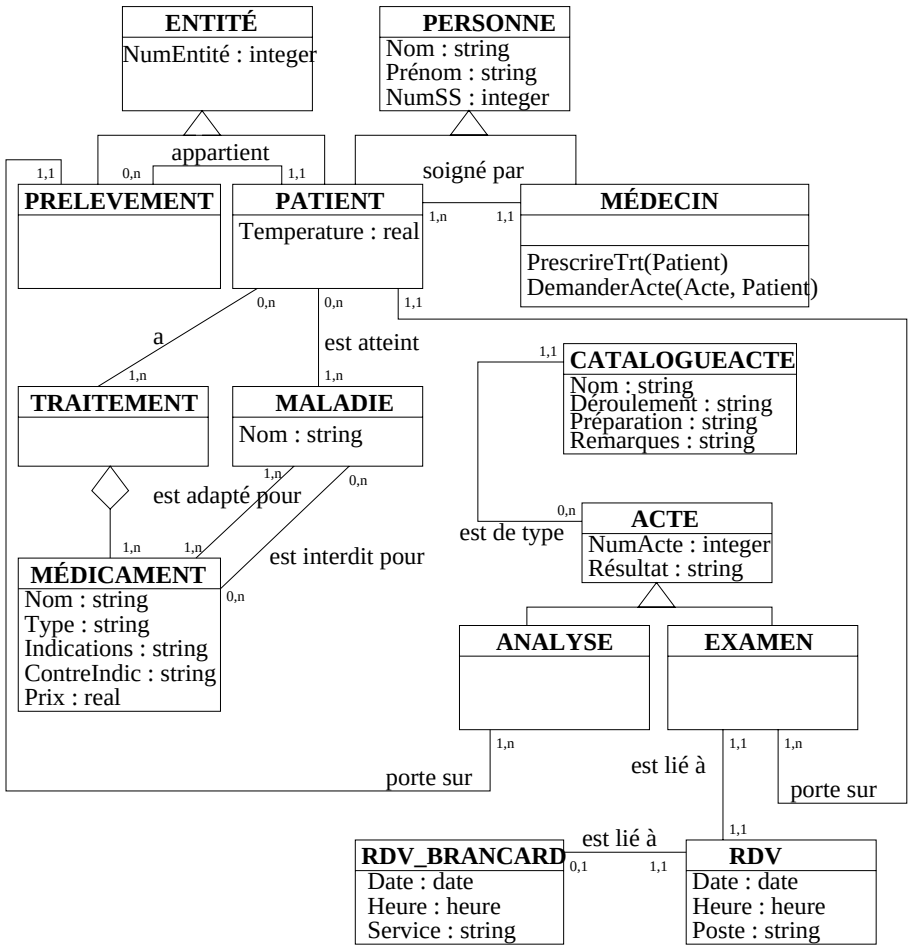


FIG. C.1 – *Modèle statique de l'application SIMT simplifiée*

class Patient **inherit** Personne, Entite **public** type
tuple (Temperature: **real**,
MedecinTraitant : Medecin,
MaladiesContractees: **set**(Maladie),
TraitementSuivi: **set**(Medicament));

class Prelevement **inherit** Entite **public** type
tuple (Proprietaire: Patient);

class Maladie **public** type
tuple (Nom: **string**,
MedicamentsAdaptés: **set**(Medicament),

MedicamentsInterdits: **set**(Medicament)) ;

class Medicament **public** type
tuple (Nom: **string**,
Type: **string**,
Prix: **real**,
Indications: **string**,
Contre-Indications: **string**) ;

CatalogueActes est la classe recensant tous les types d'actes possibles,
(par exemple, "AnalyseUrine", "AnalyseSang", "Echographie", etc.)
avec des informations particulières à chaque type d'acte.

class CatalogueActes **public** type
tuple (Nom: **string**,
Type: **string**, ("Analyse" ou "Examen")
Preparation: **string**,
Deroulement: **string**,
Remarques: **string**);

Acte est la classe recensant les actes effectivement effectués sur les patients
de l'hôpital ; chaque acte possède un attribut Type *qui donne*
le type de l'acte dans le catalogue des actes.

class Acte **public** type
tuple (NumActe: **integer**,
Type: CatalogueActe,
EntiteConcernee: Entite,
Prescripteur: Medecin,
Resultat: **string**) ;

class Examen **inherit** Acte **public** type
tuple (EntiteConcernee: Patient,
RdvExamen: Rdv) ;

class Analyse **inherit** Acte **public** type
tuple (EntiteConcernee: Prelevement) ;

class Rdv **public** type

tuple (DateRdv: Date,
HeureRdv: string,
ActePrescrit: Acte,
Poste: **string**,
Praticien: Medecin,
Patient: Patient) ;

class RdvBrancard **public** type
tuple (DateRdv: date,
HeureRdv: string,
Service: **string**) ;

