No ordre:

THESE

présentée à

L'université des Sciences et Technologies de Lille

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE

Spécialité: Productique: Automatique et Informatique Industrielle

par

Corinne BOS-PLACHEZ
Ingénieur I.S.E.N.

MODÉLISER POUR SIMULER POUR MODÉLISER
Contribution à l'Acquisition de Connaissances Comportementales

Soutenue le 30 janvier 1998 devant le jury composé de:

J.C. GENTINA	Président
M. CHEIN	Rapporteur
R. DIENG	Rapporteur
P. BORNE	Examinateur
B. BOTELLA	Examinateur
M. DAUCHET	Examinateur
P. VANHEEGHE	Examinateur

Thèse dirigée par M. P. VANHEEGHE et M. P. BORNE, préparée au Laboratoire d'Automatique et d'Informatique Industrielle de Lille (L.A.I.L U.R.A. - C.N.R.S. D1440), Ecole Centrale de Lille, I.S.E.N.

Remerciements

Je remercie Monsieur Jean Claude Gentina, Directeur de l'Ecole Centrale de Lille, d'avoir bien voulu présider le jury de cette thèse.

Que Monsieur Philippe Vanheeghe, Responsable du département Signaux et Systèmes de l'Institut Supérieur d'Electronique du Nord, soit remercié pour les conseils prodigués pendant ce travail.

Je remercie Monsieur Pierre Borne, Directeur Scientifique de l'Ecole Centrale de Lille, pour sa participation à la direction de ces travaux.

Je suis très reconnaissante envers Monsieur Michel Chein, Professeur à l'université de Montpellier II, et Madame Rose Dieng, Directeur de recherche à l'INRIA Sophia-Antipolis, d'avoir accepté d'être les rapporteurs de cette thèse.

Je remercie la société Dassault Electronique pour m'avoir confié les travaux de recherche à l'origine de cette thèse. Je remercie tout particulièrement Monsieur Bernard Botella pour m'avoir encadrée et orientée; les discussions fructueuses que nous avons eues au cours de ces trois années ont permis d'enrichir considérablement cette recherche. Son soutien moral m'a également été extrêmement précieux. Que Monsieur Patrick Taillibert soit associé à ces remerciements pour son optimisme constant et ses critiques acerbes.

Je remercie Monsieur Max Dauchet, Professeur à l'université de Lille 1, pour avoir accepté de faire partie de ce jury.

Je tiens aussi à remercier Magda Wozniak pour l'ambiance qu'elle a fait régner dans notre bureau et pour les commentaires sur mon travail.

Je remercie le Conseil régional du Nord-Pas de Calais et la société Dassault Electronique pour le financement de mes travaux.

J'ai enfin été touchée par la confiance que m'a accordée Monsieur Jean-Nöel Decarpigny, Directeur de l'I.S.E.N., en m'acceptant dans son établissement.

Que Monsieur Michel Lannoo, Directeur de la Recherche à l'I.S.E.N., soit associé à ces remerciements.

Table des matières

Chapitre 1 Introduction **9**

1.1. Acquérir des comportements humains 9
1.1.1. Motivations 9
1.1.2. Caractérisation des comportements 10
1.1.3. Problèmes 11
1.2. Modéliser pour acquérir 11
1.2.1. Approches ascendantes de modélisation 12
1.2.2. Approches descendantes de modélisation 12
1.2.3. Langages formels ou opérationnels de modélisation 13
1.2.4. Positionnement de notre approche 14
1.3. Modéliser pour simuler pour modéliser 15
1.4. Proposition d'un langage de modélisation 16
1.5. Organisation de la thèse 19

Partie I Modélisation: description du langage **21**

Chapitre 2 Représenter des connaissances avec des graphes conceptuels **23**

2.1. Niveau assertionnel 23
2.1.1. Graphes conceptuels simples 24
2.1.2. Graphes conceptuels emboîtés 25
2.2. Niveau terminologique 27
2.2.1. Différents types de représentation de connaissances terminologiques 27
2.2.2. Notre représentation d'une ontologie 29
2.3. Conclusion 35

Chapitre 3 Les concepts comportementaux **37**

3.1. Tâches et inférences 37
3.2. Actions 38
3.3. Diagrammes d'états 39
3.4. Nos objectifs 39
3.5. Introduction des concepts comportementaux 40

3.6. Conclusion 42

Chapitre 4 Le langage de modélisation 43

4.1. Comportements 43
4.2. Actions 45
4.2.1. Problème de persistance 45
4.2.2. Problème de ramification 47
4.2.3. Problème de qualification 48
4.2.4. Prise en compte de la durée 49
4.2.5. Les modèles génériques d'actions 50
4.3. Synchronisations temporelles 58
4.3.1. Relations temporelles 58
4.3.2. Synchronisation avec des événements 64
4.4. Branchements conditionnels 66
4.5. Indéterminations 67
4.6. Conclusion 69

Partie II Simulation: raisonnement sur les descriptions du monde 71

Chapitre 5 Représentation des point-situations, états, événements et transitions 73

5.1. Les propriétés du monde 73
5.1.1. Types de concepts et de relations 73
5.1.2. Référents 74
5.1.3. Emboîtements positifs 75
5.1.4. Négation 75
5.2. Point-situations 82
5.3. Etats 85
5.4. Evénements 86
5.5. Transitions 87
5.6. Conclusion 92
5.6.1. Enrichissement de l'ontologie du domaine 92
5.6.2. Prise en compte de formes plus riches de négation 92
5.6.3. Changements du monde plus expressifs 94

Chapitre 6 Démonstration 97

6.1. Démontrer avec des graphes conceptuels 97
6.1.1. La projection 97
6.1.2. Requêtes sur une base de connaissances 100

6.2. Démonstration des états et événements 105
6.2.1. Démonstration des états 105
6.2.2. Démonstration des événements 108
6.3. Conclusion 111

Chapitre 7 Mise à jour 113

7.1. Différentes approches de mise à jour 113
7.1.1. Approche STRIPS 114
7.1.2. Approches du changement minimal 114
7.1.3. Conclusion 118
7.2. Les lois dynamiques 119
7.2.1. Sémantique d'une loi dynamique 119
7.2.2. Lois dynamiques et lois statiques 121
7.3. Mise à jour des point-situations 123
7.3.1. Application d'une transition 123
7.3.2. Application d'une loi dynamique 127
7.3.3. Mise à jour 130
7.4. Conclusion 135

Partie III Simulation: interprétation de l'ontologie comportementale 137

Chapitre 8 Sémantique formelle de l'ontologie comportementale 139

8.1. La logique temporelle 139
8.1.1. Définition d'une séquence 140
8.1.2. Les objets 140
8.1.3. Les prédicats 141
8.1.4. Coréférences 142
8.2. Les contraintes 143
8.2.1. Comportements 143
8.2.2. Actions 145
8.2.3. Blocs conditionnels 149
8.2.4. Blocs disjonctifs 152
8.2.5. Blocs optionnels 153
8.3. Conclusion 154

Chapitre 9 Algorithme de simulation 155

9.1. Présentation de l'algorithme 156
9.1.1. Définition des variables 156
9.1.2. Algorithme synthétique 160

9.1.3. Algorithme détaillé 160
9.1.4. Choix des tâches dans Tpot 164
9.1.5. Instanciation d'un comportement 165
9.1.6. Démonstration 165
9.1.7. Mise à jour 168
9.1.8. Points de choix 168
9.1.9. Contrôle 169
9.2. Implémentation 169

Chapitre 10 Conclusion 173

10.1. Contributions 173
10.2. Travaux futurs 174
10.3. Vers un système d'assistance à la modélisation de comportements 175

Références 177

Annexe A Exemple de fiche réflexe 183

Annexe B L'ontologie comportementale 185

Annexe C Les relations temporelles 195

Annexe D L'ontologie générique du domaine 199

Annexe E Exemple de modélisation et résultats de simulation 205

Chapitre 1

Introduction

L'objectif de la thèse est de proposer un langage de modélisation assistant l'acquisition de connaissances comportementales dans une organisation humaine. L'originalité de ce langage est d'autoriser une simulation symbolique des comportements dont les résultats incitent l'expert à revenir sur la modélisation.

Ce chapitre introductif fixe le contexte de la thèse, puis positionne nos travaux dans le domaine de l'acquisition de connaissances. Il précise notamment ce qui nous a amenés à définir une simulation symbolique pour assister la modélisation des comportements. Il décrit ensuite plus précisément l'objectif de la thèse qui est la proposition d'un langage de modélisation. Il présente enfin le plan de la thèse qui comprend trois grandes parties. La première partie est consacrée à la description du langage de modélisation. Les deuxième et troisième parties sont relatives à la simulation. La deuxième partie traite des raisonnements sur les descriptions du monde effectués au cours de la simulation tandis que la troisième partie explique l'algorithme de simulation par rapport à la sémantique formelle du langage de modélisation.

1.1. Acquérir des comportements humains

Avant de présenter plus précisément nos objectifs, nous décrivons le contexte de la thèse qui est l'acquisition de connaissances comportementales dans une organisation humaine.

1.1.1. Motivations

L'acquisition de connaissances consiste à transférer une expertise dans des outils informatiques lors de la construction d'une base de connaissances ou d'un système à base de connaissances. Une base de connaissances est un ensemble de connaissances sur un domaine, qui sont généralement statiques. Un système à base de connaissances est composé d'une base de connaissances et de connaissances comportementales qui décrivent comment exploiter la base de connaissances pour la résolution d'un problème (diagnostic, planification...).

Nous nous intéressons à l'acquisition de ***connaissances comportementales*** d'un système lorsque celui-ci n'est pas un système à base de connaissances mais une ***organisation humaine***. Le résultat d'acquisition n'est donc pas un ensemble de méthodes de résolution de problèmes, mais une base de connaissances ***dynamiques***. Cette base de

connaissances pourra être intégrée avec d'autres composants pour construire diverses applications informatiques relatives à l'organisation humaine. Le but de ces applications est par exemple d'entraîner les membres de l'organisation, d'évaluer leurs comportements, d'étudier et de comprendre l'organisation,...; elles sont de natures diverses ([BOS 96]):

- Une simulation standard[1] génère les comportements simulés de l'organisation à partir d'un scénario initial
- Une application de suivi permet de comparer des comportements observés avec les comportements stéréotypés
- Une application de diagnostic permet de retrouver les données initiales (par exemple, des dysfonctionnements) à partir de l'observation de comportements réels

1.1.2. Caractérisation des comportements

Une organisation humaine est constituée d'un ensemble d'intervenants humains ayant des rôles différents dans l'organisation et évoluant dans un environnement complexe commun (des lieux et des instruments caractérisés par différents attributs).

Le comportement d'un intervenant caractérise son rôle dans une certaine situation (ie, à un moment donné, quand certaines propriétés de l'environnement sont établies), il décrit la façon dont cet intervenant transforme l'environnement, communique de l'information aux autres membres de l'organisation, raisonne ou prend une décision. Un comportement peut donc être caractérisé par des tâches de natures diverses devant être exécutées par un intervenant, dont on peut décrire le déroulement (les instruments utilisés, le lieu d'exécution, la procédure d'exécution...) et le résultat (la transformation de l'environnement, l'information communiquée...). Nous nommons action une tâche dont le résultat est explicitement décrit, les autres types de tâches sont des ensembles d'actions.

D'une part, l'intervenant d'une organisation n'effectue pas uniquement des actions les unes à la suite des autres, mais peut également exécuter plusieurs actions différentes en parallèle. Chaque comportement est donc un plan d'actions synchronisées par diverses relations temporelles. Comme des actions réalistes sont des actions qui durent, elles peuvent être associées à des intervalles temporels. Les relations temporelles symboliques basiques pouvant ordonner les actions sont ainsi au nombre de treize ([ALL 83]).

D'autre part, les membres d'une organisation n'agissent pas non plus les uns indépendamment des autres mais agissent de façon à ce que l'organisation en tant que tout ait un comportement cohérent. En plus des relations temporelles, les actions d'un comportement peuvent ainsi être synchronisées par des événements extérieurs, provoqués par les actions d'autres comportements.

Enfin, les personnes de l'organisation réagissent différemment suivant les situations. Les réactions diverses des personnes peuvent être prévues à l'avance dans la description des comportements; les comportements sont alors des plans conditionnels d'actions. Comme les êtres humains sont des agents capables de prendre des décisions, ils peuvent aussi avoir la liberté de choisir leurs réactions; les comportements contiennent

1. Nous utilisons le terme standard pour opposer ce type de simulation à la simulation symbolique dont le but est d'aider l'expert à enrichir la modélisation.

alors des indéterminations (relation temporelle imprécise entre deux actions, choix entre deux actions à exécuter, instruments utilisés pour réaliser une action...).

Des exemples de tels comportements peuvent être trouvés dans différents types d'organisations humaines:

- Les fiches réflexes des plans particuliers d'intervention de la sécurité civile décrivent des attitudes réflexes en cas de sinistre ou d'accident dans un site industriel (L'annexe donne un exemple de fiche réflexe extrait de [SCI 90]).
- Les missions types dans le domaine militaire décrivent des procédures à suivre pour qu'une organisation militaire puisse réaliser une mission
- Les protocoles cliniques dans le domaine médical décrivent comment une organisation médicale doit traiter des patients ayant certains symptômes ([MIK 97], [FRI 97]).

1.1.3. Problèmes

Il est difficile pour l'expert de décrire les comportements d'une organisation humaine car ce sont des connaissances complexes (agents divers, environnement complexe, relations temporelles variées...).

Ces connaissances sont en outre non ou mal établies car la connaissance de l'expert sur les comportements est incomplète; il a en tête un comportement global de l'organisation ou un but à atteindre ainsi que les rôles des membres de l'organisation mais ne sait pas facilement établir les comportements individuels de chacun des membres.

Il n'est enfin pas aisé pour l'expert de généraliser suffisamment les comportements afin qu'ils puissent être applicables dans des situations variées.

Les résultats d'acquisition sont donc incomplets, pauvres, peu robustes et n'atteignent pas le niveau de généricité requis. Ces problèmes ne sont découverts par l'expert que lorsque les applications finales sont démontrées devant lui sur des cas concrets. La prise en compte des remarques de l'expert est à ce moment là coûteuse car il est non seulement nécessaire de reconsidérer un phase de correction des comportements humains mais aussi une longue phase de développement des applications.

Pour remédier à ces problèmes, nous proposons:

- de guider l'expert dans la description des comportements par une approche descendante de ***modélisation***
- et ***surtout*** de le confronter le plus tôt possible aux résultats de modélisation grâce à une ***simulation*** symbolique des comportements

1.2. Modéliser pour acquérir

L'acquisition de connaissances peut être vue comme une activité de modélisation. Elle consiste alors à construire des modèles d'un système à base de connaissances ou d'une base de connaissances à différents niveaux d'abstractions et sous différents points de vue (modèle d'expertise, modèle de communication, modèle de

conception...). Dans [HEI 96], Heijst et al distinguent deux interprétations possibles de ce processus de modélisation: une approche ascendante et une approche descendante.

1.2.1. Approches ascendantes de modélisation

Les approches ascendantes de modélisation comportent deux phases: une phase d'élicitation et une phase de modélisation (cf figure 1).

L'élicitation consiste à collecter la connaissance de l'expert, elle est réalisée par des discussions entre l'expert et un ingénieur de la connaissance (ou cogniticien). Bien que de nombreuses techniques d'interview (libres, structurées, études de cas...) aient été proposées pour faciliter le transfert d'expertise, l'élicitation des connaissances demeure un travail long et nécessitant beaucoup de rigueur de la part du cogniticien.

La modélisation consiste alors à imposer une structure sur la connaissance déjà élicitée, elle est réalisée par le cogniticien qui peut s'aider:

- de techniques d'organisation de connaissances (Par exemple, le "Card Sorting" consiste à trier des objets dans des grilles répertoires pour structurer les connaissances ([GAM 84]))
- de méthodologies d'acquisition de connaissances (Par exemple, KOD est une méthode issue de la psychologie cognitive, proposant un modèle cognitif comprenant des taxinomies (descriptions hiérarchiques du monde), des actinomies (schémas mentaux d'actions) et des schémas d'interprétation permettant de comprendre le monde réel. Un modèle pratique et un modèle informatique correspondant au modèle cognitif sont également proposés dans cette méthode ([VOG 88]).)
- d'outils génériques d'acquisition des connaissances (Par exemple, CG-KAT - Conceptual Graph Knowledge Acquisition Tool- est un outil permettant de faciliter non seulement l'acquisition des connaissances mais aussi la recherche d'information dans les documents sources de l'expertise ([MAR 95], [MAR 96]))

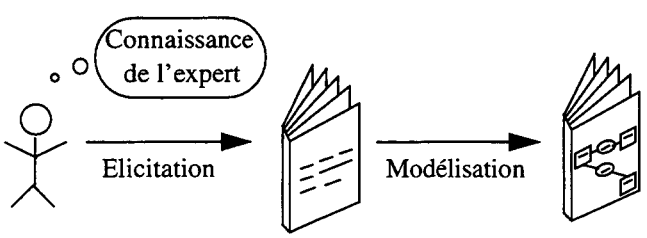


Figure 1. Approche ascendante de modélisation

1.2.2. Approches descendantes de modélisation

Les approches descendantes d'acquisition des connaissances sont des approches à base de modèles. Des modèles génériques sont sélectionnés puis instanciés avec les connaissances spécifiques de l'application afin de construire les modèles

d'expertise (cf figure 2).

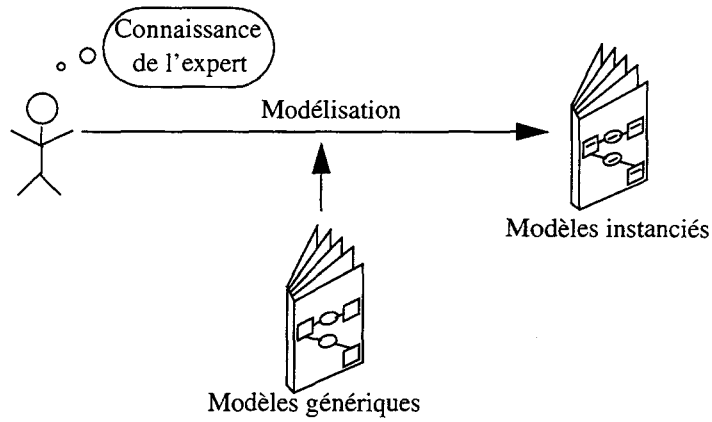


Figure 2. Approche descendante de modélisation

KADS ([SCH 88], [WIE 92]) et plus récemment CommonKADS ([BRE 94]) sont des méthodologies de développement de systèmes à base de connaissances qui préconisent l'utilisation de modèles génériques pour la modélisation de l'expertise. KADS distingue quatre niveaux de description de l'expertise:

- Le niveau domaine décrit la connaissance statique du domaine indépendamment de son utilisation pour la résolution d'un problème.
- Le niveau inférence spécifie les inférences basiques nécessaires pour la résolution du problème et décrit comment utiliser la connaissance du niveau domaine pour effectuer ces inférences.
- Le niveau tâche spécifie le contrôle (séquencement, itérations, conditions...) sur l'exécution des inférences définies au niveau inférence.
- Le niveau stratégie[1] concerne la sélection d'une tâche pour la résolution d'un but.

Ainsi, dans KADS et plus généralement dans le cadre de l'acquisition de connaissances pour la construction de systèmes à base de connaissances (ie, en ingénierie des connaissances), les modèles génériques proposés sont des modèles de tâches (encore appelés méthodes de résolution de problèmes), des modèles d'inférences ou des ontologies génériques. Ces différents modèles sont ensuite instanciés pour décrire les différents niveaux d'expertise.

1.2.3. Langages formels ou opérationnels de modélisation

Traditionnellement, les modèles génériques et donc les modèles d'expertise sont représentés avec des langages informels ou semi-formels. Récemment, des langages formels et/ou opérationnels ([FEN 91], [HAR 92]) ont été proposés pour modéliser

1. CommonKADS ne considère pas ce quatrième niveau.

l'expertise (cf figure 3).

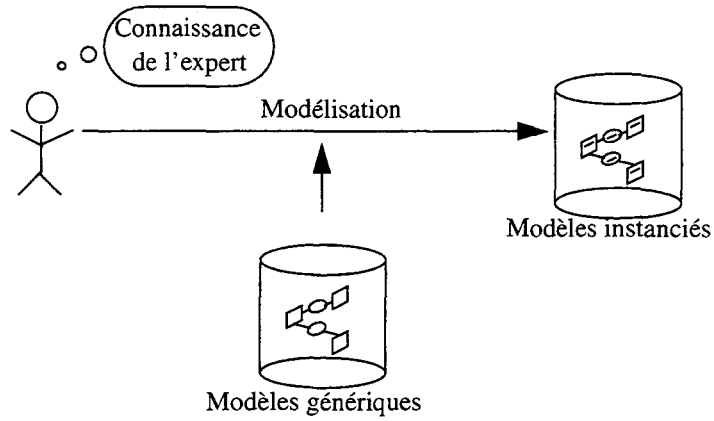


Figure 3. Approche descendante de modélisation avec un langage formel et/ou opérationnel

Le principal inconvénient de tels langages est la difficulté de lecture et de compréhension par l'expert. Leurs avantages sont les suivants ([FEN 94]):

- Ils réduisent l'ambiguïté de la représentation et donnent une sémantique claire aux modèles.
- La consistance des modèles peut être vérifiée par des preuves formelles et/ou par des exécutions symboliques.
- L'opérationnalisation des modèles est en partie automatisable; le prototypage est ainsi facilité.

1.2.4. Positionnement de notre approche

Dans notre cas, une approche descendante de modélisation est plus satisfaisante qu'une approche ascendante car elle permet de guider l'élicitation et la structuration des connaissances complexes et non établies que sont les comportements dans des organisations humaines.

En revanche, aucun des modèles génériques des approches à base de modèles n'est directement adapté pour modéliser les comportements d'une organisation humaine. Dans ces approches, les connaissances du domaine sont en effet statiques et les connaissances dynamiques modélisées ne sont que des méthodes de résolution de problèmes (par exemple, une méthode de diagnostic).

L'utilisation d'un langage formel de modélisation est également intéressant car il force à résoudre les ambiguïtés et permet de vérifier la consistance des comportements. Si les modèles de comportements contenaient des ambiguïtés ou des inconsistances, celles ci devraient être résolues par le concepteur de l'application informatique sans validation de l'expert ce qui pourrait éventuellement mener à l'obtention d'une application non réaliste.

Au vu des trois arguments précédents, notre objectif est de construire un langage ***formel, guidant*** la ***modélisation*** de ***comportements humains***. Une telle approche

de modélisation ne garantit cependant pas l'obtention de modèles suffisamment riches et applicables dans des contextes et situations variées. Il faut en plus ***valider*** les comportements en confrontant l'expert aux résultats de modélisation. Nous proposons de réaliser cette confrontation en montrant à l'expert les comportements ***simulés***.

1.3. Modéliser pour simuler pour modéliser

Généralement, on modélise un système pour le simuler. La simulation sert alors à étudier et comprendre le fonctionnement du système. C'est d'ailleurs l'objectif de la simulation standard évoquée au début de l'introduction et que nous considérons comme une application possible relative à une organisation humaine.

Nous introduisons un autre niveau de simulation, que nous qualifions de symbolique, dont l'objectif est d'assister l'expert et le cogniticien à valider la modélisation. Cette fois, on ***simule pour modéliser***. La simulation montre à l'expert les modèles de comportement "au travail" sur des cas concrets afin de lui faire prendre conscience des limitations, des performances et de la robustesse de la modélisation. L'expert va réagir et la discussion avec le cogniticien sera enrichie. Le cogniticien sera ainsi incité à revenir sur la modélisation en corrigeant et en enrichissant les modèles (cf figure 4).

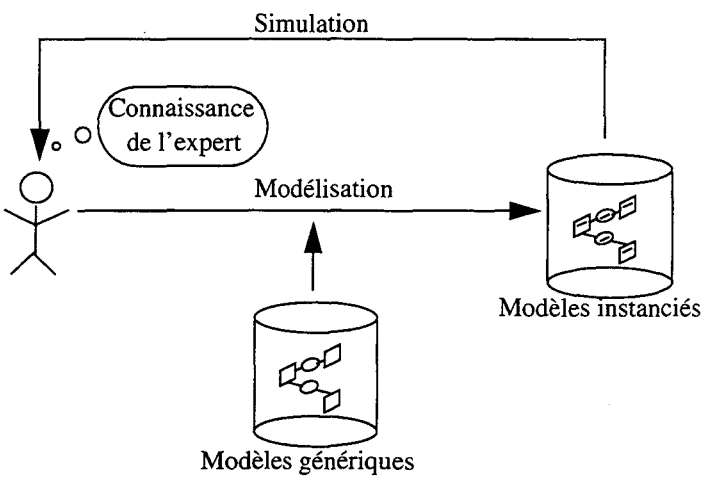


Figure 4. Simuler pour modéliser

Une telle simulation permet donc de valider les comportements ***au plus tôt***, avant la phase de développement des applications. En ce sens, nous suivons certaines recherches d'ingénierie des connaissances (cf environnement Cokace dans [COR 96]) qui préconisent la validation des modèles d'expertise avant l'implémentation du système à base de connaissance. L'environnement Cokace permet en effet au cogniticien de simuler un raisonnement avec la connaissance modélisée (ie, de faire tourner le modèle d'expertise) sans prendre en compte les détails d'implémentations. Dans notre cas, le résultat de la simulation n'est cependant pas un raisonnement mais une évolution du monde (organisation + environnement).

Nous qualifions la simulation utilisée de ***symbolique*** car elle ne simule pas fidèlement l'organisation humaine, mais une représentation approximative et simplifiée de l'organisation (les modèles de l'organisation sont d'autant plus approximatifs que les connaissances de l'expert sont incomplètes). Ce niveau de simulation correspond à la façon dont l'expert ferait tourner les comportements dans sa tête avec les seules connaissances qu'il détient. La simulation est notamment ***qualitative*** car elle ne prend pas en compte des informations quantitatives sur les durées (actions, comportements, écarts entre actions...). L'expert ne sait pas en effet attribuer des valeurs numériques à toutes les durées d'actions, toutes les durées de comportements, tous les écarts entre les actions... Même s'il est capable de donner certaines de ces valeurs, elles devront être traduites en information qualitative afin de pouvoir êtres comparées avec d'autres informations qualitatives.