Racines de persistance ou collections persistantes :

name LesPatients : **set**(Patient);
name LesPrelevements : **set**(Prelevement);
name LesMedecins : **set**(Medecin);
name LesMedecinsPresents : **set**(Medecin);
name LesMaladies : **set**(Maladie);
name LesMedicaments : **set**(Medicament);
name LesAnalyses : **set**(Analyse);
name LesExamens : **set**(Examen);
name LeCatalogueDesActes : **set**(CatalogueActe);
name LesRdv : **set**(Rdv);
name LesRdvAujourdhui : **set**(Rdv);
name LesRdvBrancards : **set**(RdvBrancard);

Annexe D

Les méta-outils GraphTalk et LEdit

Cette annexe présente les méta-outils GraphTalk et LEdit utilisés pour développer le prototype SCalP. Il ne s'agit que d'une introduction succinte à ces méta-outils complétée par un exemple d'utilisation dans le cadre d'éléments de modèles de la méthode OMT [RBP+91].

D.1 GraphTalk

GraphTalk [Par93] est un environnement objet de développement d'Ateliers de Génie Logiciel (AGL). Fondé sur un modèle à objets, il est destiné à aider au développement, à la production et à l'utilisation d'AGL et est plus particulièrement dédié aux méthodes d'analyse et de conception. Essentiellement articulé autour d'un générateur d'éditeurs graphiques, il permet surtout la construction d'outils de modélisation graphique relatifs à toutes les méthodes traditionnelles de conception de systèmes d'information. De nombreux AGLs ont ainsi été développés avec GraphTalk, mettant en œuvre les méthodes les plus utilisées comme OMT, OOA, OOD, Merise ou encore Fusion. Ces AGLs sont utilisés par un modélisateur pour représenter un système d'information avec la méthode choisie. GraphTalk peut cependant aussi être utilisé pour spécifier et implanter des ateliers ad'hoc destinés à des équipes de projets. Son utilisation se fait donc à deux niveaux : le niveau méta-modélisation où un AGL propre à une méthode est créé et le niveau modélisation où l'AGL est utilisé pour construire un modèle d'un système d'information selon la méthode choisie.

Créer un AGL sous GraphTalk consiste à concevoir un métamodèle des modèles de la

méthode choisie dans le métamodèle de GraphTalk et se déroule en deux étapes :

- la conception graphique d'un méta-modèle en utilisant l'interface graphique,
- l'enrichissement du méta-modèle grâce à l'interface de programmation.

D.1.1 Construction d'un AGL

D.1.1.1 Première étape : conception graphique

Le but de cette étape est de décrire chaque concept du modèle à l'aide des concepts proposés par GraphTalk en suivant quatre phases (cf. fenêtre *GraphTalk - OMT* de la figure D.1).

- **Spécification sémantique** : on décrit les concepts d'une méthode ainsi que les relations entre ces concepts.
- **Affectation des propriétés** : on attache un ensemble de propriétés aux concepts et relations définies dans la phase précédente. Les propriétés sont spécifiques à chaque type de concept et chaque propriété associée à un concept sera l'un de ses attributs lors de l'utilisation de l'atelier généré.
- **Spécification des formes** : on spécifie les formes graphiques que devront avoir les concepts lors de l'utilisation de l'atelier généré.
- **Spécification des fenêtres** : on construit l'interface homme-machine de l'atelier grâce à des menus, des sous-menus, des actions et des requêtes.

La méta-modélisation est réalisée avec un langage graphique appelé méta-langage. Dans ce méta-langage sont prédéfinies trois méta-classes : Méta-Objet, Méta-Lien et Méta-Propriété.

Méta-classe Méta-Objet - La fenêtre *Spécification sémantique - S Modèle Objet* de la figure D.1 montre des concepts, instances de la méta-classe **Méta-Objet**, autour desquels le méta-modèle d'un extrait du modèle objet de la méthode OMT [RBP+91] est construit.

- La classe Objet sert à déclarer les concepts de la méthode : dans l'exemple, trois instances de la classe Objet sont définies : Classe, Attribut, Opération.