Un premier mode de fonctionnement de la simulation nécessite uniquement une connaissance supplémentaire par rapport à la modélisation: un scénario initial correspondant à un cas concret. Les résultats de simulation, qui révèlent des défauts ou des manques dans les modèles de comportement, sont alors:

- des évolutions du monde, issues du scénario initial, que l'expert et le cogniticien peuvent consulter et interroger (plusieurs évolutions différentes peuvent être générées à cause de l'indétermination dans les comportements).
- des comptes rendus d'exécution mettant en évidence des inconsistances détectées automatiquement (par exemple, des actions obligatoires n'ont pas pu être exécutées, des objectifs n'ont pas pu être atteints...).

Un second mode interactif de fonctionnement de la simulation permet à l'expert de la perturber par l'introduction de connaissances anormales (une action est stoppée au début de ou pendant son exécution, un événement inattendu se produit...). De cette façon, l'expert peut juger de la robustesse des comportements

Afin d'autoriser cette simulation symbolique, le langage de modélisation doit être interprétable et avoir une sémantique bien définie pour la simulation. Ainsi, il faut pouvoir ***modéliser pour simuler,*** et ceci, en prenant en compte les incomplétudes des connaissances de l'expert.

1.4. Proposition d'un langage de modélisation

L'objectif de la thèse est donc de proposer un langage pour modéliser et simuler des comportements d'une organisation humaine. Au vu des problèmes qui se posent (connaissances complexes, incomplètes...) et des objectifs que nous voulons atteindre (simulation symbolique, modélisation descendante...), ce langage doit respecter un certain nombre de contraintes ([BOS 97a], [BOS 97c]):

- Il doit représenter et permettre de simuler l'incomplétude car il est difficile pour l'expert, au départ, de modéliser de façon complète les comportements.
- Il doit être adapté à la modélisation de comportements humains (mise en avant du concept de tâche ou d'action, contrôle flexible, indéterminations, monde

complexe).

- Sa sémantique doit être formelle pour favoriser la non ambiguïté et l'interprétabilité de la représentation.
- Il doit permettre de guider la modélisation.
- Les modèles et les résultats de simulation doivent être accessibles à l'expert afin que celui-ci se sente impliqué dans la modélisation et afin qu'il puisse détecter des défauts et des manques dans les comportements. Pour ce faire, l'expert peut et doit compter sur l'aide du cogniticien

Nous avons étudié différents langages de modélisation de connaissances comportementales. Aucun d'eux ne nous satisfait complètement.

• En ingénierie des connaissances ([SCH 88], [WIE 92]), des langages permettent de modéliser des tâches de raisonnement qui devront ensuite être exécutées par un système à base de connaissances. Ces langages sont intéressants par la mise en avant du concept de tâche pouvant être décomposée en sous-tâches, par la possibilité de créer des bibliothèques de modèles génériques permettant de guider la modélisation et par la représentation et la définition des connaissances du domaine.

Cependant, ils ne nous satisfont pas car ils ne sont pas adaptés à la modélisation de tâches humaines qui sont de natures diverses, dont les durées doivent être prises en compte, sur lesquelles un contrôle flexible doit être appliqué et qui contiennent des indéterminations.

De plus, les tâches basiques non décomposables (appelées inférences) ne sont pas forcément simulables au niveau de la modélisation. La façon dont elles transforment le monde n'est pas toujours explicitée dans le modèle d'expertise. Elles peuvent pointer vers un traitement externe à la modélisation, par exemple un programme informatique.

• Dans les domaines de l'intelligence artificielle relatifs à la modélisation du changement (planification, mise à jour, projection et explication temporelles), des langages permettent de représenter des actions par des préconditions et des postconditions ([FIK 88]). L'action est alors exécutable dans un monde courant si ses préconditions sont vérifiées dans ce monde et elle est exécutée en mettant à jour le monde courant avec ses postconditions. Une telle modélisation de l'action est avantageuse car l'action est simulable (ou exécutable) au niveau de la modélisation.

En revanche, dans notre cas, cette représentation de l'action nécessite quelques adaptations concernant la prise en compte d'actions qui durent et la résolution des problèmes de qualification et de ramification.

Il faut en plus appliquer sur ces actions un contrôle suffisamment flexible, afin d'obtenir des plans d'actions pouvant modéliser des comportements humains.

• Les langages classiques de modélisation de comportements de systèmes sont des diagrammes états-transitions. Les réseaux de Pétri ([PET 81]) sont un outil mathématique général décrivant des relations entre des conditions et des événements et permettant de visualiser des comportements comportant du parallélisme, de la synchronisation et du partage de ressources. Les statecharts, plus récemment proposés dans [HAR 87], sont des diagrammes d'états structurés. Ils permettent d'imbriquer des

états afin d'obtenir des descriptions à différents niveaux de généralisation et permettent également d'agréger des états afin d'exprimer des concurrences. Ces langages sont intéressants par leurs aspects graphiques, par les moyens de structuration (dans le cas des statecharts) et par la possibilité de simuler les comportements.

Cependant, ils ne nous satisfont pas car ils ne mettent pas suffisamment en avant la notion d'action en décrivant par exemple ses paramètres, ses prérequis, ses conditions de bon déroulement et la façon dont elle transforme le monde. Ils ne permettent pas non plus de décrire le domaine concerné (les intervenants, les endroits, les instruments, les attributs). Afin d'augmenter le pouvoir d'expression de ces langages, il faudrait en plus pouvoir décrire des aspects fonctionnels (transformations du monde) et des propriétés du domaine. C'est d'ailleurs ce que préconise la méthode OMT ([RUM 96]) par l'utilisation de trois modèles différents pour décrire un système: modèle objet, modèle dynamique (exprimé avec des statecharts) et modèle fonctionnel (exprimé avec des diagrammes de flots de données).

En outre, la sémantique des diagrammes états-transitions, reposant sur les seules notions d'état et de transition, ne permet pas de représenter naturellement et simplement un contrôle flexible, nécessaire dans le cas d'une organisation humaine, comprenant des conditions de natures différentes (nécessaires, nécessaires et suffisantes), des relations temporelles complexes entre intervalles et des indéterminations (choix entre deux actions, relations temporelles imprécises...).

Nous proposons donc une ontologie dédiée à la modélisation de comportements humains dont les concepts principaux sont ceux de processus, d'action, d'état, d'événement et de transition. Les relations liant ces concepts sont des relations temporelles, des relations de conditionnement, des relations de décomposition...Cette ontologie comporte également des concepts visant à modéliser des indéterminations afin de ne pas forcer l'expert à faire des choix qu'il ne sait pas faire ou afin qu'il puisse laisser libre choix à l'intervenant pour certaines décisions. Cette ontologie a enfin une sémantique formelle autorisant une simulation symbolique des comportements. Le langage de modélisation est alors constitué de cette ontologie exprimée dans le formalisme des graphes conceptuels ([SOW 84]). Les raisons qui nous ont incités à choisir le formalisme des graphes conceptuels sont les suivantes:

- Une notation graphique simple (rectangles, ovales, flèches) favorise la lisibilité du langage et donc l'implication de l'expert dans la modélisation. La lisibilité n'étant cependant plus garantie dans le cas de connaissances complexes et volumineuses, il faut pouvoir davantage structurer les connaissances. Ceci est possible dans le formalisme des graphes conceptuels en emboîtant les graphes.
- Chaque sommet d'un graphe est étiqueté par un label pouvant être choisi parmi tous les termes du langage naturel. Ils peuvent être choisis dans les concepts de l'expert afin que le langage lui soit plus facilement accessible. Ils peuvent également être choisis de façon à obtenir un langage dédié à la modélisation de comportements humains, c'est à dire permettant de représenter des actions, des états, des événements, des relations temporelles... Ils peuvent enfin être choisis dans des concepts du domaine afin de construire une ontologie du domaine et de décrire ainsi le monde concerné (intervenants, instruments, localisations, attributs...). Il est cependant important de préciser que le langage est uniquement dédié par le choix des labels et que donc des outils développés autour des graphes conceptuels seront

réutilisables pour d'autres applications que la modélisation de comportements humains.

- De plus, chaque label choisi peut être classé dans une hiérarchie, expliqué, documenté, défini complètement ou partiellement et contraint dans son utilisation. Ceci a l'avantage non seulement de favoriser la lecture des graphes mais surtout de guider la construction des graphes.
- Chaque graphe conceptuel ayant une interprétation en logique des prédicats du premier ordre, la connaissance est représentée de façon non ambiguë et dans un cadre logique (quantifications, négations). Des opérations de raisonnement sur les graphes sont implémentées par des algorithmes de graphes et sont consistantes et complètes vis à vis de cette interprétation logique ([CHE 92], [MUG 95]). Ces opérations, bien qu'elles nécessitent quelques extensions, nous permettent d'effectuer les traitements automatiques sur les comportements, utiles pour la simulation.

1.5. Organisation de la thèse

Le mémoire de thèse présente le langage de modélisation et explique la simulation symbolique des comportements modélisés dans ce langage.

La ***première partie*** de la thèse est consacrée à la description de l'ontologie comportementale, qui, représentée dans le formalisme des graphes conceptuels, constitue le langage de modélisation.

Dans cette partie, nous présentons le formalisme des graphes conceptuels du point de vue représentation de connaissances, sans prendre en compte les aspects de raisonnement, dans le but de favoriser la compréhension du langage de modélisation. Nous justifions le choix des concepts de l'ontologie comportementale en fonction de nos objectifs et vis à vis des concepts comportementaux des langages de modélisation évoqués dans le 1.4. Nous décrivons enfin le langage de modélisation.

Les ***deuxième et troisième parties*** sont relatives à la simulation symbolique des comportements.

La ***deuxième partie*** aborde les aspects de raisonnement sur les propriétés du monde décrivant des états, des événements et des effets d'actions. L' algorithme de simulation doit pouvoir démontrer les états et les événements à partir d'une description courante du monde pour vérifier l'exécutabilité des actions. Et il doit pouvoir mettre à jour une description courante du monde avec des effets afin de simuler l'exécution des actions.

Dans cette partie, nous décrivons d'abord la représentation des propriétés du monde, qui nécessite d'étendre le modèle de base des graphes conceptuels en prenant en compte une forme de négation. Puis, nous discutons de la démonstration des états et des événements. Nous expliquons enfin un principe de mise à jour le monde, les changements du monde étant exprimés dans le formalisme des graphes conceptuels.

La ***troisième partie*** est consacrée à l'interprétation de l'ontologie comportementale par l'algorithme de simulation.

Dans cette partie, nous définissons formellement la sémantique des concepts

et des relations de l'ontologie comportementale. Cette sémantique est exprimée en une logique temporelle réifiée. Puis, nous donnons l'algorithme de simulation qui repose sur cette sémantique.

Nous ***concluons*** en résumant nos apports et en décrivant des travaux futurs qu'il serait intéressant de réaliser pour compléter cette recherche. Nous évoquons alors des perspectives applicatives relatives à l'implémentation d'un système d'assistance à la modélisation des comportements. Dans ce système, des outils de construction assistent le cogniticien à construire les modèles de comportement et des outils de simulation assistent l'expert et le cogniticien à évaluer les modèles (cf figure 5).

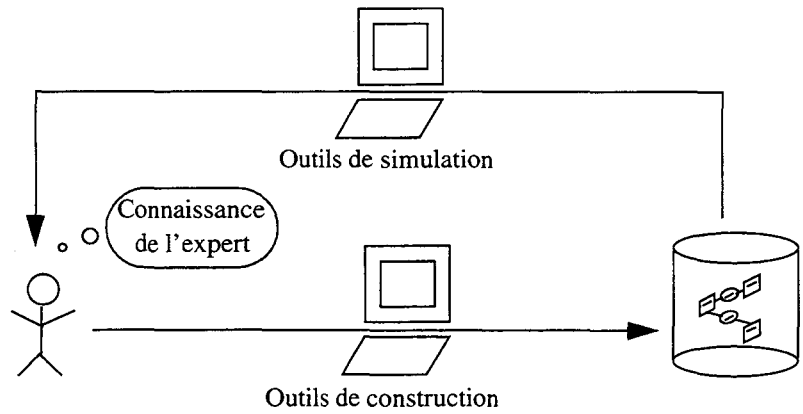


Figure 5. Un système d'assistance à la modélisation de comportements

Partie I

Modélisation: description du langage

Cette partie est consacrée aux choix des concepts comportementaux, à leur représentation et à leur sémantique intuitive. Elle présente l'ontologie comportementale qui structure et définit l'ensemble des concepts comportementaux et les relations entre ces concepts. L'ontologie comportementale représentée dans le formalisme des graphes conceptuels constitue le langage de modélisation des comportements.

Les graphes conceptuels sont un langage de représentation des connaissances, proposé par Sowa dans [SOW 84], permettant à la fois de définir du vocabulaire (ie, construire une ontologie) et d'utiliser ce vocabulaire pour asserter des faits. Les connaissances que l'on peut représenter avec des graphes conceptuels sont très variées suivant les domaines et les applications. Elles dépendent en fait du vocabulaire que l'on a défini. Avant de représenter des connaissances avec des graphes conceptuels, il faut ainsi déterminer une ontologie dédiée à ce que l'on veut représenter. Nous avons adapté le formalisme des graphes conceptuels à la modélisation de comportements humains par la construction d'une ontologie comportementale. Nous avons tout d'abord choisi les concepts comportementaux de façon à pouvoir modéliser des tâches humaines qui seront simulables au niveau de la modélisation. Nous avons alors représenté, structuré et défini ces concepts comportementaux dans le formalisme des graphes conceptuels. Le langage ainsi constitué permet de représenter et de simuler des modèles de comportements humains.

Le premier chapitre de cette partie est consacré à la description du formalisme des graphes conceptuels sous un point de vue représentation de connaissances sans prendre en compte les aspects de raisonnement. Nous introduisons ensuite les concepts principaux de notre ontologie comportementale en les comparant aux concepts de quelques langages de modélisation de comportements (langages de tâches en ingénierie des connaissances, langages de représentations d'actions en intelligence artificielle, diagrammes d'états). Nous présentons enfin le langage de modélisation en décrivant l'ontologie comportementale exprimée dans le formalisme des graphes conceptuels.

Chapitre 2

Représenter des connaissances avec des graphes conceptuels

L'objectif de ce chapitre est de rappeler quelques notions sur le formalisme des graphes conceptuels afin de favoriser la compréhension de l'ontologie comportementale exprimée dans ce formalisme. Notre vue du modèle des graphes conceptuels est proche de celle de M.Chein et M.L.Mugnier dans [CHE 92] et [MUG 95]. Nous prenons cependant quelques libertés vis à vis de leur travail de formalisation. Celles-ci seront mises en évidence et discutées dans ce chapitre.

Un langage de représentation des connaissances comporte des structures de données pour exprimer des connaissances et des mécanismes de raisonnement pour exploiter ou construire de nouvelles connaissances. Dans le cas des graphes conceptuels, les structures de données sont des graphes et les mécanismes de raisonnement des algorithmes de graphes. Dans cette partie relative au langage de modélisation, nous abordons les graphes conceptuels seulement du point de vue représentation. Les raisonnements utiles pour la simulation sont en effet uniquement effectués sur des graphes de description du monde comportant des types du domaine mais pas de types comportementaux. Les aspects de raisonnement seront ainsi abordés dans la partie suivante, relative à la description du monde et au raisonnement sur ce monde.

Les graphes conceptuels sont un langage mixte de représentation, ils permettent de représenter des connaissances terminologiques et des connaissances assertionnelles. Dans notre cas, l'ontologie comportementale et l'ontologie du domaine sont représentées au niveau terminologique. Les modèles de comportements sont des connaissances assertionnelles construites à partir des connaissances terminologiques comportementales et des connaissances terminologiques du domaine. Dans un premier temps, nous décrivons le niveau assertionnel du formalisme des graphes conceptuels (ie, les types de graphes qui vont représenter les comportements). Nous décrivons ensuite le niveau terminologique dans lequel des graphes similaires à ceux du niveau assertionnel vont servir à définir le vocabulaire de l'ontologie.

2.1. Niveau assertionnel

Les graphes conceptuels simples permettent d'asserter des faits avec une notation graphique et avec une interprétation en logique des prédicats du premier ordre. Ils peuvent être étendus en des graphes emboîtés qui sont des graphes de graphes

conceptuels permettant de créer plusieurs niveaux de représentation.

2.1.1. Graphes conceptuels simples

Un graphe conceptuel est un graphe fini, biparti que nous ne supposons pas forcément connexe[1]. Les sommets des graphes sont de deux types: les concepts expriment des entités, des collections, des abstractions ou des situations et les relations expriment la nature et les propriétés des liens entre ces concepts. Un graphe conceptuel peut être uniquement composé de sommets concepts mais chaque sommet relation doit être attaché à au moins un sommet concept. L'arité d'une relation indique le nombre d'arêtes adjacentes à ce sommet relation. Les relations que nous manipulons sont généralement d'arité binaire, bien que nous ne nous soyons pas contraints à une telle restriction

Le label d'un sommet concept est composé du type du concept, par exemple employé ou communiquer (cf figure 6), et du marqueur du concept, par exemple dupont. Le marqueur peut être individuel ou générique. Un concept qui a un marqueur individuel (comme dupont) se réfère à une instance particulière du concept tandis qu'un concept qui a un marqueur générique (noté * ou caractérisé par une absence de marqueur) se réfère à une instance non identifiée du concept. Le label d'un sommet relation est le type de la relation, par exemple, agent ou objet.

La figure 6 donne un exemple de graphe conceptuel qui correspond à la connaissance "L'employé dupont communique un message à l'employé durand". Les concepts sont représentés par des rectangles et les relations par des ovales. L'ordre associé aux arêtes adjacentes à un sommet relation est représenté par la numérotation des arêtes de 1 à l'arité de la relation. Quand la relation entre deux concepts est un nom, elle se lit "concept 1 est la relation du concept 2", si ce n'est pas un nom, elle se lit "concept 1 relation concept 2"

En résumé et de façon plus formelle, un graphe conceptuel G est un quadruplet (R,C,U,lab) où R est l'ensemble des sommets relations, C l'ensemble des sommets concepts, U l'ensemble des arêtes et lab une application qui à tout sommet associe un label.

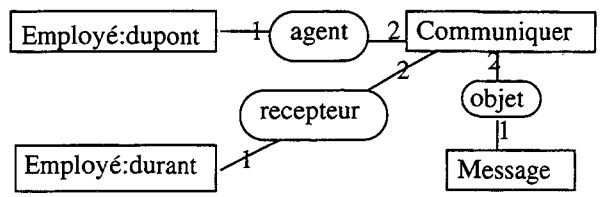


Figure 6. Un exemple de graphe conceptuel G

J.Sowa ([SOW 84]) définit un opérateur ϕ d'interprétation d'un graphe conceptuel en une formule de la logique des prédicats du premier ordre. Cet opérateur permet de donner aux graphes conceptuels une sémantique formelle:

1. Cette hypothèse s'oppose à celle de J.Sowa ([SOW 84]) qui considère que les graphes conceptuels sont des graphes connexes, mais est en accord avec certains travaux de M.L.Mugnier et M. Chein ([MUG 95]).

- Il associe à chaque concept générique une variable distincte xi
- Il associe à chaque marqueur individuel m la constante m
- Il représente chaque concept c par un prédicat unaire de nom type(c) et d'argument la variable ou la constante associée à c
- Il représente chaque relation r par un prédicat n-aire de nom type(r) et de ième argument, la variable ou constante associée au sommet concept lié à r par l'arête étiquetée i
- Il prend la conjonction de tous les prédicats
- Il ferme existentiellement les variables de la formule

La formule (1) donne ainsi l'interprétation logique du graphe de la figure 6.

$$\exists x \exists y(\text{Communiquer}(x) \wedge \text{Message}(y) \wedge \text{Employe}(\text{dupont}) \wedge \text{Employe}(\text{durant}) \wedge \text{agent}(\text{dupont}, x) \wedge \text{recepteur}(\text{durant}, x) \wedge \text{objet}(y, x)) \quad (1)$$

2.1.2. Graphes conceptuels emboîtés

Emboîter des graphes conceptuels permet de créer plusieurs niveaux de représentation.

D'une part, cela peut être avantageux pour structurer les graphes et ainsi les rendre plus lisibles dans le cas de connaissances volumineuses. On peut par exemple emboîter les attributs d'une personne dans le concept de type personne plutôt que de tous les attacher à ce concept par des relations de types attribut. Le concept personne est alors davantage mis en valeur que les concepts attribut.[1]

D'autre part, emboîter des graphes peut parfois être indispensable pour établir un lien entre un concept c et un graphe g (ie, établir un lien entre c et non seulement tous les concepts de g mais aussi toutes les relations de g). Il serait impossible de créer un tel lien avec tous les sommets concepts et relations au même niveau de représentation. Dans [MUG 95], M.L. Mugnier et M. Chein évoquent l'exemple d'un tableau qui représente un bateau sur la mer. Un lien de "description" ou "représentation" lie le concept de type tableau au graphe exprimant que le bateau est sur la mer. Les emboîtements que nous utilisons pour la modélisation des comportements appartiennent au cas indispensable d'emboîtement bien qu'ils aient en plus l'avantage de clarifier les graphes.

Nous utilisons des emboîtements de graphes de deux types: des emboîtements simples et des emboîtements typés.

Dans le cas des emboîtements simples ([MUG 95]), le label d'un concept a un troisième champ, appelé la description partielle interne du concept qui peut être une description générique ** ou un ensemble de graphes. Nous utilisons par exemple des emboîtements simples pour décrire des "états". Le graphe emboîté à l'intérieur du concept état exprime la propriété du monde invariante pendant la durée de l'état (cf figure 7).

1. La sémantique logique n'est cependant pas la même dans les deux cas (cf [CHE 97b] pour une interprétation logique des graphes emboîtés en logique des prédicats du premier ordre).

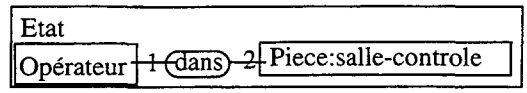


Figure 7. Un exemple d'emboîtement simple

Dans le cas des emboîtements simples, la sémantique d'emboîtement est forcément unique, elle est donnée par le type du concept emboîtant, par exemple état. Il peut être également avantageux d'emboîter dans le même concept plusieurs graphes avec différentes sémantiques d'emboîtement. Il est alors nécessaire d'utiliser des graphes emboîtés typés pour exprimer les sémantiques d'emboîtement. La description partielle interne d est alors remplacée par un ensemble {(e1,G1),(e2,G2),....,(ek,Gk)} avec ei les typages d'emboîtements et Gi les graphes emboîtés typés ([CHE 97a]). Nous utilisons par exemple des emboîtements typés pour décrire des "événements" (cf partie II pour une définition plus détaillée des événements). Les graphes emboîtés dans le concept événement expriment des propriétés du monde qui deviennent vraies, ou deviennent fausses (suivant le typage de l'emboîtement) au moment de l'événement. La notation utilisée est celle de la figure 8. Dans cet exemple, au moment de l'événement, la sirène se met à sonner et la lumière s'éteint.

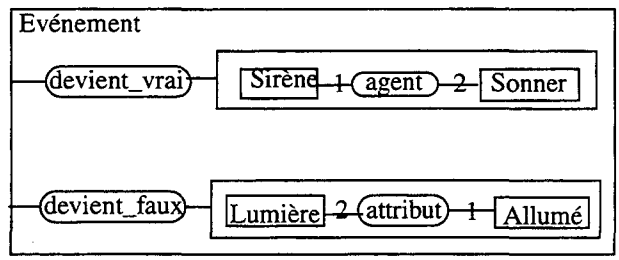


Figure 8. Un exemple d'emboîtement typé

Emboîter des graphes nécessite d'utiliser des liens de coréférence pour exprimer que certains concepts se réfèrent à la même instance. Nous autorisons ces liens entre des concepts génériques de même type et nous les représentons indifféremment par des coloriages similaires, par des lignes en pointillé ou par un nom de variable identique pour les deux concepts. Dans le graphe de la figure 9, c'est le même opérateur qui possède un téléphone et qui est dans la salle de contrôle.

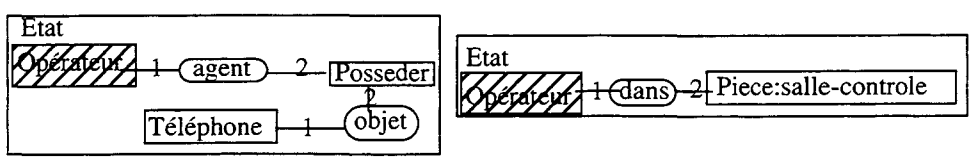


Figure 9. Emboîtements avec liens de coréférences

2.2. Niveau terminologique

Un fait asserté par un graphe conceptuel ne prend un sens que lorsque le vocabulaire utilisé pour l'assertion est clairement défini. Afin de pouvoir facilement lire des graphes, il faut ainsi pouvoir consulter l'ontologie relative à ces graphes.

L'écriture d'un graphe est beaucoup plus difficile que la lecture d'un graphe. Fixer à priori une ontologie permet d'aider à construire les graphes et permet surtout de normaliser et donc d'unifier la représentation des graphes.

Nous énumérons et expliquons les différents types de représentation qui ont été proposés dans la bibliographie sur les graphes conceptuels pour exprimer des connaissances terminologiques. Puis, nous présentons notre représentation d'une ontologie par l'utilisation de métaconcepts. Ces derniers permettent de caractériser les types de concept et de relation par des graphes, ceci, de différentes façons: en exprimant des conditions nécessaires et/ou suffisantes, en donnant des contraintes et des exemples d'utilisation...

2.2.1. Différents types de représentation de connaissances terminologiques

La notion de support

Dans [CHE 92], M.Chein et M.L.Mugnier introduisent la notion de support pour structurer les termes du vocabulaire et définir leurs contraintes d'utilisation dans les graphes. Un support est un quintuplet S=(TC, TR, s, I, c) tel que:

- TC est l'ensemble des types de concepts, il est partiellement ordonné par une relation sorte de (notée $\leq_c$), il possède un plus grand élément (T, le type universel) et un plus petit élément ($\perp$, le type absurde). Si tc $\leq_c$ tc' alors $\forall x(tc(x) \rightarrow tc'(x))$.
- TR est l'ensemble des types de relations, il est partitionné en ensembles de relations de même arité, chacun de ces ensembles est partiellement ordonné par une relation sorte de (notée $\leq_r$) et possède un plus petit élément et un plus grand élément. Si tr $\leq_r$ tr' avec tr et tr' des types de relation binaires alors $\forall x \forall y(tr(x, y) \rightarrow tr'(x, y))$.
- s associe à tout type de relation sa signature, cette signature est unique, elle fixe l'arité de la relation et donne les types maximaux de concepts qu'une relation de ce type peut lier. Une signature est représentée par un graphe étoile, c'est-à-dire un graphe composé d'un sommet relation r et d'un ensemble de sommets concept admettant comme unique relation voisine la relation r.
- I est l'ensemble des marqueurs individuels.
- c associe à tout marqueur individuel un type de concept différent du type absurde, c'est la relation de conformité du marqueur qui correspond à une relation est un.

Si on asserte des faits avec des graphes emboîtés typés, il faut ajouter au support l'ensemble des types d'emboîtement. Cet ensemble est partiellement ordonné et possède un plus grand élément (le type description).

La base canonique

Une base canonique permet de contraindre la construction des graphes. Elle est constituée d'un ensemble de graphes conceptuels. Un ensemble de règles (appelées règles de spécialisation) permet de dériver de nouveaux graphes à partir de la base canonique. Les graphes obtenus sont des graphes canoniques.

Les règles de spécialisation sont les suivantes:

- *Simplification*: si un graphe possède deux sommets relations de même type ayant les mêmes sommets concepts voisins dans le même ordre, le graphe simplifié s'obtient en supprimant un des deux sommets relations.
- *Restriction de relation*: le type d'une relation r est remplacé par un type t tel que $t \leq_r type(r)$, les contraintes fixées par les signatures doivent cependant être conservées.
- *Restriction de concept*: le type d'un concept c est remplacé par un type t tel que $t \leq_c type(c)$ et/ou le marqueur de c est remplacé par un marqueur m tel que $m \leq$ marqueur(c) ($m < m'$ si et seulement si m est un marqueur individuel et m' est un marqueur générique), les contraintes fixées par les relations de conformité doivent cependant être conservées.
- *Joint interne*: deux sommets concepts d'un même graphe ayant la même étiquette sont fusionnés
- *Joint externe*: deux sommets concepts de deux graphes différents ayant la même étiquette sont fusionnés

Propriété: l'union de l'ensemble des graphes étoiles et d'un graphe composé d'un unique concept générique de type universel forme une base canonique, dans la suite de ce chapitre, nous nommons cette base canonique B0 (Dans [MUG 93], M.L. Mugnier et M. Chein présentent une étude plus complète de la canonicité et montrent cette propriété).

Les définitions de types

Dans [SOW 84], J. Sowa introduit la notion de définition de type. Elle est dérivée des méthodes de définition d'Aristote. Un nouveau type de concept est distingué d'un type de concept déjà existant, appelé le genre, par un graphe, appelé la différence, contenant un concept de même type que le genre. La différence constitue en fait un ensemble de conditions nécessaires et suffisantes du nouveau type.

La notation que J. Sowa utilise pour définir un type est la suivante:

- type t(x) is D(x); t est le type défini, D le graphe différence et x un référent générique distingué des autres référents du graphe différence.

Par exemple, J. Sowa définit un éléphant de cirque comme un éléphant qui fait des numéros dans un cirque (cf figure 10).

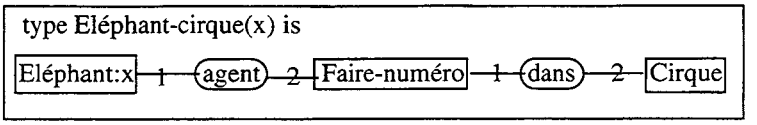


Figure 10. Un exemple de définition de type

J. Sowa étend le mécanisme de définition de types de concepts aux types de relation. Le nombre de référents distingués dans le graphe différence correspond alors à l'arité de la relation définie, par exemple pour une relation binaire:

- relation t(x,y) is D(x,y)

Les schémas

Dans [SOW 84], J. Sowa introduit la notion de schéma comme étant une façon plausible d'utiliser un type de concept dans un graphe. La notation utilisée est la même que celle utilisée pour les définitions de types:

- schema for t(x) is S(x)

2.2.2. Notre représentation d'une ontologie

Nous considérons une hiérarchie de types de concepts, une hiérarchie de types de relations, une hiérarchie de types d'emboîtements, un ensemble de marqueurs individuels et une relation de conformité tels qu'ils sont définis dans un support (cf 2.2.1). Nous appelons métaconcepts les concepts de signature, définition de type, schéma... Ils permettent de caractériser les types de concept et de relation par des graphes, ceci, de différentes façons: en exprimant des conditions nécessaires et/ou suffisantes, en donnant des contraintes et des exemples d'utilisation...

Tout d'abord, les métaconcepts documentent et expliquent les types de concepts et de relations. Dans notre cas, ils permettent ainsi de documenter le langage de modélisation (cf annexe B). Il est intéressant que cette documentation soit consultable au moment de la lecture des modèles de comportement mais aussi au moment de leur construction.

Grâce aux règles de spécialisation, ils aident ensuite à construire des graphes en contraignant et en guidant la saisie. Un graphe caractérisant un type est spécialisé pour construire un autre graphe. C'est en ce sens que notre approche de modélisation est descendante, car guidée par les métaconcepts.

Ils interviennent enfin au niveau du raisonnement sur les graphes. Dans notre cas, ils interviennent donc au moment de la simulation des comportements. Nous verrons par exemple dans la deuxième partie de la thèse comment des métaconcepts associés à des types du domaine sont utilisés pour maintenir la cohérence des descriptions du monde.

La figure 11 présente une hiérarchie de métaconcepts divisée en trois sous-hiérarchies: les contrainte d'utilisations, les définitions et les schémas. Nous définissons maintenant les différents métaconcepts de chacune de ces sous-hiérarchies. Nous donnons

leur interprétation logique et nous les illustrons avec des exemples (liés à l'ontologie comportementale ou à l'ontologie du domaine).

Nous donnons également leurs propriétés d'héritage. De telles propriétés permettent de faciliter une construction de l'ontologie par niveaux. L'ontologie comportementale constitue le premier niveau de l'ontologie globale. Ce niveau est indépendant du domaine. Il est figé car sa sémantique est exploitée par l'algorithme de simulation. Le deuxième niveau est une spécialisation du premier niveau, il est représentatif d'un domaine, par exemple celui de la sécurité civile. Dans le domaine de la sécurité civile, d'autres niveaux, plus spécialisés, peuvent par exemple être relatifs à des sites industriels particuliers.

Nous évoquons enfin brièvement les intérêts de chacun de ces métaconcepts pour la documentation, la construction des graphes et le raisonnement.

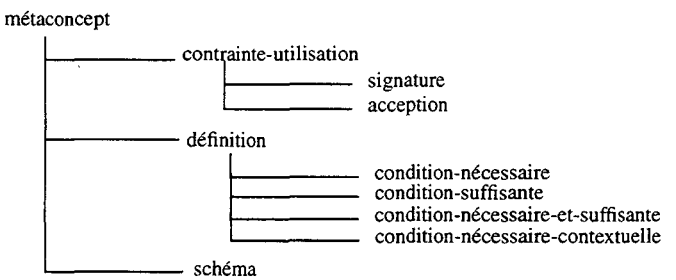


Figure 11. Hiérarchie de métaconcepts

Contraintes d'utilisation

Une ***signature*** est une contrainte d'utilisation d'un type de relation dans un graphe. Elle fixe l'arité de la relation et donne deux types maximaux de concepts que cette relation peut lier, notés (tc_1,tc_2). Contrairement à [CHE 92], nous ne considérons pas une unique signature pour un type de relation mais un ensemble de signatures. Par exemple, une relation de type précondition peut lier un concept de type état à un concept de type action ou à un concept de type déroulement (cf annexe B). Autoriser plusieurs signatures d'un type de relation évite de devoir créer des concepts artificiels non-utilisés dans les graphes (par exemple, le type de concept action-ou-comportement) ou de devoir multiplier inutilement le nombre de types de relations (par exemple, les relations précondition-action et précondition-comportement).

L'interprétation logique d'un ensemble de signatures $\{(tc_{1i},tc_{2i})\}$ d'un type de relation binaire tr est la suivante:

$$\forall x \forall y \left(tr(x, y) \rightarrow \bigvee_i (tc_{1i}(x) \wedge tc_{2i}(y)) \right) \quad (2)$$

L'interprétation logique de l'ensemble des signatures du type de relation précondition est ainsi la suivante:

$$\forall x \forall y (precondition(x, y) \rightarrow (etat(x) \wedge action(y)) \vee (etat(x) \wedge comportement(y))) \quad (3)$$

Dans [MUG 95], une relation de type tr doit respecter les signatures des sur-types de tr. Pour ce faire, il est imposé que la signature d'un type de relation tr soit

compatible avec les signatures des sur-types de tr. Deux signatures (tc_1,tc_2) et (tc_1',tc_2') de deux types de relation binaire sont compatibles si et seulement si $tc_1 \leq_c tc_1'$ et $tc_2 \leq_c tc_2'$. La signature d'un type tr est ainsi une spécialisation de chacune des signatures des sur-types de tr. On peut alors parler d'héritage par un type, des signatures de ses sur-types.

Dans notre cas, pour que chaque relation de type tr respecte les ensembles de signature de chacun des sur-types de tr, il faut que chacune des signatures de tr soit compatible avec au moins une des signatures de chacun des sur-types de tr. Ainsi, chaque signature de l'ensemble des signatures d'un type de relation tr est une spécialisation d'au moins une des signatures de chacun des sur-types de tr.

Nous étendons la notion de contrainte d'utilisation aux types de concepts. Une ***acception*** est une contrainte d'utilisation d'un type de concept dans un graphe. Une acception d'un type de concept tc donne un type maximal de concept tc' et un type maximal de relation tr, notés (tr,tc'), tels qu'il soit possible de lier un concept de type tc à un concept de type tc' par une relation de type tr.

Nous imposons qu'un graphe représentant une acception respecte les contraintes imposées par les signatures. Par exemple, une acception du verbe parler respecte la signature de la relation agent (cf figure 12). Un graphe représentant une acception est ainsi une spécialisation de B0.

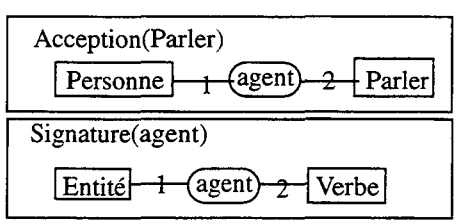


Figure 12. Une acception respectant une signature

L'interprétation logique d'un ensemble d'acceptions $\{(tr_i,tc_i)\}$ d'un type de concept tc est la suivante (trel est le plus grand type de relation binaire):

$$\forall x \forall y \left(tc(x) \wedge trel(x, y) \rightarrow \bigvee_i (tr_i(x, y) \wedge tc_i(y)) \right) \quad (4)$$

Comme dans le cas des signatures, nous imposons qu'un concept de type tc respecte les ensembles d'acceptions de chacun des sur-types de tc, il faut que chacune des acceptions de tc soit compatible avec au moins une des acceptions de chacun des sur-types de tc. Ainsi, chaque acception de l'ensemble des acceptions d'un type de concept tc est une spécialisation d'au moins une des acceptions de chacun des sur-types de tc.

En ce qui concerne les intérêts des métaconcepts contraintes d'utilisation, l'ensemble des signatures de tous les types de relations et l'ensemble des acceptions de tous les types de concepts peuvent être utilisés pour vérifier la syntaxe des graphes. Chaque graphe doit être une spécialisation (ie, dérivé par les règles de spécialisation) de B0 mais aussi de l'union de l'ensemble des acceptions et d'un graphe composé d'un unique concept générique de type universel. Les contraintes imposées sur les graphes par les acceptions sont plus strictes que celles imposées par les signatures car chaque

acception est une spécialisation de B0.

Les signatures et les acceptions peuvent également être utilisées en consultation afin de guider la construction des graphes. On peut soit commencer la construction d'un graphe par une relation et regarder quels concepts voisins on a le droit de mettre, soit commencer la construction d'un graphe par un concept et regarder quelles relations voisines on a le droit de mettre.

Les signatures et les acceptions peuvent enfin guider la construction des graphes par spécialisation. Elles sont alors considérées comme des graphes basiques (base canonique) pouvant être spécialisés afin de construire de plus gros graphes.

Définitions

J. Sowa ne définit des types que par un ensemble de conditions nécessaires et suffisantes. Il peut également être intéressant de ne définir un type que partiellement, de le définir par ses conditions de reconnaissance, ou de le définir dans un contexte particulier, les définitions servant aussi bien à définir des types de concepts que des types de relations.

• Un métaconcept ***condition nécessaire*** permet de définir partiellement un type. Par exemple, une entité se trouve toujours à un endroit (Nous utilisons indifféremment les deux notations de la figure 13 pour exprimer une condition nécessaire).

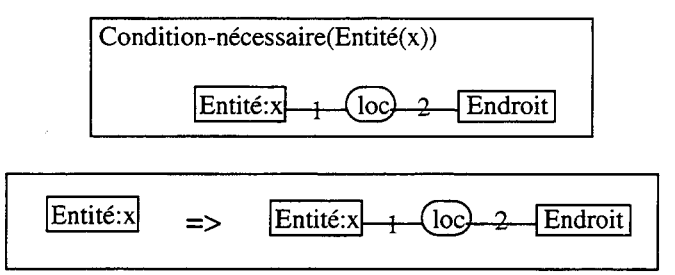


Figure 13. Une condition nécessaire du type de concept *entité*

L'interprétation logique d'une des conditions nécessaires d'un type de concept tc, notée $\lambda x G^1$ (G contient [tc:x]), est la suivante:

$$\forall x(tc(x) \rightarrow \phi(\lambda x G)) \quad (5)$$

L'interprétation logique de la condition nécessaire de la figure 13 est donc la suivante:

---

1. La notation utilisée est une abstraction unaire. Une abstraction n-aire λ a1,...an u est composée d'un graphe u appelé le corps et d'une liste de concepts génériques a1,...,an qui sont appelés les paramètres formels. Les paramètres formels sont des référents génériques distingués des autres référents dans le corps u. L'opérateur ϕ associe à une abstraction n-aire λ a1,...an u une formule de la logique des prédicats du premier ordre, où toutes les variables sont quantifiées existentiellement, sauf les ai qui sont libres ([SOW 84]).

$$\forall x(\text{Entite}(x) \rightarrow \exists y(\text{loc}(x, y) \wedge \text{endroit}(y))) \quad (6)$$

L'interprétation logique d'une des conditions nécessaires d'un type de relation binaire tr, notée λ x,yG, est la suivante:

$$\forall x \forall y(\text{tr}(x, y) \rightarrow \phi(\lambda x, yG)) \quad (7)$$

• Un métaconcept ***condition nécessaire contextuelle*** permet de définir partiellement un type dans un contexte particulier. Par exemple, un sinistre qui se produit dans la salle de contrôle est forcément grave (cf figure 14).

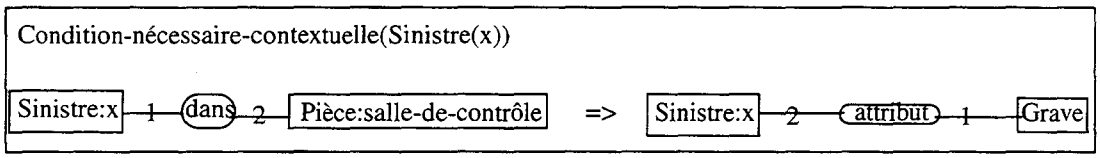


Figure 14. Une condition nécessaire contextuelle du type de concept *sinistre*

L'interprétation logique d'une des conditions nécessaires, notée λ xH, d'un type de concept tc, dans un contexte λ xG (G et H contiennent un concept [tc:x]) est la suivante:

$$\forall x(\phi(\lambda xG) \rightarrow \phi(\lambda xH)) \quad (8)$$

L'interprétation logique de la condition nécessaire de la figure 14 est donc la suivante:

$$\forall x(\text{Sinistre}(x) \wedge \text{dans}(x, \text{salledecontrole}) \wedge \text{Piece}(\text{salledecontrole}) \rightarrow$$

$$\exists y(\text{attribut}(x, y) \wedge \text{Grave}(y))) \quad (9)$$

L'interprétation logique d'une des conditions nécessaires, notée λ x,yH, d'un type de relation binaire tr, dans un contexte λ x,yG, est la suivante:

$$\forall x \forall y(\phi(\lambda x, yG) \rightarrow \phi(\lambda x, yH)) \quad (10)$$

• Un métaconcept ***condition suffisante*** permet de donner les conditions de reconnaissance d'un type. Par exemple, il suffit que deux entités soient dans la même pièce pour qu'on considère qu'elles soient proches l'une de l'autre (cf figure 15).

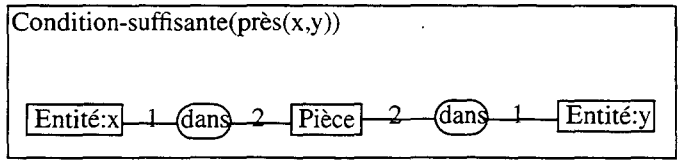


Figure 15. Une condition suffisante du type de relation *près*

L'interprétation logique d'une des conditions suffisantes, notée λxG (G contenant un concept [tc2:x]), d'un type de concept tc1 est la suivante:

$$\forall x(\phi(\lambda xG) \rightarrow tc1(x)) \quad (11)$$

L'interprétation logique d'une des conditions suffisante, notée λx,yG (G contenant deux concepts [tc1:x] et [tc2:y]), d'un type de relation binaire tr est la suivante:

$$\forall x \forall y(\phi(\lambda x, yG) \rightarrow tr(x, y)) \quad (12)$$

L'interprétation logique de la condition suffisante de la figure 15 est donc la suivante:

$$\forall x \forall y(\exists z Entite(x) \wedge Entite(y) \wedge Piece(z) \wedge dans(x, z) \wedge dans(y, z) \rightarrow pres(x, y)) \quad (13)$$

• Un métaconcept ***condition nécessaire et suffisante*** permet de définir complètement un type. Par exemple, un désastre est un sinistre dont l'attribut de gravité est grave (cf figure 16).

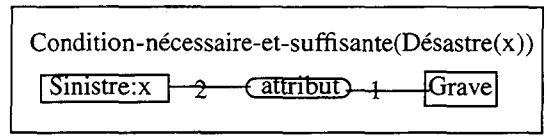


Figure 16. Une condition nécessaire et suffisante du type de concept *désastre*

L'interprétation logique d'une des conditions nécessaires et suffisantes, notée λxG (G contenant un concept [tc2:x]), d'un type de concept tc1 est la suivante:

$$\forall x(\phi(\lambda xG) \leftrightarrow tc1(x)) \quad (14)$$

L'interprétation logique de la condition nécessaire et suffisante de la figure 16 est donc la suivante:

$$\forall x(\exists y(Sinistre(x) \wedge attribut(x, y) \wedge Grave(y)) \leftrightarrow Desastre(x)) \quad (15)$$

L'interprétation logique d'une des conditions nécessaires et suffisantes, notée λx,yG (G contenant deux concepts [tc1:x] et [tc2:y]), d'un type de relation binaire tr est la suivante:

$$\forall x \forall y(\phi(\lambda x, yG) \leftrightarrow tr(x, y)) \quad (16)$$

En ce qui concerne l'héritage de tels métaconcepts, un type de concept ou de relation hérite de chacune des conditions nécessaires de ses sur-types. Ainsi, l'ensemble de toutes les conditions nécessaires d'un type est une spécialisation de chacune des conditions nécessaires de ses sur-types et de chacune des conditions nécessaires et suffisantes de ses sur-types. De la même façon, l'ensemble de toutes les conditions nécessaires contextuelles d'un type est une spécialisation de chacune des conditions nécessaires contextuelles de ses sur-types.

Nous décrivons maintenant brièvement les intérêts des métaconcepts définitions.

Par analogie aux opérations de classification automatique dans les logiques terminologiques ([BRA 85], [NEB 90]), les définitions peuvent permettre d'insérer automatiquement des types dans les hiérarchies ([LEC 95], [LEC 96]).

En outre, ces différentes définitions expriment des propriétés génériques toujours vraies c'est-à-dire des lois. Dans notre cas, ces lois peuvent être relatives à la façon de modéliser les comportements (par exemple, une action a toujours un effet sur le monde, cf annexe B) ou au domaine (par exemple, deux entités qui sont dans la même pièce sont proches l'une de l'autre).

- Les lois permettent d'économiser la saisie de certains graphes dans les assertions.
- Et elles peuvent être utilisées au niveau raisonnement pour enrichir les graphes, pour démontrer un graphe ou pour maintenir ou rétablir la cohérence des graphes.

Schémas

Les schémas donnent des formats de saisie ou de visualisation des graphes associés à un type de concept. Ils correspondent en fait à des cas plausibles d'utilisation d'un type de concept dans un graphe. Leur intérêt est documentaire et ils peuvent également faciliter la construction des graphes (ie, on copie le schéma et on le spécialise). La figure 17 donne un schéma du type de concept comportement. Ce schéma est en fait constitué de toutes les acceptions du type de concept comportement (cf annexe B).

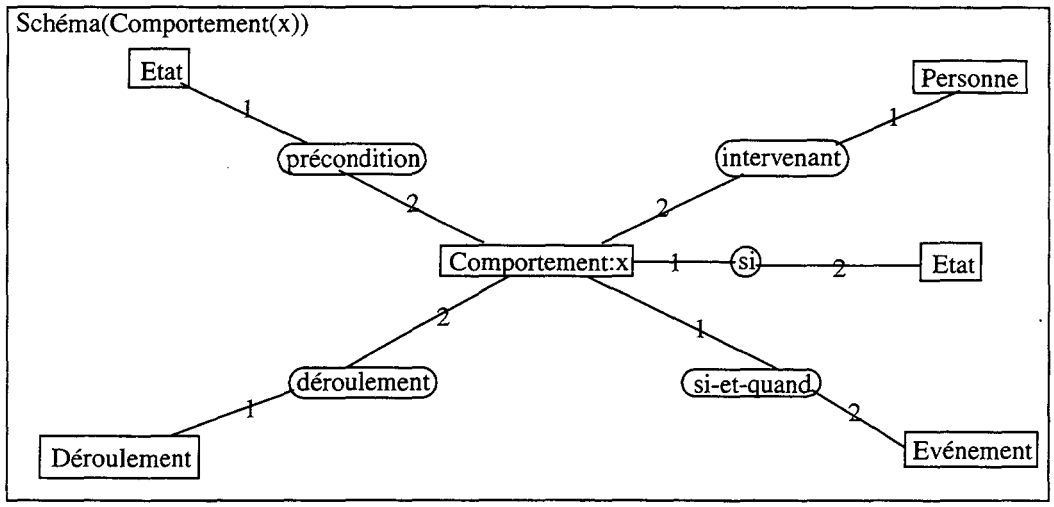


Figure 17. Un schéma du type de concept *comportement*

2.3. Conclusion

La représentation de l'ontologie comportementale et son utilisation pour modéliser des comportements nécessite de prendre en compte des extensions du formalisme de base des graphes conceptuels:

- Au niveau assertionnel, nous considérons des graphes emboîtés simples et typés afin de par exemple représenter des états et des événements.
- Au niveau terminologique, nous considérons des métaconcepts pour contraindre dans leur utilisation, documenter et définir les types de l'ontologie.

La description des propriétés du monde nécessitera de considérer d'autres extensions du formalisme de base des graphes conceptuels (notamment, la prise en compte d'une forme de négation). Nous discuterons de ces extensions dans la deuxième partie de la thèse, relative au raisonnement sur les descriptions du monde.

Chapitre 3

Les concepts comportementaux

Nous avons étudié différents langages de modélisation de connaissances comportementales: les langages de représentation de tâches en ingénierie des connaissances, les langages de représentation d'actions en intelligence artificielle et les diagrammes d'états pour la spécification de systèmes (systèmes informatiques classiques, systèmes de production). Ces langages manipulent des concepts comportementaux différents: la tâche, l'inférence, l'action, l'état, l'événement, la transition... Dans ce chapitre, nous décrivons tout d'abord brièvement ces différents concepts comportementaux classiques. Après avoir rappelé nos objectifs, nous introduisons alors les concepts principaux de notre ontologie comportementale et les positionnons vis à vis des concepts comportementaux classiques.

3.1. Tâches et inférences

En ingénierie de la connaissance ([FEN 91], [FEN 94], [SCH 88], [WIE 92]), les connaissances comportementales du modèle d'expertise d'un système à base de connaissances sont décrites au niveau tâche. Une ***tâche*** est une tâche de raisonnement (diagnostic, planification...), elle correspond à un but à atteindre au cours de la résolution d'un problème. La façon d'atteindre ce but est décrite par une méthode de résolution de problème. La méthode décompose la tâche principale en sous-tâches et spécifie le contrôle sur ces sous-tâches par du séquencement, des itérations, des branchements conditionnels.

Une tâche peut également être décomposée en ***inférences***, qui sont des étapes élémentaires du processus de résolution. Elles sont élémentaires dans le sens où leur traitement n'est pas explicité au niveau du modèle d'expertise. Elles pointent vers un traitement externe, par exemple un programme écrit dans un langage de programmation. Contrairement à une tâche, une inférence n'est ainsi pas décomposable, elle n'est décrite que par son nom et par ses paramètres d'entrée/sortie.

Dans le modèle d'expertise de la figure 18, la tâche de diagnostic médical est uniquement décomposée en inférences. Les paramètres d'entrée/sortie des inférences sont spécifiés au niveau inférence. Des rôles (représentés par des flèches en pointillé dans la figure 18) permettent d'établir des liens entre les connaissances du niveau inférence et les connaissances du domaine.