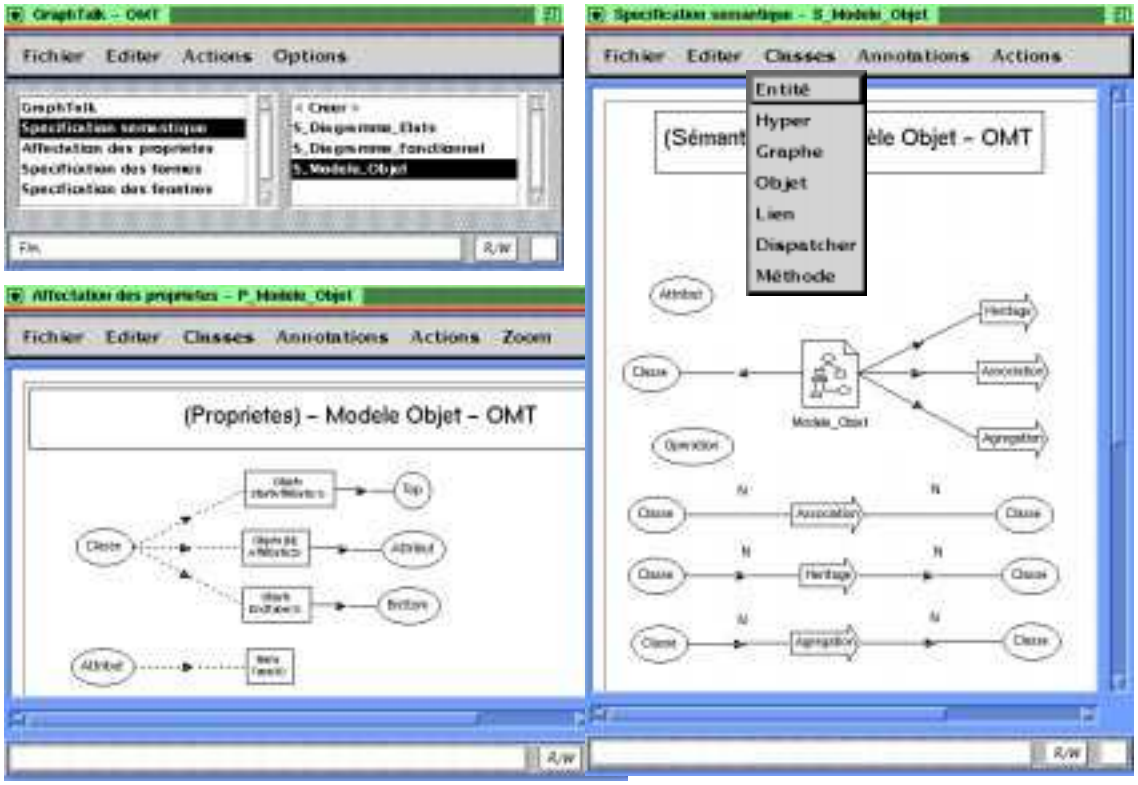


FIG. D.1 – *Méta-modélisation d'une méthode avec GraphTalk*

– La classe Lien permet de définir la sémantique des relations existant entre les concepts : dans l'exemple, trois instances de la classe Lien sont définies : Héritage, Agrégation, Association.

– La classe Graphe décrit tout ou partie d'un modèle d'une méthode en regroupant les concepts qui lui sont attachés : dans l'exemple, le graphe Modèle Objet regroupe les concepts présents dans un extrait du modèle objet de la méthode OMT : des classes et des relations d'héritage, d'agrégation et d'association.

– La classe Hypergraphe possède une seule instance composée de tous les graphes utilisés pour décrire une méthode. Dans l'exemple, l'hypergraphe (non visible) serait composé du graphe Modèle Objet et d'autres graphes utilisés pour décrire l'aspect dynamique et l'aspect fonctionnel de la méthode. Une instance de cet hypergraphe correspond à l'AGL.

– La classe Entité sert à factoriser des comportements communs à différents composants d'une modélisation.

- Les instances de la classe Méthode sont équivalentes aux méthodes traditionnelles orientées objets.

- La classe Dispatcher permet la création d'arbres comme des arbres d'héritage ou d'agrégation.

- La classe Propriété disponible pendant la phase d'affectation des propriétés (fenêtre *Affectation des Propriétés - P Modèle Objet*), regroupe des informations associées aux instances de la méta-classe Méta-Objets : dans l'exemple, une classe possède une propriété objets appelée attributs elle-même reliée à un attribut. Cette propriété signifie qu'une classe possède un ou plusieurs attributs, mais qu'un attribut n'a d'existence que par rapport à une classe. De plus, un attribut possède une propriété Menu appelée Type composée des éléments de menus suivants : string, character, integer, boolean, etc.

- La classe Forme disponible pendant la phase de spécification des formes, permet de donner des formes aux instances de la méta-classe Méta-Objet ; ces formes seront affectées automatiquement aux concepts de la méthode lors de l'utilisation de l'AGL : par exemple, une classe a une forme rectangulaire.

- Enfin, la classe Action disponible pendant la phase de spécification des fenêtres, permet de spécifier l'interface homme-machine propre à l'AGL. Elle peut être une action standard ou spécifique, un démon ou une requête et sera associée à un menu d'une des instances de la méta-classe Méta-Objet.

Méta-classe Méta-Lien - Elle permet de spécifier des relations entre les instances des classes de la méta-classe Méta-Objet : ainsi, il est possible de préciser des liens de *Composition* entre classes, d'*Héritage* entre classes, d'*Affectation* de propriétés (variables d'instances) à des classes, de *Connexion autorisée* entre classes d'objets au moyen de classes de liens, d'*Exclusion*, d'*Inclusion* ou d'*Héritage* entre classes de liens, etc. L'instanciation de cette méta-classe apparaît à divers endroits dans la figure D.1 : c'est grâce à elle que nous pouvons préciser par exemple qu'une classe appartient au graphe Modèle Objet ou encore qu'une classe est reliée à une autre par l'intermédiaire du lien orienté Héritage ou du lien non orienté Association.

Méta-classe Méta-Propriété - Elle permet d'instancier des propriétés spécifiques à chaque type de méta-objet créé : ces propriétés sont en fait des variables de classes des classes Objet, Lien, Propriété, Graphe, etc. et sont visibles au niveau instanciation lors

de l'utilisation de l'AGL. Par exemple, un objet a un nom et peut être défini comme abstrait ou non ; un lien possède des cardinalités ; enfin, les propriétés ont des valeurs par défaut.

D.1.1.2 Deuxième étape : enrichissement du méta-modèle

Le but de cette étape est d'enrichir le méta-modèle en utilisant l'interface de programmation fournie par GraphTalk. Cette interface permet de définir en Lisp, C ou C++ des fonctions implantant des fonctionnalités généralement non réalisables par la seule méta-modélisation. Les fonctionnalités ainsi définies sont disponibles pour le modélisateur. Les fonctions développées sont appelées de deux manières :

- par activation de démons, dont l'exécution est lancée automatiquement dès que la condition d'activation est satisfaite (ex : après création d'une instance de la classe, avant affichage de la valeur d'une propriété, etc.). ;
- par exécution d'actions programmées lancées à l'initiative de l'utilisateur à travers les menus.

Le méta-modèle est de plus enrichi par les fonctionnalités d'interrogation et de documentation offertes par le langage GQL. GQL (GraphTalk Query Langage) est un langage de programmation de type SQL permettant d'une part d'effectuer des accès aux informations contenues dans un hypergraphe, d'autre part de spécifier dans des masques de documentation comment accéder aux informations qui seront stockées dans le document à son exécution.

D.1.2 Utilisation de l'AGL

Après compilation, l'AGL défini est utilisable par le modélisateur pour modéliser un système d'information selon la méthode choisie. La figure D.2 montre un début de modélisation du système d'information médico-technique d'un hôpital en utilisant l'AGL précédemment défini, lequel permet l'utilisation de la méthode OMT.

La fenêtre *OMT - Modélisation OMT du SIMT* de la figure D.2 apparaît lors de l'appel de l'atelier. Elle offre la possibilité de créer un ou plusieurs graphes parmi Modèle Objet, Modèle Dynamique et Modèle Fonctionnel. Ces graphes correspondent aux trois modèles de la méthode OMT et sont trois instances de la méta-classe Graphe reliées à l'hypergraphe représentant l'AGL.

La fenêtre *Modèle Objet - Gestion des classes* est une instance du graphe Modèle Objet. Le menu Classes montre que ce niveau d'utilisation de l'AGL ne permet de créer que des instances de la classe Classe définie lors de la spécification sémantique du modèle objet de l'atelier. En effet, la classe Classe, par opposition aux classes Attribut et Opération, est rattachée au graphe et peut par conséquent être instanciée directement lors de l'utilisation de l'AGL. Dès lors, des instances de la classe Classe peuvent être définies, avec la forme rectangulaire conforme à la méthode OMT, par exemple Entité, Patient, Prélèvement et Acte. De plus, de même qu'il est possible de dire qu'une entité est associée à un ou plusieurs actes grâce à l'instance Association de la méta-classe Lien (les cardinalités étant définies comme des propriétés de cette association), il est possible de définir une relation d'héritage entre la classe Entité et les classes Patient et Prélèvement. Enfin, une classe possède la propriété Attributs composée d'un ensemble d'attributs, un attribut étant caractérisé par un Nom (propriété textuelle). Les fenêtres Objet - Patient et Prop Attribut montrent que l'attribut Prénom est ajouté à l'ensemble des attributs de la classe Patient.