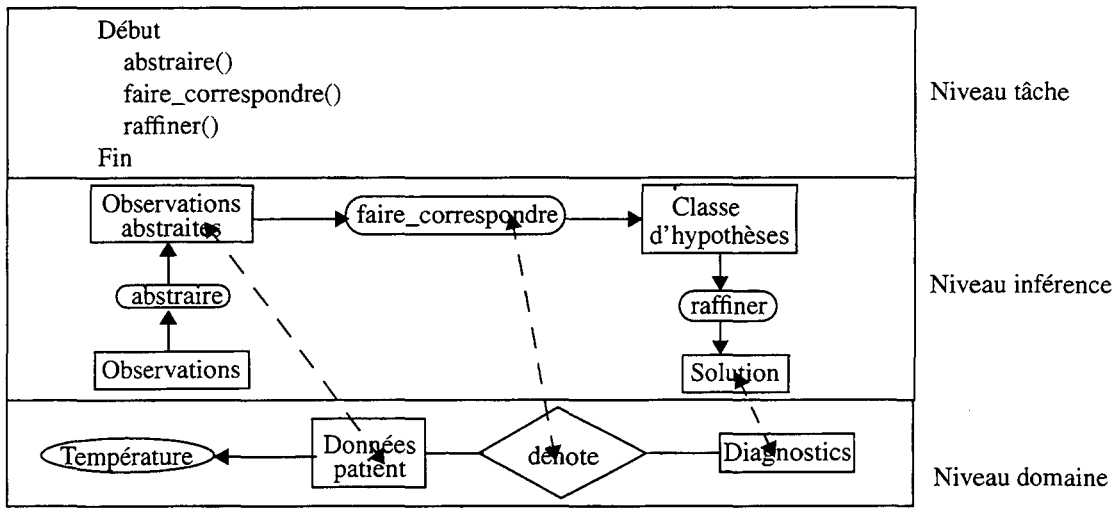


Figure 18. Un exemple de modèle d'expertise

3.2. Actions

Dans les domaines de l'intelligence artificielle relatifs à la modélisation du changement (planification [FIK 88], mise à jour [GIN 88a], projection et explication temporelle [HAN 87]), une ***action*** est définie comme un changement entre deux états e1 et e2 et est représentée par un opérateur dans lequel ce changement est explicitement décrit. Un tel opérateur est constitué d'un ensemble de préconditions qui doivent être vraies dans e1 afin que l'action soit exécutable et d'un ensemble de postconditions (encore appelées effets de l'action) qui doivent être rendues vraies en e2 quand l'action est exécutée. Ces opérateurs peuvent être logiques; les actions et les états sont alors des arguments d'une même théorie logique. Ils peuvent également être des opérateurs de changement d'état (cf figure 19); chaque état est alors représenté par une théorie logique et les effets des opérateurs décrivent les changements de formules d'une théorie à l'autre. Pour des soucis de simplification algorithmique, les opérateurs sont souvent supposés représenter des changements instantanés. Un comportement est alors un ***plan d'actions***, c'est-à-dire un ensemble totalement ou partiellement ordonné d'actions.

Préconditions:
dans(dupont,p1)
Postconditions:
dans(dupont,p2)
¬dans(dupont,p1)

Figure 19. L'action "dupont se déplace de la pièce p1 à la pièce p2" modélisée par un opérateur de changement d'état

3.3. Diagrammes d'états

Les concepts comportementaux majeurs des diagrammes d'états ([HAR 87]) sont ceux d'***événements*** et d'***états***. Un événement est un stimuli externe survenant à un instant donné. Un état est une propriété du monde (par exemple, les valeurs d'un objet dans un modèle objet) invariante sur un intervalle entre deux événements. Un changement d'état provoqué par un événement est appelé ***transition***. Un diagramme d'état est alors un graphe dont les noeuds sont des états et les arcs orientés des transitions désignées par les noms d'événements (cf figure 20). Une transition est franchie quand l'événement qui la désigne se produit. Des ***conditions*** peuvent également être utilisées comme gardes sur des transitions. Une transition gardée est franchie quand survient un événement, mais seulement si la condition de garde est vraie. Des opérations peuvent enfin être attachées aux états et aux événements; elles seront exécutées en réponse aux états et aux événements correspondants. Les opérations sont de deux types: une ***action*** est une opération instantanée tandis qu' une ***activité*** est une opération qui nécessite un certain temps. Les actions peuvent être liées aux événements ou aux entrées et sorties des états. Une activité peut être assimilée à un état ou à un diagramme d'états en entier. Il est cependant important de noter que les diagrammes d'états ne s'attachent pas à spécifier plus précisément les opérations en donnant par exemple leurs effets.

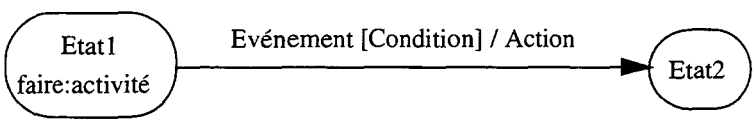


Figure 20. Exemple de notation d'une transition dans les diagrammes d'états

3.4. Nos objectifs

Nous voulons modéliser des comportements dans une organisation humaine dont nous rappelons les caractéristiques déjà explicitées dans l'introduction:

- Le domaine est composé des intervenants d'une organisation évoluant dans un environnement complexe.
- Les tâches que doivent exécuter les membres de l'organisation sont de natures diverses (transformation de l'environnement, communication, raisonnement, décision).
- Les tâches ont des durées et s'exécutent en séquence ou en parallèle.
- Les différents comportements doivent être synchronisés de façon à ce que l'organisation en tant que tout ait un comportement cohérent.
- Les réactions diverses des intervenants suivant les situations sont exprimées en conditionnant les tâches par des conditions de natures différentes (nécessaires, nécessaires et suffisantes).
- Les comportements contiennent des indéterminations pour deux raisons différentes: la connaissance de l'expert sur les comportements est incomplète ou l'expert estime

que l'intervenant est capable de prendre une décision suivant le contexte.

La phrase suivante "Quand l'incendie atteint la salle de contrôle, les pompiers doivent stopper l'incendie et évacuer les opérateurs de façon à ce qu'il n'y ait pas de blessés" est extraite des plans particuliers d'interventions de la sécurité civile ([SCI 90]). Elle résume bien les différents types de connaissance que nous devons représenter:

- des connaissances du domaine: "pompiers, opérateurs, salle de contrôle".
- des conditions et des déclenchements: l'événement "l'incendie atteint la salle de contrôle" conditionne et déclenche le comportement des pompiers.
- des tâches: "stopper l'incendie, évacuer les blessés".
- l'obligation ou le choix d'exécuter les tâches: "doit".
- les relations temporelles entre les tâches: le "et" peut indiquer une relation de précédence entre les deux tâches ou peut exprimer que leur ordre d'exécution est inconnu (indétermination).
- des objectifs: "pas de blessés"

Nous voulons en plus que les comportements soient simulables au niveau de la modélisation. Ainsi, chaque tâche doit être simulable; il faut soit décrire explicitement la façon dont elle modifie le monde, soit la décomposer en sous-tâches simulables.

3.5. Introduction des concepts comportementaux

Ainsi, dans notre cas, une ***tâche*** correspond à un but spécifique que doit atteindre un des membres de l'organisation de façon à ce l'organisation puisse atteindre un but global. Notre concept de tâche diffère de celui utilisé en ingénierie des connaissances par sa nature. Une tâche n'est en effet pas forcément un raisonnement mais peut aussi modifier l'environnement. Nous nous distinguons également des approches d'ingénierie des connaissances par les aspects divers qui sont pris en compte pour la description d'une tâche:

- sa durée et sa synchronisation temporelle vis à vis d'autres tâches
- la façon dont elle transforme le monde à divers instants (ses effets)
- un ensemble de tâches plus basiques sur lesquelles un contrôle flexible est appliqué
- un ensemble de paramètres (intervenant concerné, instruments utilisés, endroit où la tâche est exécutée...)
- un nom ou une abstraction qui résume la tâche
- des conditions d'exécution (prérequis, conditions de bon déroulement, conditions de contrôle...).

Le ***comportement*** d'un intervenant d'une organisation humaine peut être vu comme une tâche unique. Par exemple, le témoin doit déclencher l'alarme. Il peut également être vu comme un ensemble de sous-tâches plus basiques. Par exemple, le

témoin doit identifier le sinistre, décider de déclencher l'alerte, se déplacer vers le bouton d'alarme le plus proche et appuyer sur le bouton d'alarme. Nous définissons un comportement comme la tâche de plus haut niveau, il ne peut pas être sous-tâche d'une autre tâche, il représente tout ce que doit faire un intervenant dans une situation donnée.

Par analogie aux actions modélisées par des opérateurs de transformation, nous nommons ***action*** une tâche dont les effets sont représentés par des postconditions. Une action est ainsi simulable en mettant à jour un monde courant avec ses effets. Nos actions diffèrent cependant des actions de la planification atemporelle classique par la prise en compte des durées des actions et par la seule représentation des effets principaux des actions. Les effets secondaires et contextuels sont représentés de façon générique dans des lois du domaine afin de résoudre le problème de ramification (le chapitre 7 donnera plus de détails sur la prise en compte des effets secondaires et contextuels). Il est important de noter qu'une action est décomposable en actions plus détaillées, nous ne la définissons pas comme le concept de tâche de plus bas niveau.

Comme toutes les tâches doivent être simulables au niveau de la modélisation[1], les tâches qui ne sont pas des actions (les ***processus***) doivent être décrites par des actions et/ou récursivement par des processus. Nous nommons un tel ensemble d'actions et de processus un ***déroulement***.

En résumé, notre ontologie comporte deux types de tâches: les actions et les processus (cf figure 21). Nous supposons qu'un comportement est un processus (et non une action). Il n'est en effet pas évident d'expliciter sous forme de postconditions les nombreux effets d'un comportement, qui est la tâche de plus haut niveau, sans le décomposer au préalable en sous-tâches. Les autres processus permettent:

- de conditionner un ensemble de tâches (***bloc conditionnel***).
- de choisir un ensemble de tâches parmi un ensemble d'ensembles de tâches (***bloc disjonctif***).
- de rendre facultatif l'exécution d'un ensemble de tâches (***bloc optionnel***).

Un déroulement n'est pas une tâche mais est un ensemble d'actions et de processus qui sert à décrire des actions et des processus.

Notre ontologie comportementale inclut enfin les concepts d'***état***, d'***événement*** et de ***transition*** (cf figure 21), mais ce ne sont pas les concepts principaux comme c'est le cas dans les diagrammes d'états. Leur but est de spécifier les tâches. Un état est une propriété invariante du monde durant un intervalle, il exprime une condition d'une tâche. Un événement est un changement d'état qui se produit à un instant donné, il exprime une synchronisation d'une tâche. Une transition est un changement d'état qui doit être appliqué à un instant donné, elle exprime un effet d'une action. De plus, contrairement aux diagrammes d'états, nous ne représentons pas uniquement les états, événements et transitions par des noms, mais nous les représentons par des graphes

1. Notons que des concepts comportementaux tels que les inférences en ingénierie des connaissances ne nous satisfont pas car les façons dont elles modifient le monde ne sont pas décrites dans les modèles d'expertise; elles ne sont ainsi pas simulables au niveau de la modélisation.

propriétés du monde. Le chapitre 5 est consacré à la description de tels graphes.

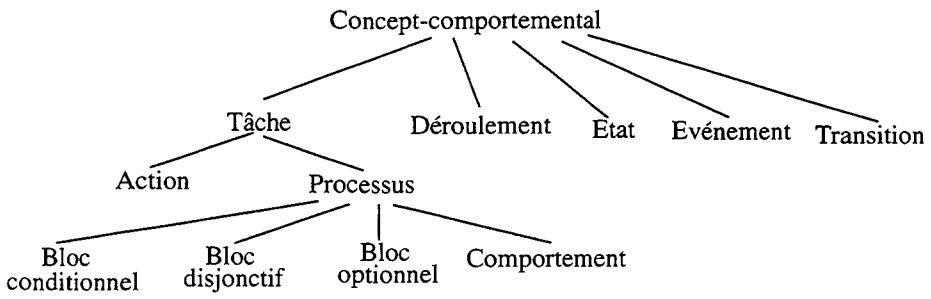


Figure 21. Hiérarchie des concepts comportementaux basiques

3.6. Conclusion

Le ***comportement*** est la tâche de plus haut niveau, il est associé à un type d'intervenant, il est décrit par un ***déroulement***.

Un ***bloc*** est une tâche appartenant à un ***déroulement***, il est décrit par au moins un ***déroulement***.

Une ***action*** est une tâche appartenant à un ***déroulement***, ses effets sur le monde sont représentés par des postconditions, elle peut en plus être décrite par un ***déroulement***.

Un ***déroulement*** est un ensemble d'***actions*** et de ***processus*** (***blocs***).

Un ***état*** est une propriété du monde invariante pendant une période de temps, il conditionne un ***comportement***, un ***bloc*** ou une ***action***.

Un ***événément*** est un changement ponctuel de la valeur de vérité d'une propriété du monde, il synchronise un ***comportement***, un ***bloc*** ou une ***action***.

Une ***transition*** est un changement ponctuel de la valeur de vérité d'une propriété du monde, elle représente les effets d'une ***action***.

Chapitre 4

Le langage de modélisation

L'ontologie comportementale proposée structure et définit les concepts comportementaux précédemment introduits et les relations entre ces concepts. Nous décrivons maintenant l'ontologie comportementale exprimée dans le formalisme des graphes conceptuels (ie, le langage de modélisation). Une version complète de l'ontologie est donnée dans l'annexe B sous la forme de hiérarchies de types et de métaconcepts associés aux différents types.

4.1. Comportements

Un ***comportement*** est le concept de tâche de plus haut niveau, il caractérise tout ce que doit faire un intervenant dans une situation donnée (ie, à un instant donné, quand certaines propriétés du monde sont établies).

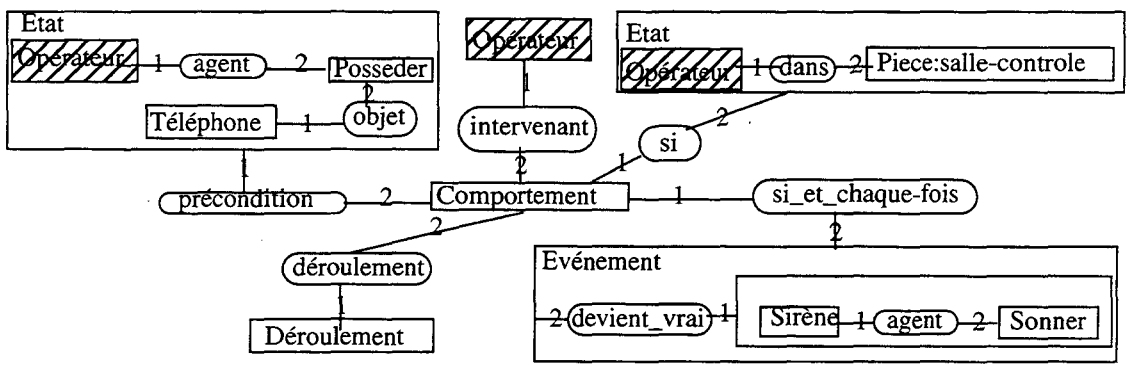


Figure 22. Un exemple de description d'un comportement

L'intervenant est un des paramètres du comportement. Nous le mettons en évidence en le représentant par un concept de type personne lié au concept comportement par une relation ***intervenant***. Un comportement est toujours appliqué par une personne; l'existence d'un concept de type personne et d'une relation intervenant est ainsi une condition nécessaire de l'existence d'un concept comportement. L'intervenant d'un comportement est unique; un concept comportement ne peut ainsi être lié qu'à une seule relation intervenant. Dans l'exemple de la figure 22, c'est un opérateur qui doit appliquer le comportement.

Nous caractérisons la situation à laquelle doit réagir l'intervenant par des ***états*** et des ***événements***. Un état est une caractérisation du monde, décrite par une propriété invariante pendant la durée de l'état. Un événement est un changement ponctuel du monde. Nous discutons de la représentation des états et des événements dans la deuxième partie de la thèse. Pour l'instant, nous supposons que la propriété décrivant un état est représentée par un graphe conceptuel emboîté dans le concept état. Nous supposons également qu'un événement est exprimé par des graphes propriétés emboîtés dans le concept événement avec deux types d'emboîtement: ***devient_vrai*** (l'état est faux juste avant l'événement mais est vrai au moment de l'événement) ou ***devient_faux*** (l'état est vrai juste avant l'événement mais est faux au moment de l'événement).

Un état qui contribue à caractériser la situation dans laquelle le comportement doit être appliqué est lié au concept comportement par une relation ***si***. Le comportement ne pourra être appliqué que si cet état est vrai, c'est une condition nécessaire de l'application du comportement[1]. Plusieurs relations si liées à un concept comportement sont interprétées comme une conjonction de relations si. Le comportement ne peut alors être appliqué que si chacun des états liés par si au comportement est vrai.

Un événement qui contribue à caractériser la situation dans laquelle le comportement doit être appliqué est lié au concept comportement par une relation ***si_et_la_première_fois*** ou une relation ***si_et_chaque_fois***. Dans le premier cas, le comportement ne pourra être appliqué qu'une seule fois par un seul intervenant alors que dans le deuxième cas, le comportement peut être appliqué plusieurs fois par un ou plusieurs intervenants. Dans les deux cas, le comportement ne pourra être appliqué que si l'événement se produit et quand l'événement se produit. L'événement est à la fois une condition nécessaire et une synchronisation temporelle de l'application du comportement. Plusieurs relations si_et_la_première_fois (respectivement si_et_chaque_fois) sont interprétées comme une conjonction de relations si_et_la_premiére_fois (respectivement si_et_chaque_fois). Un concept comportement ne peut pas à la fois être lié à une relation si_et_la_premiére_fois et à une relation si_et_chaque_fois. En revanche, il doit au moins être lié à une de ces deux relations afin d'être synchronisé dans le temps.

Chaque état et chaque événement évoqué précédemment constitue une condition nécessaire de l'application du comportement. La conjonction de l'ensemble de ces états et de ces événements constitue une condition nécessaire et suffisante de l'application du comportement. Nous nommons de tels états et événements des conditions de contrôle. Par exemple, le comportement de la figure 22 doit être appliqué par un opérateur si et seulement si celui-ci est dans la salle de contrôle et la sirène se met à sonner.

Si ses conditions de contrôle sont vérifiées, un comportement doit être appliqué. En revanche, rien ne garantit qu'il pourra être appliqué correctement. Pour ce

1. Attention, il faut distinguer condition nécessaire de l'existence d'un concept comportement, qui est un métaconcept définissant partiellement le type de concept (cf annexe B pour voir toutes les conditions nécessaires de l'existence d'un concept comportement), d'une condition nécessaire de l'application d'un comportement, qui est représentée par un état ou un événement dans la modélisation d'un comportement.

faire, il faut qu'un certain nombre de conditions supplémentaires soient vérifiées. Ces conditions sont des préconditions (ou prérequis), qui doivent être vraies juste avant l'application du comportement. Elles sont des conditions nécessaires de l'application du comportement, mais elles ne contrôlent pas l'application du comportement. Par exemple, pour appliquer correctement le comportement de la figure 22, l'opérateur concerné doit posséder un téléphone. Les préconditions d'un comportement sont représentées par des états liés au comportement par des relations ***précondition***.

Nous avons vu dans le chapitre précédent qu'un comportement est un processus, c'est à dire que la façon dont il modifie le monde est décrite par un ***déroulement***. Ce déroulement est unique et est représenté par un concept déroulement lié au concept comportement par une relation ***déroulement***. Un déroulement est un ensemble d'actions (ie, un ensemble de spécialisations de modèles d'actions) et de processus. Ces actions et processus sont synchronisés dans le temps par des relations temporelles et/ou des événements. Ils sont obligatoires par défaut mais peuvent aussi être conditionnés. Un déroulement peut enfin contenir des indéterminations. La suite de cette partie est consacrée à la description des déroulements: les actions, les synchronisations temporelles, les branchements conditionnels (blocs conditionnels) et les indéterminations (notamment blocs disjonctifs et blocs optionnels).

4.2. Actions

Notre premier objectif est de modéliser une action de façon à ce qu'elle soit simulable. Toute action est simulable car nous définissons le concept d'action comme une tâche dont les effets sont explicitement représentés. L'action est alors simulée (ou exécutée) en mettant à jour un monde courant avec les effets.

Notre deuxième objectif est de modéliser une action de manière réaliste. En ce qui concerne la description des effets, nous nous attachons ainsi à résoudre le problème de persistance et le problème de ramification. En ce qui concerne la description des conditions d'exécutions, nous nous attachons à résoudre le problème de qualification. Enfin, au niveau temporel, nous prenons en compte des actions qui durent, c'est-à-dire des actions pouvant être associées à des intervalles temporels.

Notre troisième objectif est de pouvoir spécifier les caractéristiques d'un type d'action dans des modèles génériques. Au cours de la simulation, ces modèles génériques seront spécialisés en accord, non seulement avec les descriptions des actions dans les déroulements, mais aussi avec les descriptions courantes du monde.

4.2.1. Problème de persistance

Le problème de persistance ou "Frame Problem" concerne la description de ce qui ne change pas dans le monde quand une action est exécutée. Il a été identifié dans les travaux de J.Mc Carthy et P.Hayes sur le calcul de situations ([MCC 69]).

J.Mc Carthy et P.Hayes représentent les actions dans une logique situationnelle où les états (ou situations) et les actions sont les termes du langage. Dans cette logique, la formule Holds(p,e) signifie que la propriété p est vraie dans l'état e et la fonction Result(a,e) est équivalente au résultat de l'exécution de l'action a à partir de l'état e. La formule 17 représente ainsi l'action "dupont va de la pièce p1 à la pièce p2".

$$\forall e(Holds(dupont-dans-p1, e) \rightarrow$$

$$Holds(dupont-dans-p2, Result(dupont-va-de-p1-a-p2, e))) \quad (17)$$

La formule 17 ne permet cependant pas de déduire que toutes les propriétés du monde, qui ne sont pas concernées par l'action, ne changent pas. J.Mc Carthy et P.Hayes proposent alors d'introduire des formules de persistance ou "Frame Axioms" qui indiquent ce qui ne change pas dans le monde quand une action est exécutée.La formule 18 est une formule de persistance qui signifie que le téléphone de la pièce p2 ne change pas de position quand dupont va de la pièce p1 jusque dans la pièce p2.

$$\forall e(Holds(telephone-dans-p2, e) \rightarrow$$

$$Holds(telephone-dans-p2, Result(dupont-va-de-p1-a-p2, e))) \quad (18)$$

Dans un monde complexe décrit par un grand nombre de propriétés, il est cependant contraignant de devoir définir toutes ces formules de persistance. Le Frame Problem n'est ainsi pas résolu efficacement par une telle approche.

Les approches par opérateurs de changement d'état, telles que celle de STRIPS ([FIK 88]), résolvent le Frame Problem en supposant que seuls les changements du monde sont décrits. Ces changements sont alors représentés par des changements de formules entre deux théories de la logique des prédicats du premier ordre. Une liste d'ajouts (ou add-list) contient les formules qui seront vraies à la suite de l'exécution de l'action. Une liste de retraits (ou del-list) contient les formules qui seront fausses à la suite de l'exécution de l'action. Les listes d'ajouts et de retraits de l'action dupont va de la pièce p1 à la pièce p2 seront alors les suivantes:

- add-list: dans(dupont,p2)
- del-list: dans(dupont,p1)

Dans notre cas, nous faisons une hypothèse similaire à l'hypothèse de STRIPS pour résoudre le Frame Problem. Les propriétés du monde persistent lors de l'exécution d'une action sauf si le contraire est dit explicitement dans la représentation de l'action. Les changements du monde sont exprimés par des changements de formules entre deux théories exprimées avec des graphes conceptuels. Pour expliciter les changements, nous utilisons le concept de ***transition.*** Contrairement à un événement, une transition n'est pas un changement du monde qui va être testé par l'algorithme de simulation, mais c'est un changement du monde qui va être appliqué par l'algorithme de simulation. Nous discutons plus précisément la représentation des transitions dans la seconde partie de la thèse. Pour l'instant, nous supposons qu'une transition est exprimé par des états emboîtés dans le concept transition avec deux types d'emboîtements: ***devient_vrai*** et ***devient-faux***. Par exemple, la figure 23 exprime la transition dupont entre dans la pièce p2.

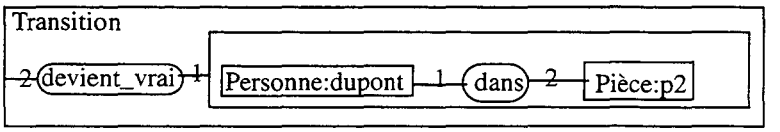


Figure 23. Un exemple de transition

4.2.2. Problème de ramification

Le problème de ramification concerne la description de ce qui change dans le monde quand une action est exécutée.

Les opérateurs à la STRIPS ne résolvent pas efficacement le problème de ramification car toutes les formules qui deviennent vraies et toutes les formules qui deviennent fausses doivent être explicitement définies. De plus, pour prendre en compte des effets contextuels (ie, les effets dépendant de la description initiale du monde avant l'exécution de l'action), il doit y avoir autant d'opérateurs que de mondes initiaux possibles. Si on considère que deux entités qui sont dans la même pièce, sont proches l'une de l'autre, l'action dupont va de p1 à p2 est maintenant représentée par quatre opérateurs différents. Les conditions d'application de ces opérateurs sont décrites dans les préconditions (cf figure 24).