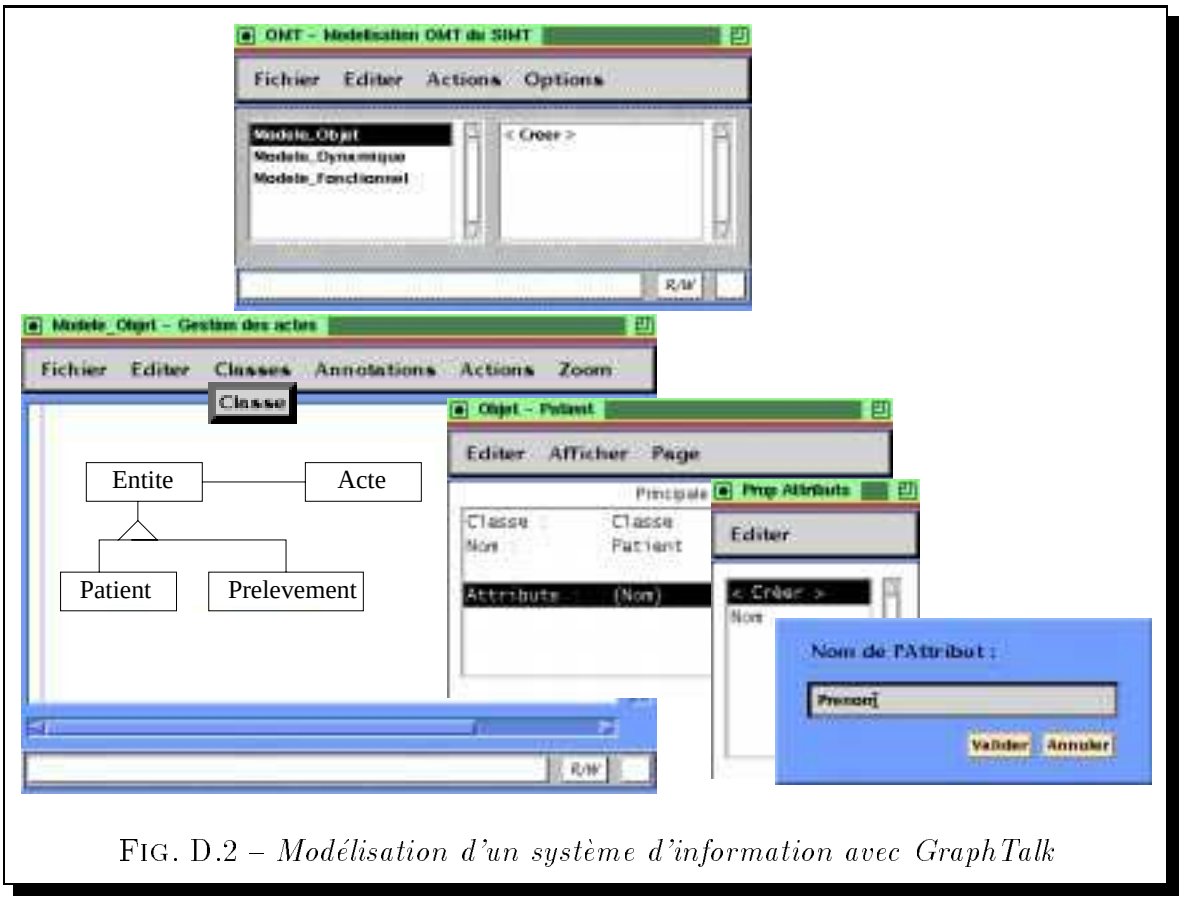


FIG. D.2 – *Modélisation d'un système d'information avec GraphTalk*

D.2 LEdit

Le logiciel LEdit [Par94] est un méta-éditeur permettant de spécifier les grammaires de langages existants (C++, SQL, etc.) ou non et de générer les éditeurs syntaxiques et lexicaux associés. La figure D.3 montre un exemple d'utilisation du méta-outil LEdit avec le langage C++ : la fenêtre ledit - cppc.le montre la définition en LEdit d'une partie de la grammaire du langage C++ ; la fenêtre cppc - untitled appartient à l'atelier généré par LEdit à partir de cette grammaire et permettant d'éditer et de déclarer des classes selon la syntaxe C++. L'annexe E donne la traduction en LEdit de la grammaire de NAOS.