Préconditions:	**Préconditions:**	**Préconditions:**	**Préconditions:**
p1-vide	p1-vide	p2-vide	dans(x,p1)
p2-vide	dans(x,p2)	dans(x,p1)	dans(y,p2)
dans(dupont,p1)	dans(dupont,p1)	dans(dupont,p1)	dans(dupont,p1)
Add-list:	**Add-list:**	**Add-list:**	**Add-list:**
dans(dupont,p2)	dans(dupont,p2)	dans(dupont,p2)	dans(dupont,p2)
Del-list:	pres(dupont,x)	**Del-list:**	pres(dupont,y)
dans(dupont,p1)	**Del-list:**	dans(dupont,p1)	**Del-list:**
	dans(dupont,p1)	pres(dupont,x)	dans(dupont,p1)
			pres(dupont,x)

Figure 24. Multiplication des opérateurs STRIPS en cas d'effets contextuels

D'autres approches ([COR 94], [GIN 88a], [WIN 88]) résolvent le problème de ramification en introduisant des lois du domaine. Les lois du domaine sont représentées par des formules de la logique des prédicats du premier ordre. Une méta-information sur ces lois indique qu'elles ne peuvent pas changer. Cette fois, seuls les effets principaux sont représentés dans les opérateurs. Les effets secondaires et contextuels sont calculés à partir de ce petit nombre d'effets et des lois du domaine. Les quatre opérateurs de la figure 24 se résument alors en un seul opérateur et en une loi du domaine (cf figure 25). Le calcul de la théorie logique après l'exécution de l'action est connu en intelligence artificielle comme le problème de la mise à jour.

Préconditions:
dans(dupont,p1)
Add-list:
dans(dupont,p2)
Del-list:
dans(dupont,p1)

$\forall x \forall y \forall z(dans(x, z) \wedge dans(y, z) \leftrightarrow pres(x, y))$

Figure 25. Représentation d'une action avec prise en compte de lois du domaine

Dans notre cas, seuls les effets principaux sont donnés dans la représentation d'une action. Ils sont exprimés par des concepts transitions. Les lois du domaine sont elles explicitées par des métaconcepts (condition nécessaire, condition suffisante, condition nécessaire contextuelle, condition nécessaire et suffisante) quand ils sont associés à des types du domaine (personne, instrument, endroit...) et non à des types comportementaux (action, état, événement...). La figure 26 donne un exemple d'une telle loi du domaine "Deux entités sont proches l'une de l'autre si et seulement si elles se trouvent dans la même pièce". Le chapitre 7 de la thèse est entièrement consacrée au principe de mise à jour reposant sur ces deux représentations des effets et des lois du domaine.

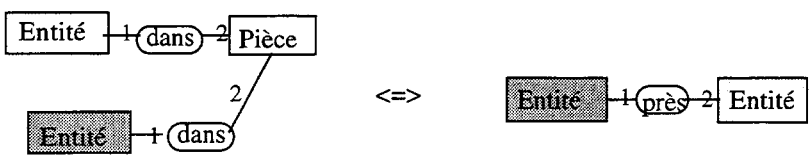


Figure 26. Un exemple de loi du domaine représentée avec des graphes conceptuels

En plus d'une représentation unique des effets principaux, nous nous attachons à résoudre le problème de ramification grâce à la simulation. On peut en effet perturber la simulation et observer les conséquences d'effets supplémentaires (transitions supplémentaires) qu'on aurait oubliés de mentionner. Les résultats de simulation montrent alors si les modèles de comportements sont suffisamment robustes pour remédier au problème de ramification.

4.2.3. Problème de qualification

Le problème de qualification est lié à la description des préconditions d'une action (ie, les conditions devant être vraies pour que l'action soit exécutable). Le nombre de préconditions d'une action est important et nous ne savons pas quand il est raisonnable de supposer que l'action peut être exécutée. Par exemple, si on a oublié de préciser que la pièce p2 devait être accessible pour que dupont puisse y entrer, l'exécution du déplacement de dupont va aboutir à un monde incohérent dans lequel p2 n'est pas accessible alors que dupont est dans p2.

Très peu d'approches ont été proposées pour la résolution du problème de qualification. Ginsberg ([GIN 88b]) a tenté de le résoudre en définissant un ensemble

spécifique de lois du domaine (les lois de qualification) qui ne servent pas à calculer les effets secondaires et contextuels des actions mais qui caractérisent les lois du monde à ne pas violer. Si en exécutant une action, l'état résultant viole les lois de qualification, c'est que l'action n'était en fait pas qualifiée, elle n'aurait pas dû être exécutée. Dans l'exemple du déplacement de dupont, une loi de qualification est exprimée par la formule 19.

$$\forall p \forall x(\neg accessible(p) \rightarrow \neg dans(x, p)) \quad (19)$$

Dans notre cas, cette approche de Ginsberg pourrait fournir une autre façon de qualifier les actions. Elle présenterait cependant un inconvénient au niveau de la simulation, car il faudrait exécuter l'action avant de voir si elle est exécutable.

Comme pour le problème de ramification, nous testons les conséquences du problème de qualification grâce à la simulation. Il est ainsi possible de perturber la simulation en empêchant certaines actions de s'exécuter correctement. Les résultats de simulation montrent alors ce qui se passerait si des préconditions avaient été oubliés dans les actions et si ces préconditions n'étaient pas vérifiées. Ils montrent ainsi si les modèles de comportement sont suffisamment robustes pour remédier au problème de qualification.

4.2.4. Prise en compte de la durée

Dans les approches décrites précédemment, une action est supposée s'exécuter instantanément. Les seuls composants des opérateurs sont ainsi des préconditions (vraies juste avant l'exécution de l'action) et des postconditions (vraies juste après l'exécution de l'action). L'attribution d'une durée aux actions nous amène à augmenter le nombre de composants des opérateurs. La figure 27 donne un exemple de représentation d'action avec certains de ces composants.

Des conditions nécessaires à la bonne exécution de l'action peuvent être vérifiées juste avant l'action (préconditions) mais aussi pendant toute la durée de l'action; ce sont alors des conditions de bon déroulement.Une précondition est représentée par un état lié au concept d'action par une relation ***précondition*** tandis qu'une condition de bon déroulement est représentée par un état lié au concept d'action par une relation ***condition_bon_déroulement***. Plusieurs relations précondition (respectivement condition_bon_déroulement) sont interprétées comme une conjonction de relations précondition (respectivement condition_bon_déroulement).

Les effets peuvent être pris en compte à divers instants de l'exécution de l'action: au début de l'action pendant l'action, à la fin de l'action ou après l'action.Ils sont représentés par des transitions liées aux actions par les relations suivantes: ***effet_début***, ***effet_intermédiaire***, ***effet_fin*** et ***effet_retardé***. De par la définition d'une action, l'existence d'une transition et d'une relation de type effet est une condition nécessaire à l'existence d'un concept action.

Une action peut également être décomposée en sous-actions plus précises. L'action est alors décrite par un déroulement unique, qui est lié au concept d'action par une relation ***déroulement***. En ce qui concerne la simulation, l'intérêt de ce déroulement pourra être la simulation de l'exécution de l'action à différents niveaux de granularité.

La façon dont une action est exécutée peut enfin être abstraite par une propriété invariante pendant toute la durée de l'action. Une telle ***abstraction*** est décrite

par un graphe emboîté dans le concept action.

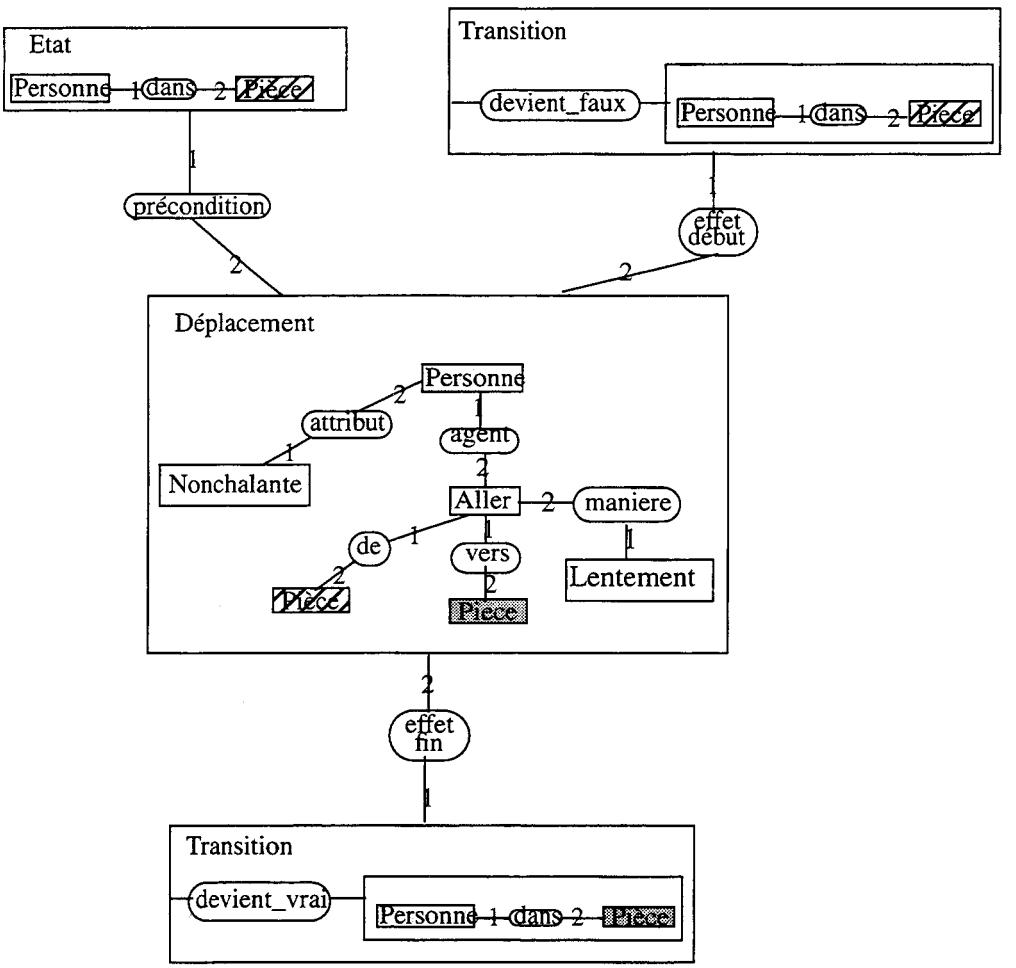


Figure 27. Une représentation de l'action déplacement

4.2.5. Les modèles génériques d'actions

Des actions de même type vont souvent devoir être exécutées plusieurs fois par différents intervenants, à différents instants ou avec divers instruments. Il est ainsi avantageux de spécifier de façon générique les préconditions, les conditions de bon déroulement, les déroulements et les effets de ces actions dans des modèles, ceci, afin d'éviter de les spécifier à maintes reprises dans les déroulements.

Les modèles génériques sont appliqués par l'algorithme de simulation afin qu'il puisse récupérer une spécialisation des préconditions, des conditions de bon déroulement, des déroulements et des effets des actions. Appliquer un modèle nécessite de spécialiser ce modèle. La façon dont les modèles sont spécialisés est originale car les spécialisations se font en accord, non seulement avec les descriptions des actions dans les déroulements, mais aussi avec les descriptions courantes du monde.

Nous décrivons tout d'abord la représentation d'une action dans un déroulement, puis la représentation d'un modèle générique d'action. Nous introduisons alors comment un modèle générique est appliqué vis à vis d'une action et d'un monde courant. Finalement, nous donnons les propriétés d'héritage des modèles génériques.

Description d'une action dans un déroulement

Dans un déroulement, une action est décrite par un graphe emboîté dans le concept action (ie, une abstraction de l'action). Dans l'action de la figure 28, ce graphe se traduit en langage naturel par la phrase "La personne dupont va à une certaine vitesse de la pièce p1 vers une pièce dans laquelle il y a un téléphone".

L'abstraction d'une action contient un concept de type verbe d'action et un ensemble de concepts liés par des relations casuelles[1] au concept de type verbe d'action. Ces derniers concepts sont les ***paramètres de l'action***. Les paramètres peuvent être:

- instanciés avec des référents individuels
- génériques spécialisés par des graphes condition (dans l'exemple de la figure 28, il y a un téléphone dans la pièce vers laquelle se dirige dupont)
- génériques non spécialisés par des graphes conditions

Nous distinguons les paramètres qui doivent être instanciés dans l'action avant l'application du modèle (***paramètres à instancier***) des autres paramètres. Les paramètres à instancier doivent exister dans le monde au moment du début de l'action, ce sont des ***paramètres préexistants***. Les autres paramètres qui ne doivent pas obligatoirement être instanciés dans l'action peuvent être préexistants ou non. Il y a ainsi trois types de paramètres: les paramètres à instancier préexistants (i_p_paramètres), les paramètres à ne pas instancier préexistants (ni_p_paramètres) et les paramètres à ne pas instancier non préexistants (ni_np_paramètres). Ces paramètres sont distingués dans l'action par des types différents de relations casuelles: i_p_relation casuelle, ni_p_relation_casuelle et ni_np_relation_casuelle (cf annexe D).

Dans l'exemple de la figure 28, le concept personne et les concepts pièce sont des i_p_paramètres tandis que le concept vitesse est un ni_np_paramètre.

Dans l'exemple de la figure 29, le concept personne et le concept boîte_outil sont des i_p_paramètres. En revanche, le concept outil est un ni_p_paramètre. La personne dupont va choisir la boîte à outils avant de choisir un outil dans la boîte. Comme l'outil choisi dépend de la boîte choisie, l'outil doit être instancié après la boîte (ie, après l'application du modèle).

---

1. Les relations casuelles lient un verbe de situation et une entité, elles sont issues des cas sémantiques de C.J. Fillmore [FIL 68] et caractérisent ainsi les relations de sens qui existent entre les noms et le verbe dans une phrase simple. Nous considérons les relations casuelles comme des relations du domaine (cf partie II).

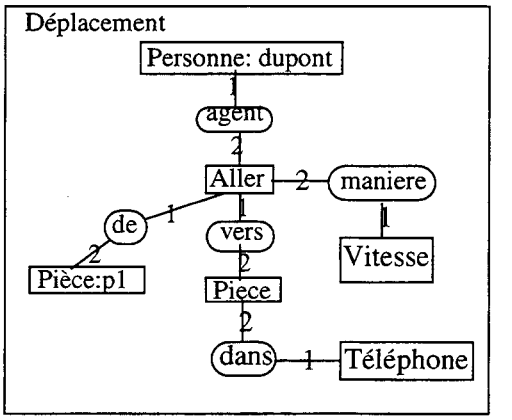


Figure 28. Une action dans un déroulement

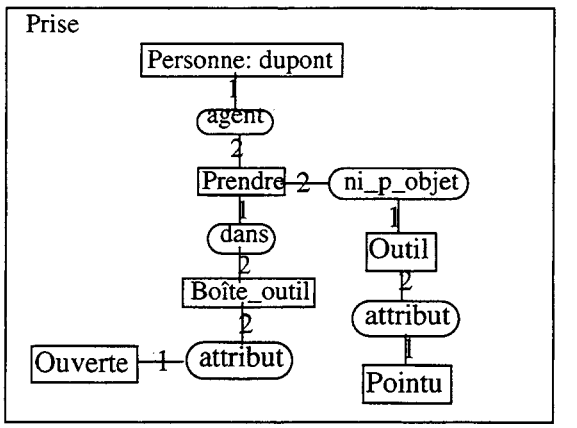


Figure 29. Une action dans un déroulement comportant un ni_p_paramètre

Description d'un modèle générique d'action

Un modèle générique d'un type d'action est une ***condition nécessaire contextuelle*** de ce type (figure 30). Il est à noter que nous préférons cependant utiliser une notation simplifiée comme celle de la figure 27 pour représenter un tel modèle.

Le ***contexte*** est un graphe emboîté dans le concept action (ie, une abstraction de l'action). Ce graphe contient un concept de type verbe d'action et un ensemble de concepts liés par des relations casuelles au concept de type verbe d'action. Ces derniers concepts sont les ***paramètres du modèle***. Les paramètres d'un modèle sont génériques, spécialisés ou non par un graphe condition. Comme dans le cas des actions, nous distinguons trois types de paramètres dans les paramètres d'un modèle.

La ***condition nécessaire*** est une conjonction de préconditions, conditions de

bon déroulement, déroulement et effets. Des liens de coréférence peuvent lier des concepts génériques des préconditions, conditions de bon déroulement, déroulement et effets à certains concepts génériques de l'abstraction du modèle.

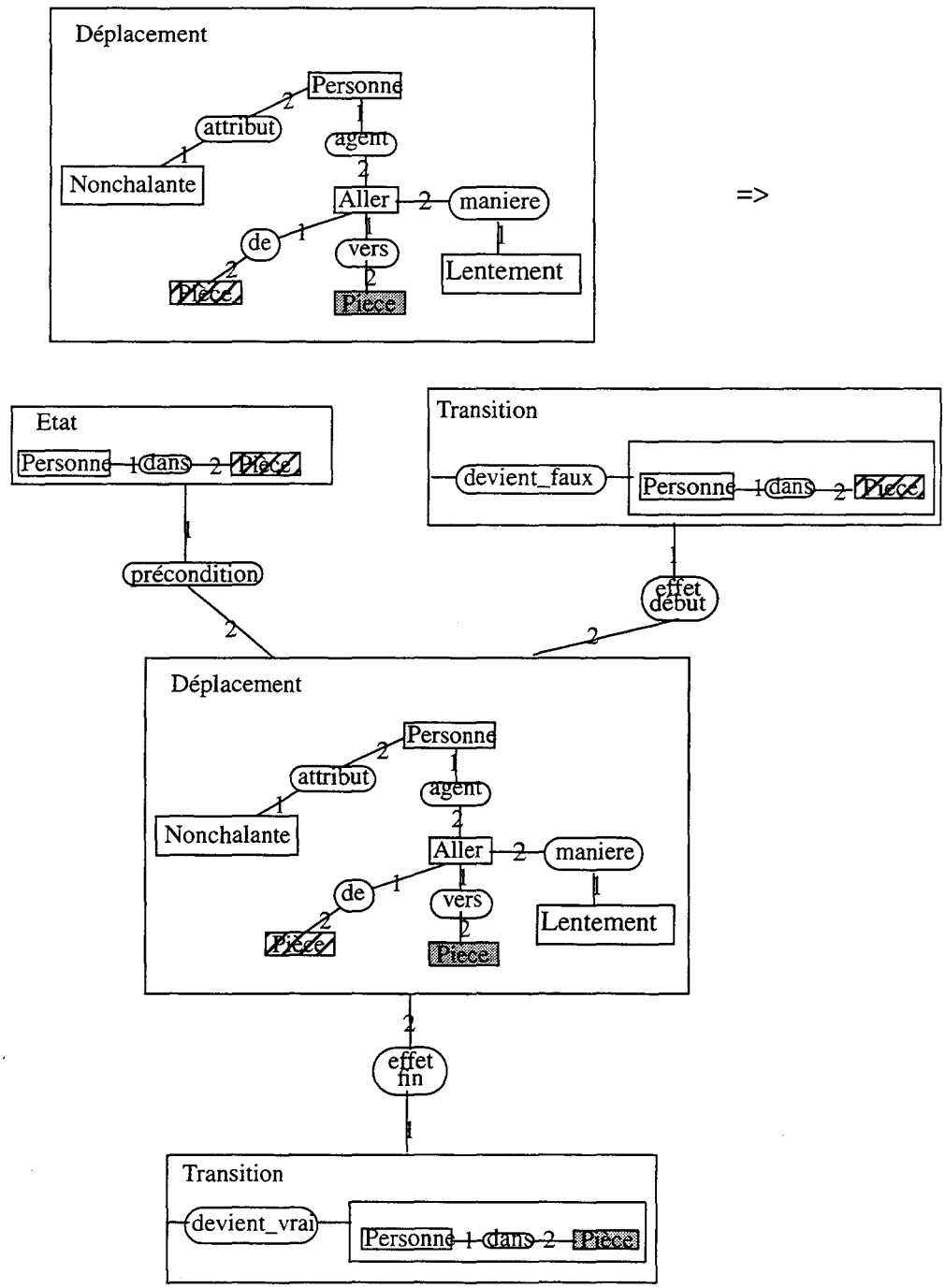


Figure 30. Modèle générique d'un type d'action = condition nécessaire contextuelle du type

Un type d'action n'a pas obligatoirement un seul modèle mais peut être associé à une ***disjonction de modèles*** (ie, une disjonction de conditions nécessaires contextuelles). Par exemple, la figure 31 donne un second modèle générique du type déplacement.

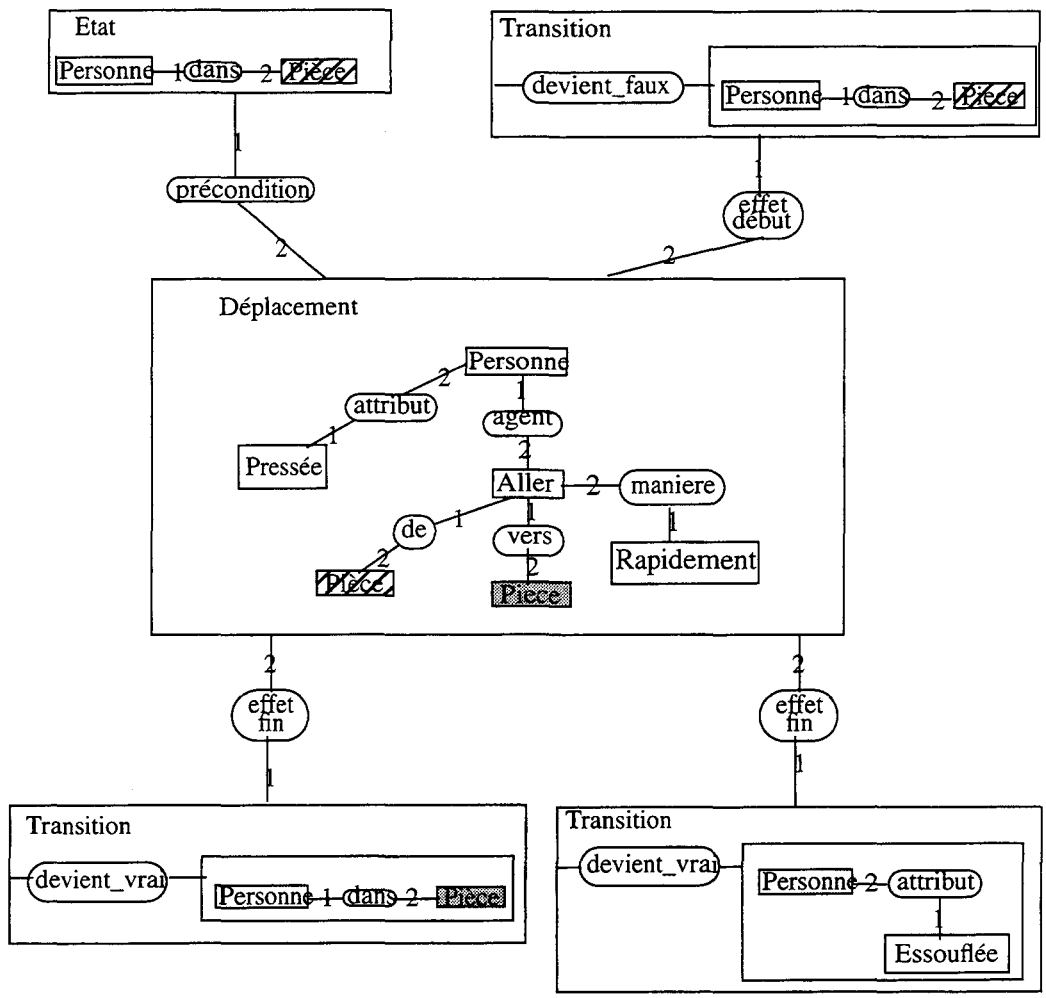


Figure 31. Un second modèle générique du type déplacement

Application d'un modèle générique

Appliquer un modèle générique (figure 27) sur une action d'un déroulement (figure 28) par rapport à une description courante du monde (figure 32) consiste à retrouver une spécialisation des préconditions, conditions de bon déroulement, déroulements et effets de l'action. Pour ce faire, il faut

- Instancier les i_p_paramètres de l'action vis à vis du monde[1]

1. Les ni_p_paramètres de l'action seront instanciés au moment de la vérification des préconditions. Si des graphes conditionnent ces paramètres, ils seront également testés au moment de la vérification des préconditions.

- il faut instancier ces paramètres de façon à ce qu'ils existent dans le monde et de façon à ce que les graphes qui les conditionnent soient vrais dans le monde (si ce n'est pas possible, l'algorithme de simulation détecte un problème, ce problème est que l'action n'est pas instanciable).

- Choisir un modèle ayant le même type que l'action.
- Vérifier si le modèle choisi correspond bien à l'action instanciée et au monde

  - il faut unifier les paramètres de l'action instanciée et du modèle afin d'obtenir un modèle spécialisé (si ce n'est pas possible, le modèle choisi n'est pas applicable, il faut choisir un autre modèle[1]).

  -il faut vérifier si les graphes qui conditionnent les i_p_paramètres du modèle spécialisé sont vrais dans le monde (si ce n'est pas possible, le modèle choisi n'est pas applicable, il faut choisir un autre modèle).

  -il faut vérifier si les graphes qui conditionnent les ni_np_paramètres du modèle spécialisé s'unifient avec les graphes qui conditionnent les ni_np_paramètres de l'action instanciée (si ce n'est pas possible, le modèle peut être appliqué mais l'utilisateur doit être prévenu de la différence entre l'action et le modèle d'action).

- Propager les liens de coréférences de l'abstraction du modèle spécialisé vers les préconditions, conditions de bon déroulement, déroulement et effets de l'action.

L'application des modèles peut amener de l'indétermination au cours de la simulation si un modèle peut être spécialisé de différentes façons ou si plusieurs modèles peuvent être appliqués en même temps.

La figure 33 donne l'exemple de l'application du modèle de la figure 27 sur l'action de la figure 28 vis à vis de la description du monde de la figure 32.

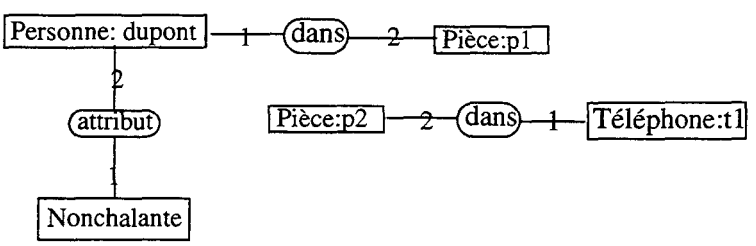
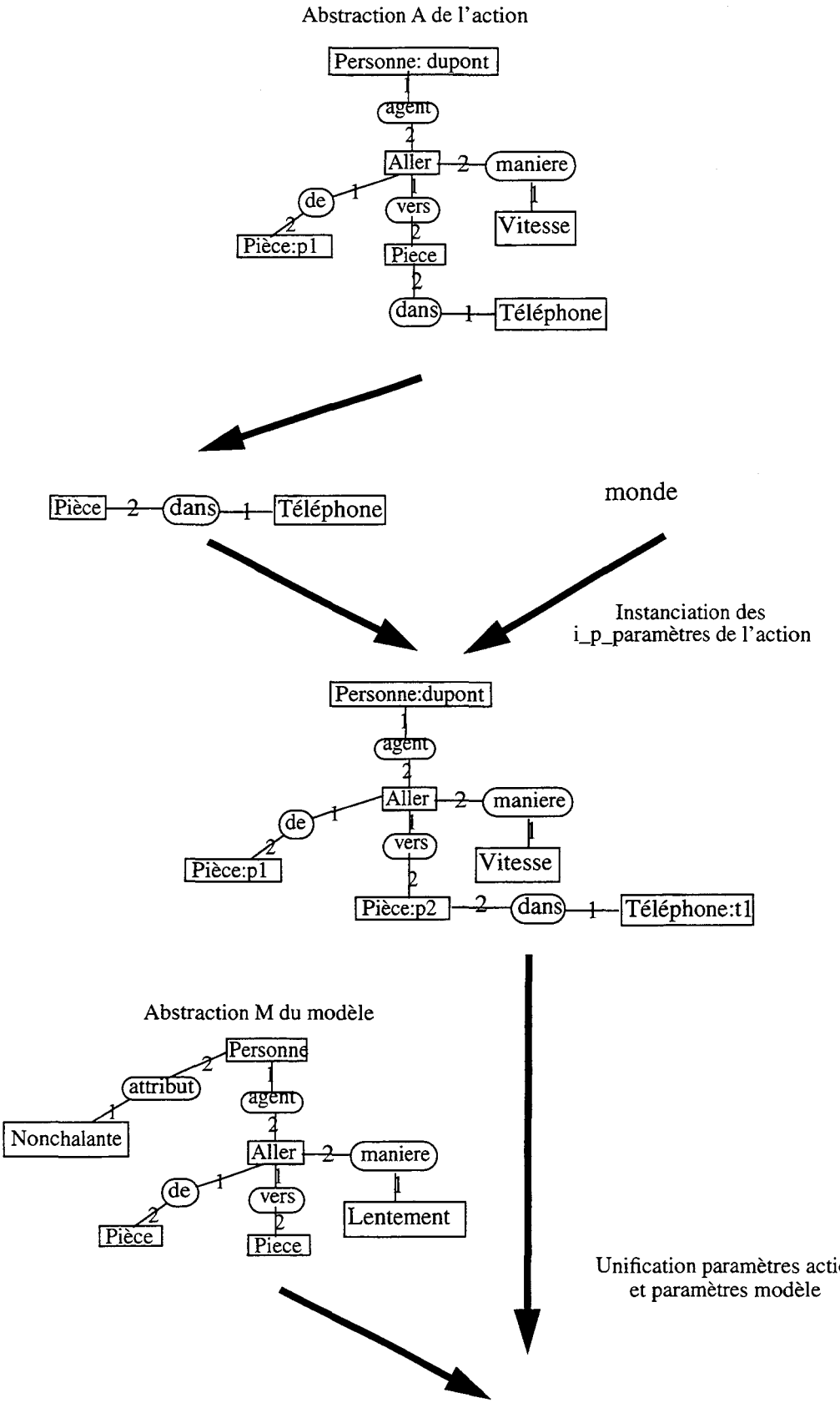


Figure 32. Une description du monde

1. Si aucun modèle n'est applicable, l'algorithme de simulation détecte un problème.


Abstraction A de l'action
Personne: dupont
agent
Aller
maniere
de
vers
Vitesse
Pièce:p1
Piece
dans
Téléphone
Pièce
dans
Téléphone
monde
Instanciation des
i_p_paramètres de l'action
Personne:dupont
agent
Aller
maniere
de
vers
Vitesse
Pièce:p1
Pièce:p2
dans
Téléphone:t1
Abstraction M du modèle
Personne
attribut
agent
Nonchalante
Aller
maniere
de
vers
Lentement
Pièce
Piece
Unification paramètres action
et paramètres modèle

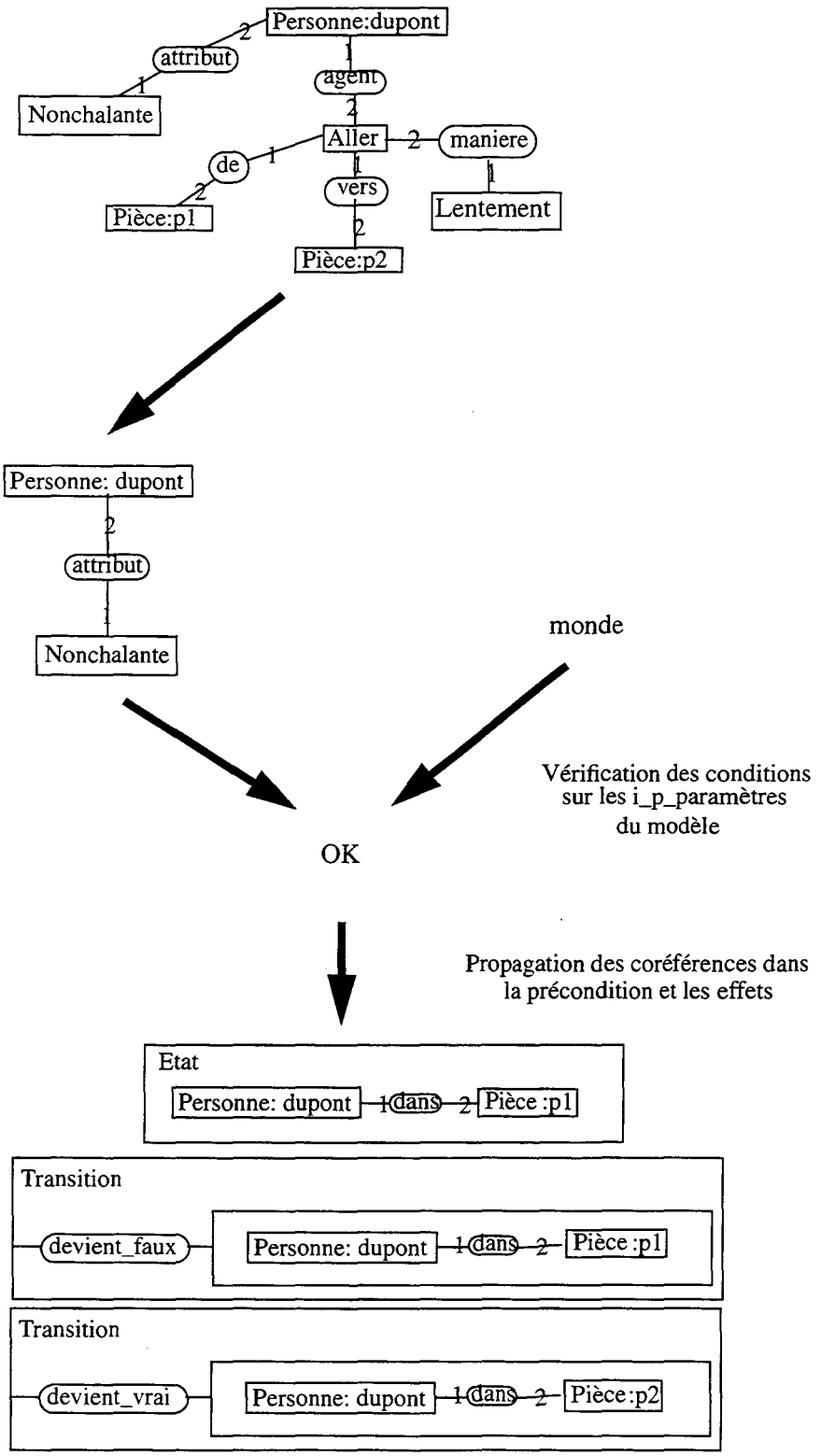


Figure 33. Exemple d'application d'un modèle d'action

Héritage des modèles génériques d'action

L'ensemble de tous les modèles d'un type d'action admet des propriétés d'héritage similaires à celles d'une disjonction de conditions nécessaires. Ainsi, chaque modèle d'un type d'action est une spécialisation d'au moins un modèle de chacun de ses sur-types. Par exemple, le modèle de déplacement de la figure 27 est une spécialisation de l'unique modèle du type action (figure 34).

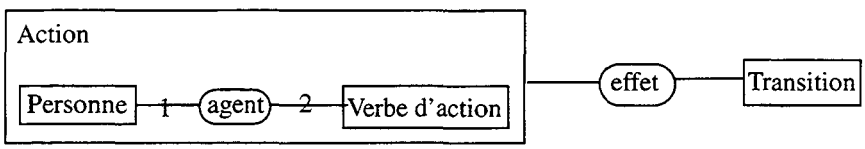


Figure 34. L'unique modèle du type action

4.3. Synchronisations temporelles

Nous considérons l'information sur la durée des tâches et les positions temporelles des tâches uniquement de façon qualitative. La connaissance de l'expert sur les comportements n'est pas complète et exiger de lui l'attribution de durées numériques aux tâches serait un travail d'analyse long et difficile. Dans un premier temps, l'expert pense à des comparaisons de durée de tâches (plus long, plus court) ou à la position relative des tâches (avant, pendant, après...). Ce sont de telles informations qualitatives que nous voulons simuler et donc représenter. Pour ce faire, nous exploitons deux types de synchronisations temporelles: des relations temporelles symboliques entre les actions et processus et des événements qui déclenchent les débuts et fins d'actions.

4.3.1. Relations temporelles

Nous ne considérons pas des actions instantanées, mais des actions qui durent. Au niveau temporel, une action peut ainsi être vue comme un intervalle temporel ou comme un couple d'instants (le début et la fin de l'action). Les positions relatives des actions peuvent alors être données par des relations entre intervalles ou des relations entre instants. Allen ([ALL 83]) propose un ensemble de treize relations primitives donnant toutes les relations temporelles symboliques possibles entre intervalles (cf figure 35). Ces relations sont mutuellement exclusives dans le sens ou une seule de ces relations est vraie entre deux intervalles. Les relations temporelles symboliques, primitives, mutuellement exclusives entre instants ([MCD 82]) sont elles au nombre de trois (cf figure 36).

Relation	Relation inverse	Signification
Egal	Egal	-------- --------

Relation	Relation inverse	Signification
Avant	Après	--------- ---------
Touche	Touché-par	---------- ----------
Recouvre	Recouvert-par	---------- ---------
Débute	Débuté-par	---------- ----------
Contenu-dans	Contient	------- ----------
Termine	Terminé-par	---------- ----------

Figure 35. Relations temporelles symboliques entre intervalles

Relation	Relation inverse	Signification
Avant	Après	. .
Egal	Egal	.

Figure 36. Relations temporelles symboliques entre instants

Nous proposons de synchroniser les actions avec des relations temporelles entre intervalles plutôt qu'avec des relations entre instants pour deux raisons:

- La représentation est plus simple. L'utilisation de relations temporelles entre instants nécessiterait en effet de manipuler des concepts début d'action et fin d'action plutôt qu'uniquement le concept d'action.
- La représentation est également plus expressive si l'on doit utiliser des relations temporelles imprécises. Par exemple, il est impossible d'exprimer la relation avant ou après entre intervalles avec des relations entre instants[1] ([GHA 89]).

Plusieurs relations temporelles liées à une action sont interprétées comme une conjonction de relations temporelles. Dans l'exemple de la figure 37, deux cas sont possibles, a1 termine avant a2 ou a2 termine avant a1. Dans les deux cas, a3 est après a1 et a2.

1. Les algorithmes de vérification statique de consistance sont cependant plus complexes dans le cas d'une représentation avec des intervalles ([VIL 86]). Dans l'état actuel de nos recherches, nous ne vérifions pas la consistance statique des relations temporelles mais effectuons uniquement une vérification dynamique à partir de certains scénarios initiaux grâce à la simulation. Nous privilégions donc avant tout l'expressivité.

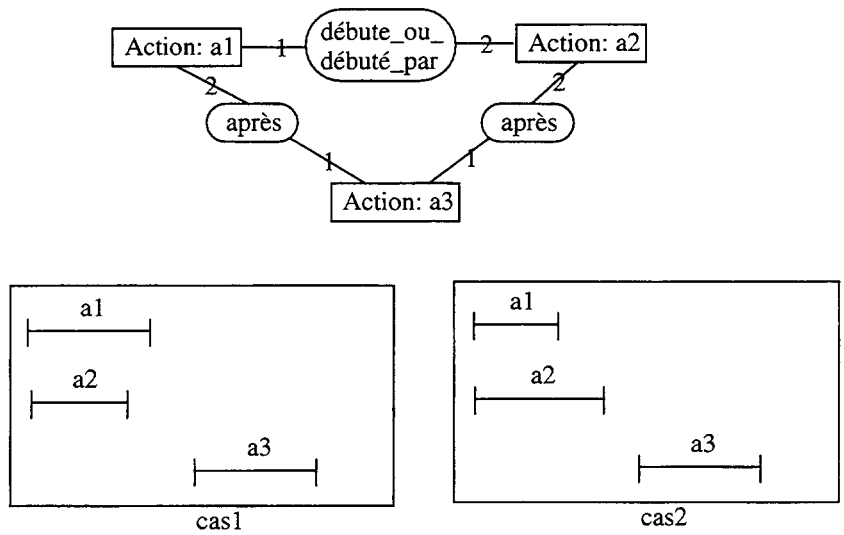


Figure 37. Conjonction de relations temporelles

Si des actions peuvent être synchronisées avec les relations égal, touche, débute et termine, cela signifie qu'elles sont susceptibles de débuter ou terminer exactement aux mêmes instants. Dans le cas de telles simultanéités, on peut alors se demander quelle est la signification d'effets contradictoires. Imaginons par exemple que l'expert dise que deux opérateurs quittent une pièce en même temps et que l'effet d'une action quitter une pièce est le fait d'être dans l'embrasure de la porte de la pièce. Les deux opérateurs se trouvent alors au même instant dans l'embrasure d'une même porte ce qui peut être impossible suivant la taille de la porte. Pour résoudre ce problème, nous supposons que l'égalité entre débuts et fins d'actions est la conséquence d'une granularité trop grossière. En analysant plus finement la position des deux actions, l'expert va en fait argumenter que l'un des deux opérateurs quitte la pièce juste avant l'autre.

Des entités temporelles, situationnelles ou spatiales peuvent être perçues avec plus ou moins de détails, on dit alors qu'elles sont vues à des niveaux de granularités plus ou moins précises. Par exemple, au niveau temporel, l'intervalle [4200 4500] exprimé en secondes sera vu en minutes comme l'intervalle [70 75] et en heures comme l'instant 1. En ce qui concerne la description des actions, l'action téléphoner peut être décrite de façon plus détaillée par les actions décrocher le combiné et composer le numéro. En ce qui concerne le spatial, deux maisons qui semblent collées l'une à l'autre vues de loin sont en fait séparées par une rue lorsqu'on s'en approche.

Les travaux sur une représentation granulaire du temps ont pour objectif de représenter le temps sous plusieurs niveaux de détails, appelés granularités. Dans [EUZ 93], des opérateurs de conversion des représentations entre deux granularités sont proposés. Ces opérateurs peuvent être de type ascendant (passage d'une granularité fine à une granularité plus grossière) ou descendant (passage d'une granularité grossière à une granularité plus fine). En descente, les intervalles restent des intervalles mais l'égalité entre instants peut se révéler contredite, elle est alors remplacée par la disjonction de avant, après et égal (cf figure 38).

Granularité g	Granularité plus précise que g
Avant	Avant
Egal	Avant ou Après ou Egal
Après	Après

Figure 38. Opérateur descendant de conversion sur les relations classiques entre instants

Dans notre représentation de la position des actions par des relations temporelles, nous considérons uniquement deux granularités: un niveau grossier et un niveau précis. Le niveau grossier correspond à la position des actions donnée au moment de l'élicitation. Il retranscrit la première pensée de l'expert. Le niveau précis correspond à une analyse plus fine de la position des actions.

Nous supposons que les débuts et fins d'actions ne peuvent être simultanés que quand leurs positions relatives n'ont pas suffisamment été analysées. Ainsi, le niveau grossier autorise la simultanéité entre débuts et fins d'intervalles tandis que le niveau précis l'exclut.

Dans notre cas, l'égalité au niveau grossier est donc toujours contredite au niveau précis. En revanche, le niveau précis garde quand même une trace de l'égalité au niveau grossier en prenant en compte deux types de durées: une durée imperceptible et une durée significative. Les relations temporelles possibles entre instants au niveau précis sont alors au nombre de quatre: avant, après, juste avant et juste après (cf figure 39). Deux instants égaux au niveau grossier sont séparés par une durée imperceptible au niveau précis. Deux instants différents au niveau grossier sont séparés par une durée significative au niveau précis. La figure 40 donne la conversion descendante des relations temporelles entre instants; les relations avant et après restent les mêmes au niveau précis tandis que la relation égal est remplacée par la disjonction des relations juste avant et juste après.

Relation	Relation inverse	Signification
Avant	Après	. .
Juste avant	Juste après	. .

Figure 39. Relations temporelles entre instants au niveau précis

Niveau grossier	Niveau précis
Avant	Avant

Niveau grossier	Niveau précis
Egal	Juste avant ou Juste après
Après	Après

Figure 40. Conversion descendante des relations entre instants

Les relations temporelles possibles entre intervalles sont issues des relations d'Allen, elles sont au nombre de 22 (cf figure 41). La notation que nous utilisons pour exprimer ces relations indique l'ordre entre les débuts des actions (d1 et d2) et les fins des actions (f1 et f2), un point représente une durée imperceptible tandis qu'un trait représente une durée significative. Chaque relation d'Allen faisant intervenir une égalité au niveau grossier est convertie au niveau précis en une disjonction de relations temporelles (cf figure 42).

Relation	Relation inverse
d1.d2-f1.f2	d2.d1-f2.f1
d2.d1-f1.f2	d1.d2-f2.f1
d1-f1-d2-f2	d2-f2-d1-f1
d1-f1.d2-f2	d2-f2.d1-f1
d1-d2.f1-f2	d2-d1.f2-f1
d1-d2-f1-f2	d2-d1-f2-f1
d1.d2-f1-f2	d2.d1-f2-f1
d2.d1-f1-f2	d1.d2-f2-f1
d1-d2-f2-f1	d2-d1-f1-f2
d1-d2-f1.f2	d2-d1-f2.f1
d1-d2-f2.f1	d2-d1-f1.f2

Figure 41. Relations temporelles entre intervalles au niveau précis

Niveau grossier	Niveau précis
Egal	d1.d2-f1.f2 ou d2.d1-f2.f1 ou d2.d1-f1.f2 ou d1.d2-f2.f1
Avant	d1-f1-d2-f2

Niveau grossier	Niveau précis
Après	d2-f2-d1-f1
Touche	d1-f1.d2-f2 ou d1-d2.f1-f2
Touché-par	d2-f2.d1-f1 ou d2-d1.f2-f1
Recouvre	d1-d2-f1-f2
Recouvert-par	d2-d1-f2-f1
Débute	d1.d2-f1-f2 ou d2.d1-f1-f2
Débuté-par	d2.d1-f2-f1 ou d1.d2-f2-f1
Contenu_dans	d2-d1-f1-f2
Contient	d1-d2-f2-f1
Termine	d2-d1-f2.f1 ou d2-d1-f1.f2
Terminé-par	d1-d2-f1.f2 ou d1-d2-f2.f1

Figure 42. Conversion descendante des relations entre intervalles

Certaines relations du niveau grossier correspondent à des disjonctions de relations au niveau précis; ce sont donc des relations imprécises, leur utilisation introduit de l'indétermination dans les comportements. Le nombre de relations imprécises correspondant à des disjonctions de relations primitives est très grand (2^{22}-23). Il est ainsi impossible de proposer une hiérarchie contenant tous ces types de relations. Nous proposons une hiérarchie de relations temporelles comprenant:

- Les relations primitives du niveau précis
- Les relations du niveau grossier (ou relations d'Allen ([ALL 83]))
- La relation la plus imprécise, appelée relation temporelle et correspondant à la disjonction des 22 relations primitives
- Un certain nombre de relations intermédiaires correspondant à des disjonctions de relations primitives. Nous préférons exprimer ces relations intermédiaires par des termes du langage naturel plutôt que par des disjonctions afin de favoriser leur compréhension. Par exemple, la relation débute_avant (nommée est_né_avant dans [FRE 92]) correspond à la disjonction de 11 relations temporelles primitives (toutes celles où d1 est avant d2 dans la notation des relations temporelles) (cf annexe C pour quelques exemples de relations temporelles imprécises).

En résumé, l'expert peut saisir les synchronisations temporelles entre actions au niveau précis, au niveau grossier ou même de façon encore plus imprécise.

L'algorithme de simulation, lui, génère les résultats de simulation au niveau précis. Ces résultats peuvent cependant être convertis au niveau grossier par une conversion ascendante (cf figure 43). Le niveau grossier constitue alors une simplification du niveau précis. Dans un premier temps, le niveau grossier suffira souvent à l'expert et au cogniticien pour détecter des défauts ou des manques dans les comportements. Une critique plus pointu des comportements pourra ensuite être faite en demandant les résultats de simulation au niveau précis. Nous discuterons de façon plus détaillée de ces aspects relatifs à la simulation dans la troisième partie de la thèse.

Niveau précis	Niveau grossier
Avant	Avant
Juste avant	Egal
Juste après	Egal
Après	Après

Figure 43. Conversion ascendante des relations entre instants

4.3.2. Synchronisation avec des événements

La synchronisation avec des événements permet de synchroniser une action, non pas directement avec une autre action, mais avec des changements qui se produisent dans le monde sans tenir compte de quelles actions ont provoqué ces changements. Par exemple, un opérateur se déplace vers le téléphone de la salle dés que celui-ci se met à sonner, peu importe la personne qui appelle. Le concept de type action est lié au concept de type événement par une relation ***dès_que*** (cf figure 44). Plusieurs relations temporelles dès_que liées à un concept action sont interprétées comme une conjonction de relations dès_que.

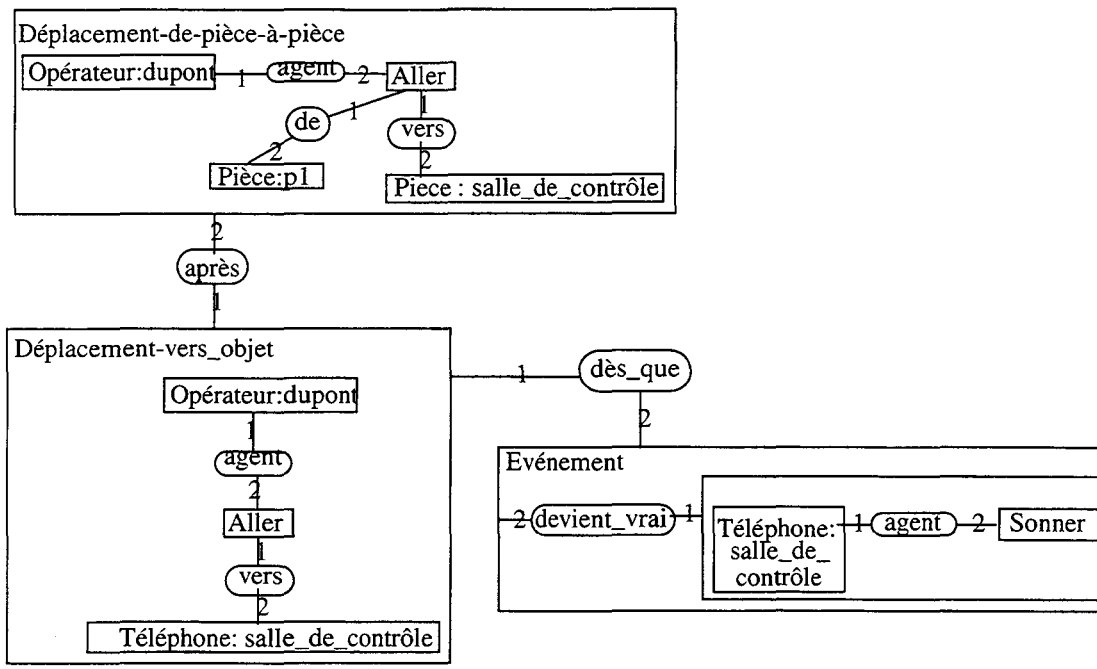


Figure 44. Une action synchronisée par une relation temporelle et par un événement

Examinons maintenant plus précisément la sémantique que nous avons choisie d'attribuer à la relation dès_que liant une action et un événement.

L'événement est une condition nécessaire et suffisante de l'exécution d'une action obligatoire. Comme l'action est obligatoire, il faut que l'événement se produise à un moment donné. Et quand il se produit, l'action est exécutée. Si l'action n'était pas obligatoire, on serait dans le cas d'un conditionnement et non pas uniquement d'une synchronisation (cf partie suivante sur les branchements conditionnels). Si l'événement n'était qu'une condition nécessaire à l'exécution de l'action, on serait plutôt dans le cas d'une précondition, exprimée non pas par un état mais par événement.

L'action est exécutée la première fois que l'événement se produit. Nous considérons cependant que c'est une première fois vis à vis des autres dépendances temporelles et non pas dans l'absolu. Par exemple, l'opérateur se déplace vers le téléphone de la salle de contrôle quand celui-ci sonne mais uniquement après s'être rendu dans la salle de contrôle (cf figure 44). Les relations temporelles sur une action synchronisée par un événement définissent ainsi une plage d'attente de l'événement. Une autre solution serait de décrire explicitement cette plage d'attente par deux événement autres que celui qui synchronise l'action. Mais nous n'avons pas choisi cette option qui alourdit la représentation.