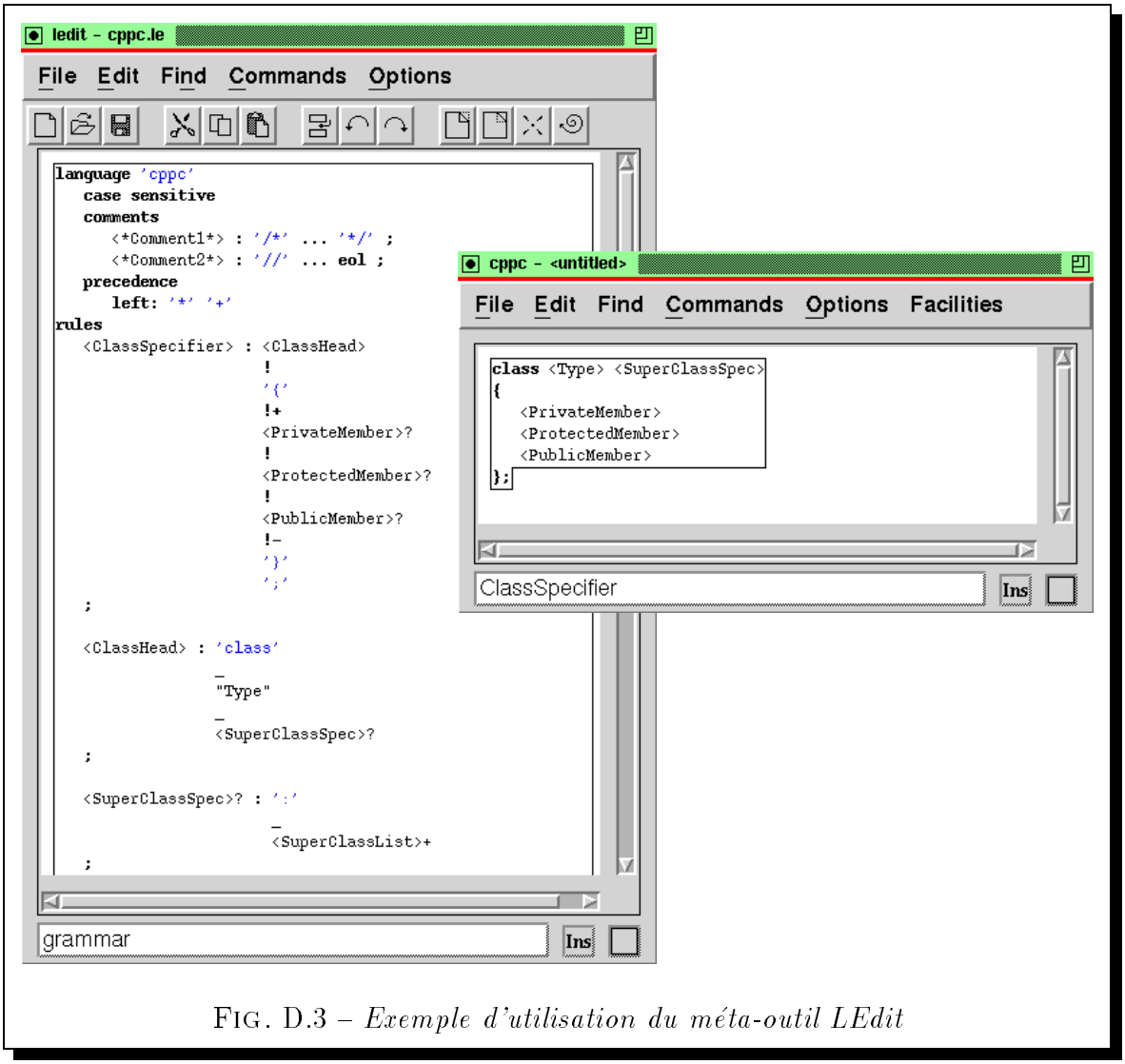


FIG. D.3 – *Exemple d'utilisation du méta-outil LEdit*

Dans LEdit, une grammaire est définie par une *forme concrète* (constituée par une

ensemble de *nœuds concrets*) et une *forme abstraite* (constituée par une ensemble de *nœuds abstraits*). Les nœuds abstraits sont structurés sous la forme d'un graphe, les nœuds concrets sous la forme d'un arbre. La définition d'une grammaire en LEdit consiste à l'implanter sous forme de graphe dans lequel sont définis des nœuds abstraits choisis parmi les types suivants (cf. fenêtre ledit - cppc.le, figure D.3) :

- **FixedArity**, nœud constitué par une suite fixe d'éléments, par exemple
  < ClassSpecifier > ;
- **Optionnal**, nœud constitué par une suite fixe d'éléments qui peuvent ne pas être instanciés, par exemple < SuperClassSpec >? ;
- **NaryPlus**, nœud constitué par une suite de longueur variable d'éléments devant contenir au moins un élément, par exemple < SuperClassList > + ;
- **NaryStar**, nœud constitué par une suite de longueur variable ou éventuellement nulle d'éléments,
- **Phylum**, nœud qui définit un élément à prendre dans un ensemble,
- **Terminal**, nœud qui représente un élément défini par une expression régulière lexicale, par exemple "Type".

Tout nœud abstrait de la grammaire dérive un nœud concret spécifique ajouté à l'arbre des nœuds concrets de la grammaire. L'interface de programmation de LEdit qui permet par exemple d'interroger le type des nœuds abstraits, d'affecter une valeur à un terminal ou de se déplacer dans l'arbre des nœuds concrets, permet de coupler un outil ayant une architecture ouverte (en particulier GraphTalk) avec un éditeur généré par LEdit. Ainsi, la génération de code à partir d'outils de spécification, la gestion de cohérence inverse ou la traduction de programmes sont des exemples d'applications réalisables rapidement.