De plus, nous ne considérons pas que l'action débute exactement au moment de l'événement mais au moment suivant. Nous prenons ainsi en compte un délai de réflexion de l'intervenant entre l'événement et le début de l'exécution de l'action.

Nous supposons enfin qu'une action synchronisée par une relation dès_que ne peut pas être liée à une relation temporelle qui positionne l'action juste après le début ou la fin d'une tâche. Cette action est en effet déjà synchronisée par la relation temporelle. Si nous ne faisions pas cette hypothèse, il serait nécessaire de vérifier la cohérence des deux synchronisations.

La sémantique de la relation dès_que correspond à une certain nombre de choix que nous avons effectués. Il serait possible d'enrichir l'ontologie comportementale en proposant d'autres relations correspondant aux autres choix possibles. Par exemple, une relation précondition pourrait lier une action à un événement ou bien une relation au_moment_où pourrait ne refléter aucun délai de réflexion de l'intervenant entre l'événement et l'action. Nous préférons cependant ne pas introduire ces relations car ils ne sont pas pris en compte dans la sémantique formelle et l'algorithme de simulation définis dans la troisième partie de la thèse.

Nous proposons en revanche d'enrichir l'ontologie comportementale avec une relation ***jusque*** qui synchronise la fin d'une action avec une sémantique similaire à la synchronisation du début d'une action avec dès_que. L'événement lié à l'action par une relation jusque est une condition nécessaire et suffisante de la terminaison de l'action. L'action se termine la première fois (vis à vis des autres dépendances temporelles) où l'événement se produit. L'action ne se termine pas exactement au moment de l'événement mais au moment suivant.

4.4. Branchements conditionnels

Par défaut, les actions d'un déroulement sont obligatoires. Cependant, certaines actions ne doivent être exécutées que dans certaines situations. Il est nécessaire de pouvoir exprimer des conditions afin de représenter ces situations. Nous les représentons par des états ou des événements. Les actions conditionnées sont rassemblées dans le déroulement d'un processus particulier: un ***bloc conditionnel***. La condition est liée au bloc conditionnel par une relation ***si***. Le déroulement est lié au bloc conditionnel par une relation ***alors***. Un bloc conditionnel admet au moins pour relations voisines une relation si et une relation alors et il peut également être lié à un déroulement par une relation ***sinon***. Ce déroulement indique alors ce que l'intervenant doit faire quand la condition n'est pas vérifiée. Quand il n'y a pas de relation sinon attachée au bloc conditionnel, l'intervenant n'a rien de spécifique à faire lorsque la condition n'est pas vérifiée, il peut continuer à appliquer le reste de son comportement. En résumé, la condition état ou événement est une condition nécessaire et suffisante à l'exécution de la partie alors du bloc conditionnel tandis que la négation de la condition est une condition nécessaire et suffisante à l'exécution de la partie sinon du bloc conditionnel s'il y en a une.

Les conditions sont testées à un moment où il est possible pour l'intervenant de prendre une décision, c'est-à-dire dans une certaine plage temporelle. Celle-ci est déterminée par des relations temporelles liant le bloc conditionnel à d'autres actions ou à d'autres processus. Par exemple, après avoir identifié le sinistre, l'intervenant peut estimer si le sinistre est grave. S'il est grave, il doit appuyer sur le bouton d'alarme, sinon il doit uniquement prévenir les opérateurs de la salle de contrôle (cf figure 45).

Dans le cas où la condition est un événement, la différence avec une synchronisation temporelle est que l'événement ne doit pas obligatoirement se produire. Si on reprend l'exemple du téléphone qui sonne dans la salle de contrôle, cette fois, l'opérateur n'attend pas le coup de téléphone pour continuer à appliquer son comportement. Il ne va décrocher le téléphone que si celui-ci se met à sonner (cf figure 46).

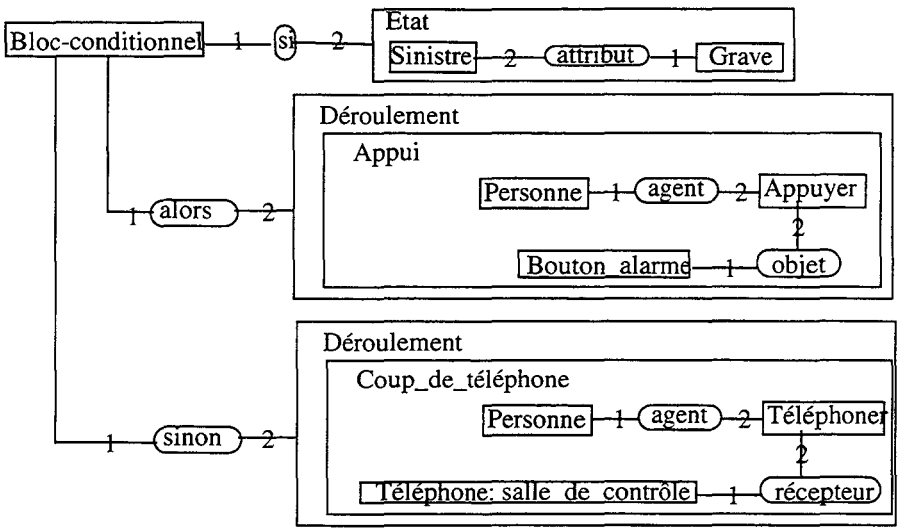


Figure 45. Un bloc conditionnel alors/sinon conditionné par un état

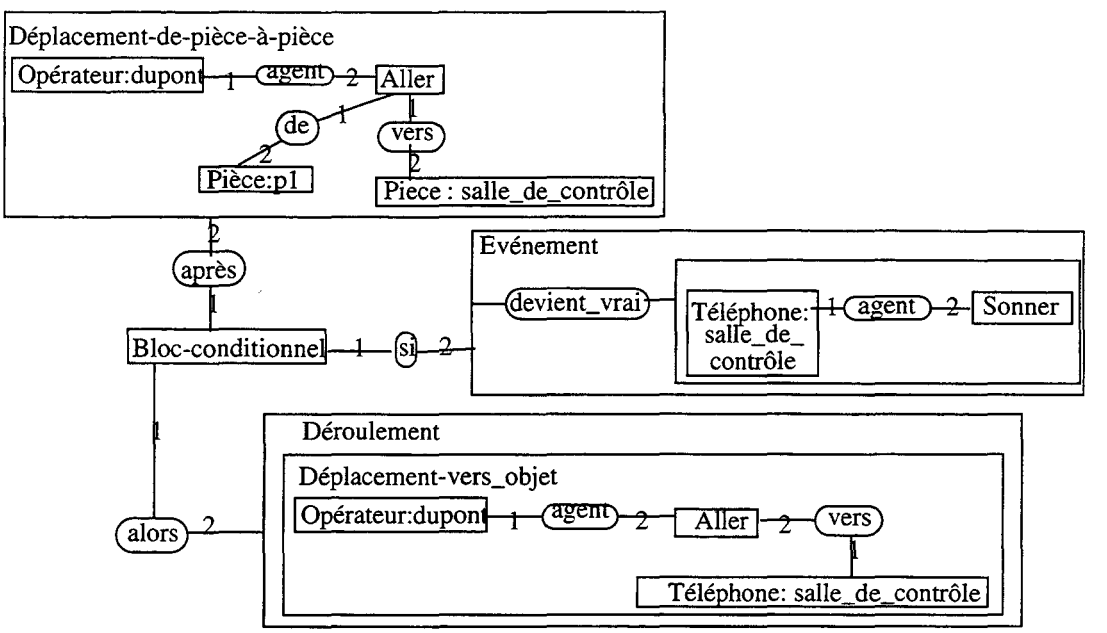


Figure 46. Un bloc conditionnel si conditionné par un événement

4.5. Indéterminations

Le premier type d'indétermination que l'on peut trouver dans les comportements est l'utilisation de relations temporelles imprécises pour synchroniser les actions.

Le deuxième type d'indétermination concerne l'obligation d'effectuer

certaines actions. Des actions peuvent être ou ne pas être exécutées. De telles actions sont optionnelles, elles sont rassemblées dans les déroulements de ***blocs optionnels.*** Un ensemble d'actions à exécuter peut être choisi parmi plusieurs ensembles d'actions à exécuter. Ces ensembles d'actions au choix sont rassemblés dans les déroulements de ***blocs disjonctifs***.

Le déroulement d'un bloc optionnel est lié au concept bloc_optionnel par une relation ***déroulement***. Un bloc optionnel admet une et une seule relation voisine déroulement.

Les deux déroulements d'un bloc disjonctif sont liés au concept bloc_disjonctif par les relations ***déroulement1*** et ***déroulement2***. Un seul de ces déroulements sera choisi et exécuté.

Le troisième type d'indétermination provient de la présence de concepts génériques dans les descriptions d'états, d'événements, de transitions ou d'actions. Par exemple, le directeur ne peut se rendre à l'usine que s'il possède une voiture. S'il possède deux voitures, il peut choisir l'une ou l'autre pour se rendre à l'usine, d'où la présence d'indétermination.

Le quatrième et dernier type d'indétermination provient d'une disjonction de modèles d'action applicables pour une même action d'un déroulement.

Plusieurs raisons peuvent inciter l'expert à exprimer de telles indéterminations. Les deux premières raisons correspondent à un engagement de l'expert, la troisième raison correspond à son ignorance:

- Dans le premier cas, n'importe quel choix doit permettre d'atteindre l'objectif. Par exemple, le témoin peut appeler l'opérateur de la salle de contrôle avant ou après avoir appelé le directeur de l'usine, l'expert pense que le résultat sera similaire.
- Dans le deuxième cas, au moins un des choix doit permettre d'atteindre l'objectif et l'expert pense que l'intervenant est capable de trouver le bon choix. Par exemple, le pompier doit stopper la progression du feu avant ou après avoir soigné les blessés, suivant la gravité du feu, la gravité des blessures et la localisation du feu. C'est le cas de réactions diverses des intervenants suivant les situations (réactions contextuelles). Ces situations (ou contextes) ne sont cependant pas explicitées. Pour ce faire, il aurait par exemple fallu utiliser des blocs conditionnels.
- Le troisième cas est le cas de l'ignorance, c'est le vrai cas d'indétermination. L'expert ne veut pas s'engager sur la signification de l'indétermination, il veut juste voir les résultats de la simulation de l'indétermination. Ces résultats peuvent soit l'aider à résoudre l'indétermination, soit l'aider à s'engager sur sa signification. A la fin du processus de modélisation, il ne doit ainsi plus y avoir d'indéterminations dues à l'ignorance de l'expert dans les comportements.

Quelle que soit la raison de l'indétermination, le principe de simulation d'une indétermination sera le même (ie, plusieurs évolutions différentes du monde seront générées). En revanche, exprimer les raisons d'indéterminations dans les comportements pourrait permettre d'effectuer quelques vérifications automatiques sur ces comportements. Plus précisément, des vérifications pourraient être réalisées dans le cas

des deux premières raisons d'indétermination. On testerait l'engagement de l'expert. Dans le cas de la première raison d'indétermination, il doit y avoir autant d'évolutions du monde valides générées qu'il y a de choix possible. Dans le cas de la deuxième raison, il doit y avoir au moins une évolution du monde valide générée correspondant à un des choix possibles[1].

Nous pourrions ainsi ajouter à l'ontologie des types de concepts et de relations permettant d'exprimer les raisons d'indéterminations dans les comportements. Par exemple, deux sous types du type bloc_disjonctif (bloc_disjonctif_tous_choix et bloc_disjonctif_choix_contextuel) seraient définis pour indiquer l'engagement de l'expert vis à vis de l'indétermination. L'exemple de la figure 47 indiquerait alors que la personne peut soit appeler le premier téléphone de la salle de contrôle, soit le deuxième avec un résultat similaire. Des sous types de bloc_optionnel, état, événement et transition et des sous types des relations temporelles imprécises pourraient également être proposés pour indiquer les raisons d'indétermination.

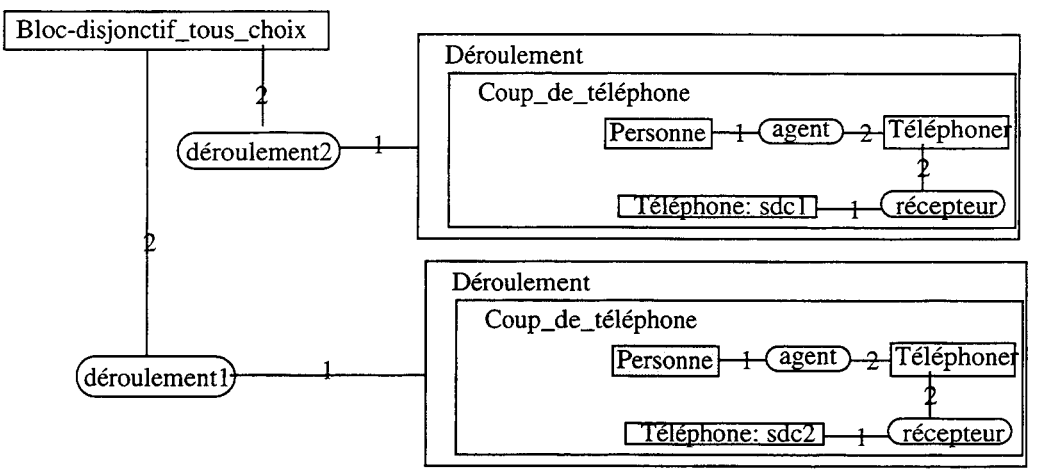


Figure 47. Un bloc disjonctif dans lequel tous les choix doivent permettre d'atteindre l'objectif

4.6. Conclusion

Dans ce chapitre, nous avons présenté les types de l'ontologie comportementale, structurés et définis dans le formalisme des graphes conceptuels. En choisissant et définissant ces types, nous avons posé un certain nombre d'hypothèses (engagements ontologiques). L'ontologie pourrait être enrichie en relaxant ces hypothèses ou en faisant des hypothèses différentes. Notre objectif n'était cependant pas de privilégier l'expressivité, mais de construire une ontologie comportementale interprétable par l'algorithme de simulation. Ainsi, dans la partie 3 de la thèse, tous les types comportementaux sont définis formellement et sont interprétés par l'algorithme général de simulation.

1. L'inconvénient combinatoire d'une telle vérification est qu'elle ne peut être faite que sur l'ensemble de toutes les évolutions du monde générées. Aucune erreur ne peut être détectée en observant une seule évolution du monde.

Partie II

Simulation: raisonnement sur les descriptions du monde

Cette partie est consacrée aux raisonnements sur les descriptions du monde nécessaires pour simuler l'exécution des actions. Comme les descriptions du monde sont représentées par des graphes conceptuels, ces raisonnements sont des raisonnements sur les graphes. Ils sont de deux types: la démonstration consiste, à partir d'une description courante du monde, à déterminer la valeur de vérité d'une propriété du monde et la mise à jour consiste, à partir d'une description courante du monde et d'une information nouvelle signalant un changement du monde, à prédire la description résultante du monde.

Une évolution du monde (ie, un résultat de simulation) est décrite par un ensemble totalement ordonné d'instants clés entre lesquels aucun changement ne se produit. A chacun de ces instants clés, est associée une description instantanée du monde, appelée point-situation. Un état conditionne l'exécution d'une action ou d'un processus (bloc-conditionnel ou comportement). Un événement synchronise une action ou conditionne et synchronise un processus (bloc-conditionnel ou comportement). Une transition donne un effet d'une action sur le monde. Afin de vérifier l'exécutabilité des actions, il faut démontrer des états et des événements à partir de point-situations. Et afin de simuler l'exécution des actions, il faut mettre à jour des point-situations avec des transitions.

Le premier chapitre de cette partie est consacré à la représentation des point-situations, états, événements et transitions, tous décrits par des propriétés du monde exprimées avec des graphes conceptuels. Le formalisme de base des graphes conceptuels est étendu par la prise en compte de négation. Le deuxième chapitre est relatif à la démonstration. Nous décrivons tout d'abord les opérations de démonstration dans le modèle de base des graphes conceptuels. Nous détaillons ensuite un principe de démonstration des états (respectivement des événements) basé sur un choix de représentation des point-situations et des états (respectivement des événements). Le troisième chapitre est relatif à la mise à jour. Nous décrivons quelques approches de mise à jour reposant sur différentes représentations du monde et des effets des actions. Nous proposons alors un principe de mise à jour, le monde, les changements du monde et les lois du domaine étant exprimés dans le formalisme des graphes conceptuels.

Chapitre 5

Représentation des point-situations, états, événements et transitions

Le point-commun entre les point-situations, états, événements et transitions est leur description par des propriétés du monde exprimées avec des graphes conceptuels. Dans un premier temps, nous nous attachons à caractériser les graphes propriétés du monde. Dans ces graphes, la négation peut être décrite implicitement par une hypothèse du monde clos ou explicitement par des emboîtements négatifs. Nous discutons alors de la représentation des point-situations, des états, des événements et des transitions.

5.1. Les propriétés du monde

Les propriétés du monde sont représentées par des graphes conceptuels. Les types des sommets concepts et relations de ces graphes appartiennent à l'ontologie du domaine, que nous allons décrire brièvement. Nous caractériserons ensuite les référents et les emboîtements positifs autorisés dans les sommets concepts de ces graphes. Finalement, nous discuterons de la représentation de la négation dans ces graphes, qui peut être implicite ou explicite.

5.1.1. Types de concepts et de relations

Les concepts utilisés dans les graphes propriétés du monde sont des concepts du domaine, ils représentent:

- des intervenants, des instruments, des collections d'intervenants et/ou d'instruments
- leur localisation
- leurs attributs (par exemple, le fonctionnement, l'accessibilité, la couleur des instruments ou l'âge et la rapidité des intervenants)
- les états de croyance, vouloir, savoir, possession des intervenants

Au cours de ce travail de thèse, nous n'avons pas figé l'ontologie du domaine. Figer l'ontologie comportementale ainsi que sa sémantique était nécessaire pour pouvoir définir l'algorithme de simulation. Il n'est en revanche pas nécessaire de figer une ontologie du domaine car la simulation est complètement indépendante de celle-ci.

Les exemples de la thèse et les premières expérimentations du langage et de

la simulation reposent sur l'ontologie générique de haut niveau de l'annexe D. Des travaux de recherche restent à faire, concernant cette ontologie. Nous la décrivons cependant brièvement afin de favoriser la bonne compréhension des exemples de la thèse. Dans l'ontologie du domaine, nous distinguons les types de concepts concrets qui existent physiquement dans le monde des types de concepts abstraits qui correspondent à des réifications de relations. Les concepts concrets sont:

- des entités (personnes ou instruments)
- des collections (ensemble de personnes et/ou d'instruments)
- des concepts spatiaux (les quatre concepts spatiaux endroit, chemin, surface, volume correspondent aux trois dimensions de l'espace comme dans [JAC 90]).

Les concepts abstraits sont:

- des verbes de situation (verbes d'actions tels que se déplacer ou verbes d'états tels que posséder)
- des attributs
- des propositions exprimant la croyance, la volonté ou la connaissance d'une personne.

Les relations exprimant la nature et les propriétés des liens entre ces concepts sont les suivantes:

- Les relations casuelles lient un verbe de situation et une entité, elles sont issues des cas sémantiques de C.J. Fillmore [FIL 68] et caractérisent ainsi les relations de sens qui existent entre les noms et le verbe dans une phrase simple.
- Les relations spatiales lient une entité ou un verbe d'action à un concept spatial, elles positionnent ainsi une entité ou l'exécution d'une action dans l'espace.
- Les relations attribut lient une entité à un attribut.
- Les relations sur les collections permettent enfin de décrire l'appartenance d'une entité à une collection ou l'inclusion d'une collection dans une autre collection.

Des métaconcepts caractérisent les types de l'ontologie du domaine (cf annexe D). Les graphes propriétés doivent vérifier les contraintes d'utilisation imposées par les métaconcepts signatures et acceptions. Ils doivent également être cohérents vis-à-vis des métaconcepts définition, qui constituent un ensemble de lois du domaine toujours vraies[1].

5.1.2. Référents

Un concept concret est générique ou individuel (dans ce dernier cas, son référent appartient à l'ensemble I des référents individuels). Comme les concepts abstraits

1. Comme il n'y a aucune forme de négation dans le modèle de base des graphes conceptuels, les graphes sont obligatoirement cohérents vis-à-vis des métaconcepts définition. La prise en compte de négation nécessiterait de vérifier la cohérence des graphes propriétés du monde.

expriment des relations entre des concepts concrets, donner des noms à leurs référents est fastidieux. Ces concepts seront ainsi généralement génériques. Comme nous le verrons par la suite, dans les points-situations, ils peuvent éventuellement être référencés automatiquement par des numéros différents.

Figure 48. Indéterminations dans une propriété du monde

Un référent générique d'un concept concret ou abstrait est interprété logiquement par une variable quantifiée existentiellement. Il exprime une indétermination. Dans l'exemple de la figure 48, une personne est dans une usine, mais on ne sait pas quelle est cette personne, ni dans quelle usine elle se trouve:

$$\exists x \exists y(\text{personne}(x) \wedge \text{usine}(y) \wedge \text{dans}(x, y)) \qquad (20)$$

5.1.3. Emboîtements positifs

Dans les propriétés du monde, des graphes peuvent être emboîtés dans des concepts d'un unique type: le type proposition. La proposition est l'objet d'une croyance, d'une volonté ou d'une connaissance d'un intervenant. L'emboîtement utilisé est un emboîtement simple sans typage. Par exemple, la propriété de la figure 49 exprime qu'un opérateur sait qu'il y a un sinistre dans l'usine de Chocques.

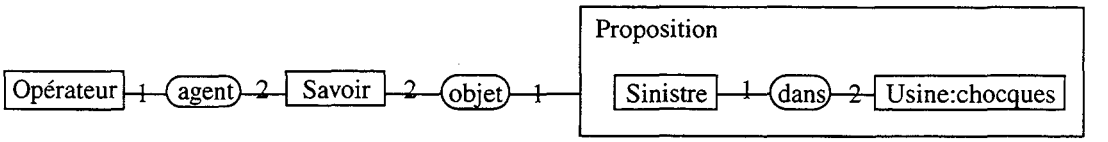


Figure 49. Un emboîtement dans un concept de type proposition

5.1.4. Négation

Dans le modèle de base des graphes conceptuels, le raisonnement est limité à la démonstration de la vérité de propriétés positives (ie, équivalentes à des formules conjonctives positives). Cependant, au cours de la simulation, nous avons besoin de démontrer la vérité ou la fausseté de graphes ou de morceaux de graphes pour prouver l'exécutabilité ou la non exécutabilité des actions. Il nous est ainsi indispensable d'étendre le modèle de base des graphes conceptuels en prenant en compte une forme de négation.

Dans les bases de données, afin de limiter le volume d'information à conserver, la négation est définie de façon implicite. Seules les informations positives sont représentées. En revanche, dans les systèmes de raisonnement logique, contenant souvent peu de volume d'information, la négation est représentée explicitement par un connecteur noté $\neg$.

Dans un premier temps, nous allons décrire une représentation implicite de la

négation dans le modèle des graphes conceptuels par une hypothèse du monde clos. Nous discuterons ensuite d'une représentation explicite de la négation par des contextes négatifs.

Monde clos

Il est possible de ne décrire que des propriétés positives du monde en utilisant le formalisme de base du modèle des graphes conceptuels et en considérant que la négation est définie de façon implicite par une hypothèse du monde clos. L'écriture des négations est alors économisée.

En logique des prédicats du premier ordre, le monde clos d'une théorie a été défini de la façon suivante par R. Reiter dans [REI 77]:

- Tout atome de base qu'on ne peut pas démontrer à partir d'une théorie logique T est supposé faux dans le monde clos de la théorie T (noté CWA(T)). Ainsi, CWA(T)=T+{ ¬p avec p atome de base et p ne peut pas être démontré à partir de T }

R. Reiter a montré que l'utilisation unique de clauses de Horn[1] dans une théorie logique garantit que le monde clos de la théorie soit consistant. Les figure 50 et figure 51 montrent des exemples de théories logiques dont les mondes clos sont inconsistants. Dans le cas de la figure 50, la théorie T est constituée d'une clause qui n'est pas une clause de Horn. Dans le cas de la figure 51, la théorie T contient une formule avec une variable quantifiée existentiellement; son monde clos est inconsistant si on fait une hypothèse du domaine clos (ie, si les seules constantes sont celles de la théorie).

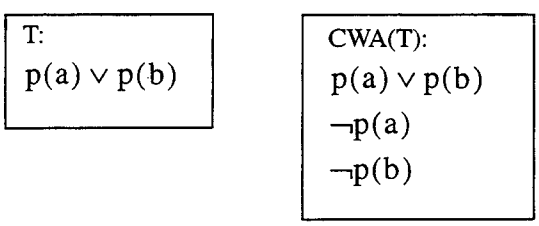


Figure 50. Un monde clos inconsistant en cas de clauses quelconques

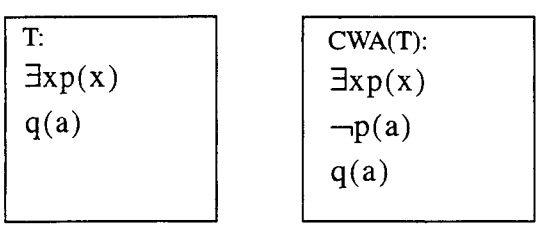


Figure 51. Un monde clos inconsistant en cas de domaine clos et quantificateurs existentiels

1. Une clause est une disjonction de littéraux dont les arguments sont soit des constantes, soit des variables quantifiées universellement. Une clause de Horn comporte au plus un littéral positif.

Utiliser le monde clos d'une théorie exprimée avec des graphes conceptuels pour économiser l'écriture des négations nécessite de prendre quelques précautions.En effet, si nous faisons une hypothèse de domaine clos, les concepts génériques étant interprétés comme des variables quantifiées existentiellement, il est possible que le monde clos de la théorie exprimée avec des graphes conceptuels soit inconsistant. Un moyen de prévenir une telle inconsistance est d'interdire les concepts génériques, tous les concepts doivent alors avoir des référents individuels. Dans notre cas, nous devons ainsi nommer les concepts concrets et numéroter automatiquement les concepts abstraits, qu'il serait laborieux de nommer.