Annexe E

Grammaire de NAOS en LEDIT

```
language 'NAOS'
case insensitive
comments
 <*Comment1*> : '/*' ... '*/' ;
 <*Comment2*> : '//' ... eol ;
precedence
 left: '*' '+'
rules
 <rules_set> : <rule_definition> ;

//DEFINITION D'UNE REGLE
 <rule_definition> : <init_rule> <rule_body>+ <end_rule> ;
 <init_rule> : <epsilon> ;
 <rule_body>+ : '//****************' ! <rule_def> ! '//****************'
                ! ... ! '//****************' ;
 <end_rule> : <epsilon> ;
 <rule_def> : [create] ! <precedes>? ! <coupling>? ! <exec_mode>?
              ! <on_event> ! <condition>? ! <action> ;

//[create] (DEFINITION D'UNE REGLE)
 [create] : <create_first>
            | <create_after> ;
 <create_first> : 'CREATE RULE' _ "rule_name" ;
 <create_after> : 'RULE' _ "rule_name" ;
```

```
//<precedes>? (DEFINITION D'UNE REGLE)
 <precedes>? : 'PRECEDES' _ <rule_list>+ ;
 <rule_list>+ : "rule_name" ',' _ ... ;

//<coupling>? (DEFINITION D'UNE REGLE)
 <coupling>? : 'COUPLING' _ '('[couplingEC]')' ;
 [couplingEC] : [coupling_mode]
               | <coupling_mode_EC> ;
 <coupling_mode_EC> : [coupling_mode] ',' _ [coupling_mode] ;
 [coupling_mode] : <immediate>
                  | <deferred> ;
 <immediate> : 'IMMEDIATE' ;
 <deferred> : 'DEFERRED' ;

//<exec_mode>? (DEFINITION D'UNE REGLE)
 <exec_mode>? : [exec_mode_r_w] ;
 [exec_mode_r_w] : <r_trans>
                  | <w_trans>
                  | <rw_trans> ;
 <r_trans> : 'IN r_trans' ;
 <w_trans> : 'IN w_trans' ;
 <rw_trans> : 'IN rw_trans' ;

//<on_event> (DEFINITION D'UNE REGLE)
 <on_event> : 'ON' _ [composite_event] ;
 [composite_event] : <comp>
                    | <comp_op_comp>
                    | <neg_comp>
                    | <prim> ;
 <comp> : '('[composite_event]')' ;
 <comp_op_comp> : [composite_event] _ [operator] _ [composite_event] ;
 [operator] : <conjonction>
              | <disjonction>
              | <exclusive_disjonction>
              | <sequence>
              | <strict_sequence> ;
```

```
 <conjonction> : '&&' ;
 <disjonction> : '||' ;
 <exclusive_disjonction> : '^' ;
 <sequence> : ',' ;
 <strict_sequence> : ';' ;
 <neg_comp> : '!' _ [composite_event] ;
 <prim> : [primitive_event] _ <with_delta>? ;
 <with_delta>? : 'WITH' _ "delta_struct_id" ;
 [primitive_event] : <p_event1>
                   | <p_event2> ;
 <p_event1> : <when>? _ [w_p_event] ;
 <when>? : [when_clause] ;
 [when_clause] : <when_before>
                | <when_after> ;
 <when_before> : 'BEFORE' ;
 <when_after> : 'AFTER' ;
 <p_event2> : [p_event] ;

//w_p_event
 [w_p_event] : <create_event>
              | <destroy_event>
              | <retrieve_event>
              | <update_event>
              | <insert_event>
              | <delete_event> ;
 <create_event> : 'CREATE' _ <persistence_clause>? _ "object" ;
 <destroy_event> : 'DESTROY' _ <persistence_clause>? _ "entity_name" ;
 <retrieve_event> : 'RETRIEVE' _ <persistence_clause>? _ [component_value] ;
 <update_event> : 'UPDATE' _ <persistence_clause>? _ [component_value] ;
 <insert_event> : 'INSERT' _ <persistence_clause>? _ [component_value] ;
 <delete_event> : 'DELETE' _ <persistence_clause>? _ [component_value] ;

//p_event
 [p_event] : <meth_beg_event>
            | <meth_end_event>
            | <attach_event>
            | <detach_event>
```

```
            | <trans_beg_event>
            | <trans_end_event>
            | <trans_com_event>
            | <trans_abo_event>
            | <prog_beg_event>
            | <prog_end_event>
            | <app_beg_event>
            | <app_end_event>
            | <user_event_event>
            | <temporal_event_event> ;
<meth_beg_event> : 'METHOD_BEGIN' _ <persistence_clause>? _ <method_call> ;
<meth_end_event> : 'METHOD_END' _ <persistence_clause>? _ <method_call> ;
<attach_event> : 'ATTACH' _ "object" ;
<detach_event> : 'DETACH' _ "object" ;
<trans_beg_event> : 'TRANSACTION_BEGIN' _ <in_application>? ;
<trans_end_event> : 'TRANSACTION_END' _ <in_application>? ;
<trans_com_event> : 'TRANSACTION_COMMIT' _ <in_application>? ;
<trans_abo_event> : 'TRANSACTION_ABORT' _ <in_application>? ;
<prog_beg_event> : 'PROGRAM_BEGIN' _ "program_name" _ <argument_list_p>?
                   _ <in_application>? ;
<prog_end_event> : 'PROGRAM_END' _ "program_name" _ <argument_list_p>?
                   _ <in_application>? ;
<app_beg_event> : 'APPLICATION_BEGIN' _ <application_name>? ;
<app_end_event> : 'APPLICATION_END' _ <application_name>? ;
<user_event_event> : 'USER_EVENT' _ "user_event_name" _ '('<parameter_list>?')' ;
<parameter_list>? : <parameter>+ ;
<parameter>+ : <param_name_type> ',' _ ... ;
<param_name_type> : "parameter_name" ':' [parameter_type] ;
[parameter_type] : <string>
                   | <int>
                   | "class_name" ;
<string> : 'STRING' ;
<int> : 'INTEGER' ;
<temporal_event_event> : 'TEMPORAL_EVENT' _ [temporal_spec] ;
[temporal_spec] : <temporal_absolute>
                  | <temporal_periodic>
                  | <temporal_relative>
```

```
                  | <temporal_relative_periodic> ;
 <temporal_absolute> : 'AT' _ <date_time> ;
 <date_time> : <date> _ <time> ;
 <date> : "day_value" ':' "month_value" ':' "year_value" ;
 <time> : "hour_value" ':' "minute_value" ;
 <temporal_periodic> : 'EVERY' _ <duration_list>+ _ <from>? _ <until>? ;
 <temporal_relative> : 'EVERY' _ <duration_list>+ _ <from>? _ <until>?
                       _ <relative_clause> ;
 <temporal_relative_periodic> : <duration_list>+ _ <relative_clause> ;
 <duration_list>+ : <duration> _ ... ;
 <duration> : "integer" _ [temporal_unit] ;
 [temporal_unit] : <second_unit>
                  | <minute_unit>
                  | <hour_unit>
                  | <day_unit>
                  | <week_unit>
                  | <month_unit>
                  | <year_unit> ;
 <second_unit> : 'SECOND' ;
 <minute_unit> : 'MINUTE' ;
 <hour_unit> : 'HOUR' ;
 <day_unit> : 'DAY' ;
 <week_unit> : 'WEEK' ;
 <month_unit> : 'MONTH' ;
 <year_unit> : 'YEAR' ;
 <from>? : 'FROM' _ <date_time> ;
 <until>? : 'UNTIL' _ <date_time> ;
 <relative_clause> : 'AFTER' _ [primitive_event] ;

//<persistence_clause>?
 <persistence_clause>? : 'NAMED' ;

//[component_value]
 [component_value] : <c_v_1>
                    | <c_v_2>
                    | [c_v_3] ;
 <c_v_1> : "entity_name" <position>? <path>? ;
```

```
 <c_v_2> : 'TYPE' _ '('"object"')' ;
 [c_v_3] : [type_O2] ;

//[type O2]
 [type_O2] : <set>
             | <unique_set>
             | <list>
             | <tuple>
             | "entity_name" ;
 <set> : 'SET' _ '('[type_O2]')' ;
 <unique_set> : 'UNIQUE SET' _ '('[type_O2]')' ;
 <list> : 'LIST' _ '('[type_O2]')' ;

 <tuple> : 'TUPLE' _ '('<attribute_list>+')' ;

//<attribute_list>+
 <attribute_list>+ : <attribute> ',' _ ... ;
 <attribute> : "identifier" _ ':' _ [type_O2] ;

//<path>?
 <path>? : [path_obj_val] ;
 [path_obj_val] : <path_obj>
                  | <path_val> ;
 <path_obj> : '->'"attribute_name" <position>? <path_val_sup>? ;
 <path_val_sup>? : <path_val> ;
 <path_val> : '.'"attribute_name" <position>? <path_val_sup>? ;

//<position>?
 <position>? : '['"integer" <position_s>?']' ;
 <position_s>? : "integer" ;

//<in_application>?
 <in_application>? : 'IN APPLICATION' _ "appli_name" ;

//<application_name>?
 <application_name>? : "appli_name" ;
```

```
//<method_call>
 <method_call> : "object" '->' "method_name" '('<argument_list>?')'
                 _ <result>? ;
 <argument_list_p>? : '('<argument>+')' ;
 <argument_list>? : <argument>+ ;
 <argument>+ : [argument_type] ',' _ ... ;
 [argument_type] : "argument_name"
                   | <underscore> ;
 <underscore> : '_' ;
 <result>? : "identifier" ;

//<condition>? (DEFINITION D'UNE REGLE)
 <condition>? : 'IF' _ [condition_1_2] ;
 [condition_1_2] : <condition1>
                   | <condition2> ;
 <condition1> : 'RANGE OF' _ "variable" _ 'IS' _ "type" _
                'SELECT' _ "O2_query_statement" ;
 <condition2> : "O2_query_statement" ;

//<action> d'une regle
 <action> : 'DO' _ <instead>? _ [action_body] ;
 <instead>? : 'INSTEAD' ;
 [action_body] : <a_body1>
                 | <a_body2> ;
 <a_body1> : '{ '!"O2C_statement"!' }' ;
 <a_body2> : '{%%' ! "O2C_declaration" ! '%%' ! "O2C_statement" !'}' ;

//Noeuds terminaux
 "rule_name" : /[a-z]+/ ;
 "program_name" : /[a-z]+/ ;
 "attribute_name" : /[a-z]+/ ;
 "method_name" : /[a-z]+/ ;
 "appli_name" : /[a-z]+/ ;
 "argument_name" : /[a-z]+/ ;
 "entity_name" : /[a-z]+/ ;
 "object" : /[a-z]+/ ;
 "user_event_name" : /[a-z]+/ ;
```

```
 "parameter_name" : /[a-z]+/ ;
 "class_name" : /[a-z]+/ ;
 "delta_struct_id" : /[a-z]+/ ;
 "O2_query_statement" : /[a-z]+/ ;
 "O2C_statement" : /[a-z]+/ ;
 "O2C_declaration" : /[a-z]+/ ;
 "identifier" : /[a-z]+/ ;
 "variable" : /[a-z]+/ ;
 "type" : /[a-z]+/ ;
 "integer" : /[1-9]+/ ;
 "day_value" : /[1-9]+/ ;
 "month_value" : /[1-9]+/ ;
 "year_value" : /[1-9]+/ ;
 "hour_value" : /[1-9]+/ ;
 "minute_value" : /[1-9]+/ ;
end
```