Utiliser le monde clos pour exprimer la négation de façon implicite nécessite en fait de connaître la valeur de vérité de toutes les propriétés du monde (ie, la connaissance sur le monde doit être complète).

Contextes négatifs

Dans [SOW 84], J. Sowa introduit la notion de contexte négatif afin de pouvoir nier l'existence de graphes conceptuels[1]. La notation utilisée est celle de l'exemple de la figure 52, qui signifie qu'aucune autruche ne vole.

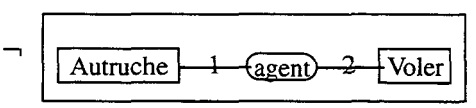


Figure 52. Contexte négatif de J. Sowa

Si c est un contexte négatif contenant un ensemble de graphes G1,...,Gn alors:

$$\phi(c) = \neg(\phi(G_1) \wedge \ldots \wedge \phi(G_n)) \quad (21)$$

L'interprétation logique du graphe de la figure 52 est ainsi la formule 22 équivalente à la formule 23.

$$\neg(\exists x \exists y(\text{autruche}(x) \wedge \text{voler}(y) \wedge \text{agent}(x, y))) \quad (22)$$

$$\forall x \forall y(\neg\text{autruche}(x) \vee \neg\text{voler}(y) \vee \neg\text{agent}(x, y)) \quad (23)$$

Il est à noter qu'un graphe contenu dans un contexte négatif peut lui même contenir un contexte négatif. On obtient ainsi plusieurs niveaux de négation et on peut exprimer n'importe quelle formule de la logique des prédicats du premier ordre sans symbole fonctionnel dont la sémantique intuitive n'est pas forcément une négation mais peut aussi être une implication ou une disjonction.

J. Sowa introduit alors la notion de ligne d'identité pour indiquer que deux concepts appartenant à deux contextes négatifs différents représentent exactement le

1. Il s'est pour cela inspiré du cut introduit par C.S. Peirce dans ses graphes existentiels ([SOW 93]).

même individu[1]. Une ligne d'identité est un graphe connexe non orienté g dont les sommets sont des concepts et les arcs des paires de concepts, appelés liens de coréférence[2]:

- Aucun concept ne peut appartenir à plus d'une ligne d'identité.
- Un concept a dans g domine un autre concept b s'il existe un chemin <a1,a2,...,an> de g où a=a1 et b=an, et pour tout i, soit ai et ai+1 apparaissent dans le même contexte, ou le contexte de ai domine celui de ai+1.
- Deux concepts a et b sont coréférents si soit a domine b, soit b domine a.
- Un concept a est dominant si a domine chaque concept qui domine a.

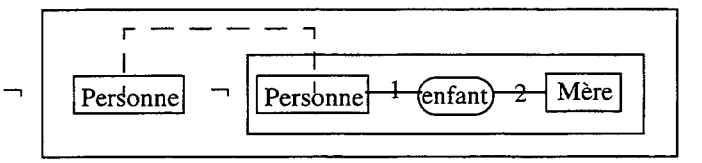


Figure 53. Un lien de coréférence interprété par l'assignation d'une variable unique

Dans certains cas, un lien de coréférence pourrait être interprété logiquement en assignant une variable unique aux concepts coréférents. Le graphe de la figure 53 s'interpréterait alors par la formule 24 équivalente à la formule 25:

$$\neg\exists x(\text{personne}(x) \wedge \neg\exists y(\text{mere}(y) \wedge \text{enfant}(x, y))) \quad (24)$$

$$\forall x(\text{personne}(x) \rightarrow \exists y(\text{mere}(y) \wedge \text{enfant}(x, y))) \quad (25)$$

Ce n'est cependant pas possible par exemple quand:

- un concept négatif est coréférent avec deux concepts positifs (figure 54)
- le concept coréférent négatif n'est pas générique (figure 55)

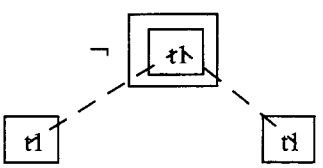


Figure 54. Un concept négatif coréférent avec deux concepts positifs

1. Il s'inspire des lignes d'identité des graphes existentiels de C.S. Peirce ([SOW 93]).
2. On retrouve la notion de lien de coréférence que nous avons déjà introduite dans la première partie de la thèse. Les liens de coréférence permettaient alors d'indiquer que deux concepts appartenant à deux emboîtements positifs différents se référaient au même individu.

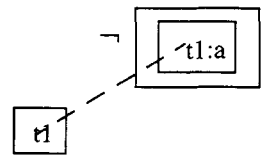


Figure 55. Concept coréférent négatif non générique

J. Sowa propose l'algorithme suivant pour simplifier un graphe u en supprimant ses lignes d'identité:

Algorithme 1: Supprimer-lignes-identité(u)
Début
Assigner un unique nom de variable à chaque concept générique de u
Pour tout a dans l'ensemble des concepts dominants de u **boucler**
x ← identificateur(a)
concaténer "=x" au champ référent de tous les concepts dominés par a
FPour
Effacer tous les liens de coréférence de u
Fin

$\phi(u)$ peut alors être calculée sachant que si b est un concept de u de la forme [t:x1=x2=...=xn], alors $\phi(b)$ est de la forme $t(x1) \wedge (x1 = x2) \wedge \ldots \wedge (x1 = xn)$. Les interprétations logiques des graphes des figure 53, figure 54 et figure 55 sont alors respectivement les formules 26, 27 et 28.

$$\neg\exists x(\text{personne}(x) \wedge \neg\exists y\exists z(\text{personne}(y) \wedge (y = x) \wedge \text{mere}(z) \wedge \text{enfant}(y, z))) \quad (26)$$

$$\exists x\exists y(t1(x) \wedge t1(y) \wedge \neg\exists z(t1(z) \wedge (x = z) \wedge (y = z))) \quad (27)$$

$$\exists x(t1(x) \wedge \neg(t1(a) \wedge (x = a))) \quad (28)$$

Les lignes d'identité introduisent des termes avec égalité dans les formules associées aux graphes conceptuels et l'égalité est une notion avec laquelle il sera difficile de raisonner. Dans [LAP 97], S. Lapalut a d'ailleurs montré un problème dû aux lignes d'identité dans l'adaptation par J. Sowa des règles d'inférences du système β de C.S. Peirce aux graphes conceptuels.

En outre, bien que les lignes d'identité et plusieurs niveaux d'imbrication de contextes négatifs permettent d'obtenir au moins toute l'expressivité de la logique des prédicats du premier ordre (et même plus en prenant en compte l'égalité), les représentations obtenues sont peu intuitives et difficiles d'accès pour un expert. Par exemple, il est loin d'être immédiat et naturel d'interpréter le graphe de la figure 53 comme une implication (ou une règle) qui signifie que toute personne a une mère. Les liens de coréférence des contextes négatifs viennent en plus s'ajouter aux liens de coréférence des emboîtements positifs ce qui alourdit énormément la représentation.

Plutôt que d'avoir une grande expressivité, notre objectif est d'obtenir un bon

compromis entre:

- l'expressivité
- la lisibilité
- le raisonnement sur les graphes[1]

Pour décrire les propriétés du monde, nous utilisons une autre représentation explicite de négation, moins expressive mais plus lisible. Cette représentation n'autorise qu'un seul niveau de négation et n'utilise pas de lignes d'identité, elle permet de nier l'existence d'une conjonction de sommets concepts et relations. Le contexte entoure la conjonction niée de sommets et contrairement aux contextes négatifs de J. Sowa, il peut être traversé par un arc liant une relation à un concept[2]. Nous représentons un tel contexte par un ovale (cf figure 56).

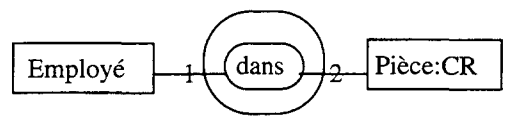


Figure 56. Négation représentée avec un contexte

Nous imposons les contraintes d'utilisation suivantes pour les contextes ovales:

- Un contexte contenant un concept ayant des relations voisines doit aussi contenir ces relations voisines.
- Deux contextes ne peuvent pas s'intersecter (Notamment, un contexte ne peut pas contenir un autre contexte).

La négation exprimée avec des contextes ovales respectant les contraintes ci-dessus peut être traduite en une négation exprimée avec les contextes de J. Sowa (cf figure 57). On n'obtient alors que des graphes avec un seul niveau de négation et dont les liens de coréférence peuvent être interprétés en assignant une variable unique aux concepts coréférents.

1. Nous rappelons que nous avons volontairement omis les aspects de raisonnements dans ce chapitre, nous les aborderons dans les deux chapitres suivants. Dans ce chapitre, nous abordons la négation uniquement du point de vue représentation et interprétation logique.
2. Un tel contexte rejoint la notion d'aire introduite par S. Lapalut dans [LAP 97]. Ce dernier autorise cependant plusieurs niveaux d'imbrication d'aires.

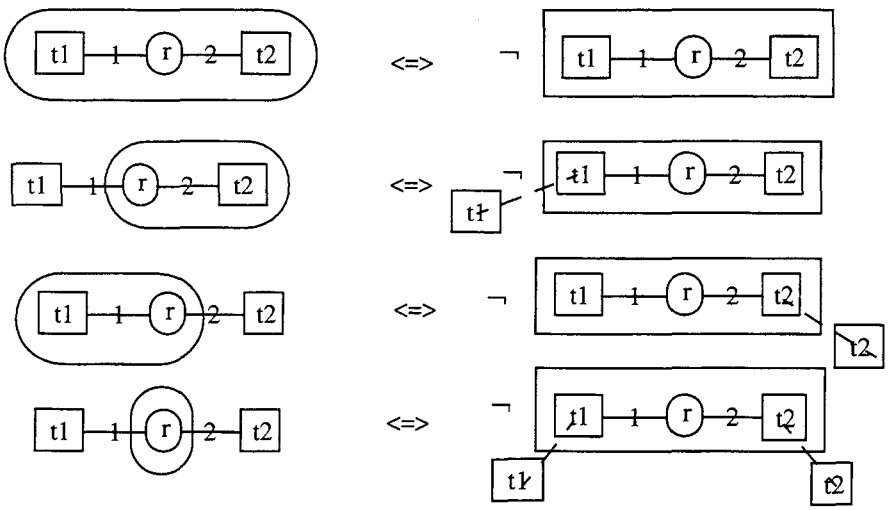


Figure 57. Equivalence avec la notation de J. Sowa

Les interprétations logiques des graphes contenant des contextes négatifs ovales sont de la forme:

$$Cp \wedge Rp \wedge \neg(Cn_1 \wedge Rn_1) \wedge \ldots \wedge \neg(Cn_i \wedge Rn_i) \qquad (29)$$

- Cp est une conjonction de prédicats unaires correspondant aux concepts à l'extérieur des contextes, dont les arguments sont soit des constantes, soit des variables quantifiées existentiellement
- Rp est une conjonction de prédicats binaires correspondant aux relations à l'extérieur des contextes, dont les arguments sont soit des constantes, soit des variables quantifiées existentiellement
- $\forall i$, Cn_i est une conjonction de prédicats unaires correspondant aux concepts à l'intérieur du contexte i, dont les arguments sont soit des constantes, soit des variables quantifiées universellement
- $\forall i$, Rn_i est une conjonction de prédicats binaires correspondant aux relations à l'intérieur du contexte i, dont les arguments sont soit des constantes, soit des variables quantifiées existentiellement, soit des variables quantifiées universellement

L'opérateur ϕ procède de la façon suivante pour donner l'interprétation logique d'un graphe avec des contextes négatifs ovales:

- Il associe à chaque concept générique une variable distincte xi
- Il associe à chaque marqueur individuel m la constante m
- Il représente chaque concept c par un prédicat unaire de nom type(c) et d'argument la variable ou la constante associée à c
- Il représente chaque relation r par un prédicat n-aire de nom type(r) et de ième

argument, la variable ou constante associée au sommet concept lié à r par l'arête étiquetée i

- Pour chaque contexte, il prend la conjonction des prédicats des sommets à l'intérieur de ce contexte
- Il nie chacune des conjonctions obtenues
- Il prend alors la conjonction de ces conjonctions niées et des prédicats des sommets qui n'étaient dans aucun contexte
- Il quantifie existentiellement les variables des concepts génériques qui n'étaient dans aucun contexte (les quantificateurs existentiels sont placés au début de la formule logique).
- Il quantifie universellement les variables des concepts génériques à l'intérieur des contextes (les quantificateurs universels sont placés avant chaque négation).

L'interprétation logique du graphe de la figure 56 est ainsi la suivante:

$$\exists x(employe(x) \wedge piece(cr) \wedge \neg dans(x, cr)) \quad (30)$$

Après avoir discuté de la représentation des propriétés du monde, nous allons maintenant présenter comment ces propriétés vont permettre de décrire des point-situations, des états, des événements et des transitions.

5.2. Point-situations

Un point-situation n'est pas un concept de l'ontologie comportementale (Les concepts comportementaux sont ceux introduits dans le chapitre 3). Un point-situation n'appartient ainsi pas au langage de modélisation. C'est une description instantanée du monde, il donne la valeur de vérité de toutes les propriétés du monde à un instant clé de l'évolution du monde. Les point-situations sont décrits par des graphes propriétés du monde, ils présentent les résultats de simulation à l'expert et au cogniticien (cf partie III pour une description détaillée de la simulation). Un point situation n'est pas forcément complet dans la sens où la valeur de vérité de certaines des propriétés peut être indéterminée. En revanche, nous faisons une hypothèse de domaine clos dans chaque point-situation, c'est-à-dire que les seuls référents individuels existants sont ceux du point-situation.

Deux représentations de la négation sont envisageables dans les point-situations: une représentation implicite par une hypothèse du monde clos et une représentation explicite par des contextes négatifs[1]. Dans le cas de la représentation implicite, le point-situation doit être complet; tous les concepts doivent avoir des référents individuels. Le seul moyen de prendre en compte des indéterminations est alors de traiter des point-situations en parallèle dans plusieurs évolutions du monde. Dans le cas d'une représentation explicite, le point-situation peut contenir des indéterminations; les référents des concepts peuvent être génériques. Il faut par exemple quatre point-situations

1. Les algorithmes de démonstration et de mise à jour présentés dans la suite de cette partie reposent sur une représentation implicite de la négation dans les point-situations.

exprimés avec le monde clos (cf figure 58) (et donc déjà quatre évolutions différentes du monde) pour exprimer un scénario initial équivalent à celui de la figure 59 exprimé avec une représentation explicite de la négation. Il n'y a que quatre point-situations car une même personne ne peut pas être dans deux pièces différentes, car une personne se trouve forcément dans une pièce, car une personne ne peut pas être une pièce et car une pièce ne peut pas être une personne (Ces connaissances sont nécessaires pour compléter le point-situation exprimé avec des contextes ovales.).

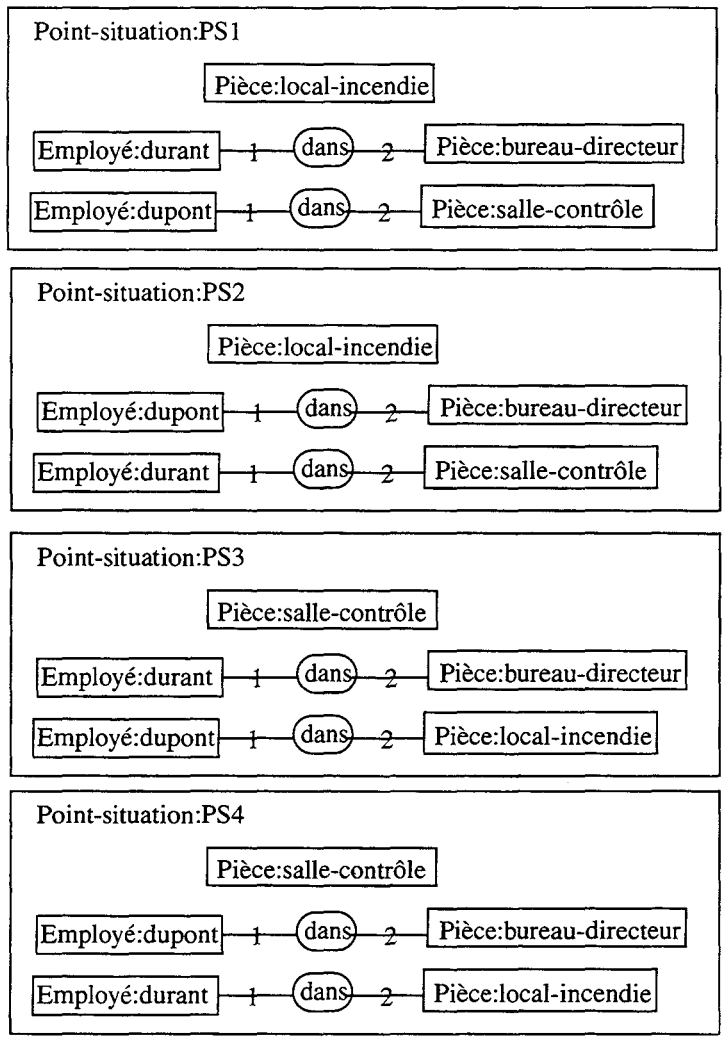


Figure 58. Des point-situations avec une représentation implicite de la négation

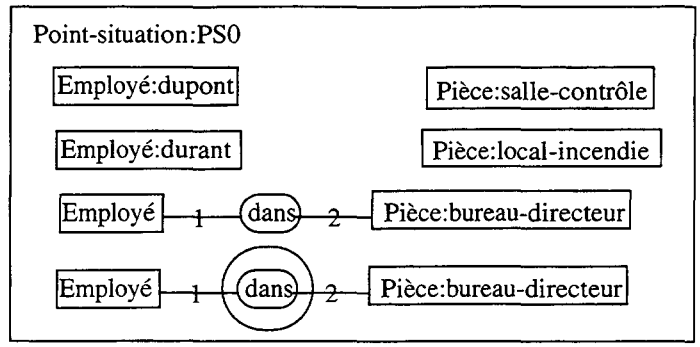


Figure 59. Un point-situation avec une représentation explicite de la négation

Afin de donner l'interprétation logique des point-situations, états, événements et transitions, nous ajoutons un second argument temporel aux prédicats correspondant aux concepts et un troisième argument temporel aux prédicats correspondant aux relations binaires. Cet argument temporel peut être un instant ou un intervalle, représenté par un couple d'instants[1].

L'interprétation logique d'un point-situation décrit par un graphe propriété du monde G et associé à un instant clé i est $\phi(G, i)$.

L'interprétation logique du point-situation de la figure 59 associé à l'instant i est ainsi la suivante:

$$\text{employe(dupont, i)} \wedge \text{employe(durant, i)} \wedge \text{piece(salledecontrole, i)} \wedge \text{piece(localincendie, i)}$$

$$\wedge \text{piece(bureaudirecteur, i)} \wedge \exists x(\text{employe}(x, i) \wedge \text{dans}(x, \text{bureaudirecteur}, i))$$

$$\wedge \exists y(\text{employe}(y, i) \wedge \neg\text{dans}(y, \text{bureaudirecteur}, i)) \quad (31)$$

L'hypothèse de domaine clos nous donne une connaissance supplémentaire sur la description du monde à l'instant i:

$$(\forall x(\text{employe}(x, i) \rightarrow (x = \text{dupont}) \vee (x = \text{durant}) \vee (x = \text{salledecontrole}) \vee$$

$$(x = \text{localincendie}) \vee (x = \text{bureaudirecteur}))) \quad (32)$$

$$\forall x(\text{piece}(x, i) \rightarrow (x = \text{dupont}) \vee (x = \text{durant}) \vee (x = \text{salledecontrole}) \vee$$

$$(x = \text{localincendie}) \vee (x = \text{bureaudirecteur})) \quad (33)$$

L'interprétation logique du point-situation PS1 de la figure 58 associé à l'instant i est la suivante:

$$\text{employe(dupont, i)} \wedge \text{employe(durant, i)} \wedge \text{piece(salledecontrole, i)} \wedge$$

$$\text{piece(localincendie, i)} \wedge \text{piece(bureaudirecteur, i)} \wedge \text{dans(durant, bureaudirecteur, i)} \wedge$$

1. Nous définissons de façon plus détaillée la logique temporelle utilisée dans le chapitre 8.

$$\text{dans(dupont, sallecontrole, i)} \quad (34)$$

L'hypothèse du domaine clos nous donne à nouveau la connaissance supplémentaire des formules 32 et 33. L'hypothèse du monde clos nous donne ensuite la connaissance suivante:

$$\neg\text{piece(dupont, i)} \wedge \neg\text{piece(durant, i)} \wedge$$
$$\neg\text{employe(sallecontrole, i)} \wedge \neg\text{employe(localincendie, i)} \wedge \neg\text{employe(bureaudirecteur, i)} \wedge$$
$$\neg\text{dans(dupont, dupont, i)} \wedge \neg\text{dans(durant, durant, i)} \wedge \neg\text{dans(sallecontrole, sallecontrole, i)} \wedge$$
$$\neg\text{dans(bureaudirecteur, bureaudirecteur, i)} \wedge \neg\text{dans(localincendie, localincendie, i)} \wedge$$
$$\neg\text{dans(dupont, durant, i)} \wedge \neg\text{dans(dupont, localincendie, i)} \wedge \neg\text{dans(dupont, bureaudirecteur, i)} \wedge$$
$$\neg\text{dans(durant, dupont, i)} \wedge \neg\text{dans(durant, localincendie, i)} \wedge \neg\text{dans(durant, sallecontrole, i)} \wedge$$
$$\neg\text{dans(localincendie, dupont, i)} \wedge \neg\text{(localincendie, durant, i)} \wedge$$
$$\neg\text{dans(localincendie, bureaudirecteur, i)} \wedge \neg\text{dans(localincendie, sallecontrole, i)} \wedge$$
$$\neg\text{dans(bureaudirecteur, dupont, i)} \wedge \neg\text{dans(bureaudirecteur, durant, i)} \wedge$$
$$\neg\text{dans(bureaudirecteur, localincendie, i)} \wedge \neg\text{dans(bureaudirecteur, sallecontrole, i)} \wedge$$
$$\neg\text{dans(sallecontrole, dupont, i)} \wedge \neg\text{dans(sallecontrole, durant, i)} \wedge$$
$$\neg\text{dans(sallecontrole, bureaudirecteur, i)} \wedge \neg\text{dans(sallecontrole, localincendie, i)} \quad (35)$$

5.3. Etats

Un état est un concept du langage de modélisation, qui conditionne l'exécution d'une action ou d'un processus (bloc-conditionnel ou comportement). Un état est une caractérisation du monde pendant une période de temps (la durée de l'état[1]). Il est décrit par un ensemble de propriétés invariantes durant toute cette période. Ces propriétés sont représentées par des graphes propriétés du monde emboîtés à l'intérieur du concept état. Il est impossible d'utiliser le monde clos pour représenter la négation dans les états car on ne décrit pas complètement le monde, mais uniquement quelques-unes de ses propriétés. Pour la même raison, on ne fait pas d'hypothèse de domaine clos. La négation sera ainsi explicitement représentée dans les états avec des contextes négatifs. La figure 60 donne un exemple d'état: "Un employé n'est pas dans le bureau du directeur".

1. La durée d'un état peut être nulle.

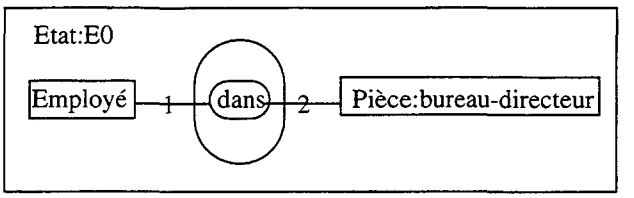


Figure 60. Un exemple d'état

L'interprétation logique d'un état E décrit par un graphe G sur un intervalle (i,j) est $\phi(G, (i, j))$.

L'interprétation logique sur (i,j) de l'état de la figure 60 est ainsi la suivante:

$$\text{piece}(\text{bureaudirecteur}, (i, j)) \wedge \exists x(\text{employe}(x, (i, j)) \wedge \neg\text{dans}(x, \text{bureaudirecteur}, (i, j))) \quad (36)$$

5.4. Evénements

Un événement est un concept du langage de modélisation qui synchronise une action ou conditionne et synchronise un processus (bloc-conditionnel ou comportement). C'est un changement ponctuel du monde qui sera testé par l'algorithme de simulation. Les événements sont décrits par des graphes emboîtés à l'intérieur des concepts événements avec deux types d'emboîtements: devient-vrai ou devient faux. Ces graphes sont des graphes propriétés du monde ne contenant aucune forme de négation. La figure 61 donne un exemple d'événement dont l'interprétation en langage naturel est "une sirène se met à sonner alors qu'aucune sirène ne sonnait juste avant".

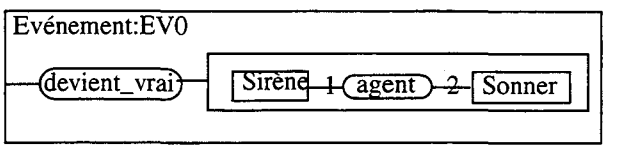


Figure 61. L'événement "une sirène se met à sonner"

- Soit un événement EV qui se produit en i, décrit par un graphe propriété G sans négation avec la sémantique d'emboîtement devient-vrai, son interprétation logique est $\neg\phi(G, i-1) \wedge \phi(G, i)$
- Soit un événement EV qui se produit en i, décrit par un graphe propriété G sans négation avec la sémantique d'emboîtement devient-faux, son interprétation logique est $\phi(G, i-1) \wedge \neg\phi(G, i)$

L'interprétation logique de l'événement de la figure 61 qui se produit en i est ainsi la suivante:

$$\forall x1 \forall y1 \neg(\text{sirene}(x1, i-1) \wedge \text{agent}(x1, y1, i-1) \wedge \text{sonner}(y1, i-1)) \wedge$$

$$\exists x2 \exists y2(\text{sirene}(x2, i) \wedge \text{agent}(x2, y2, i) \wedge \text{sonner}(y2, i)) \quad (37)$$

5.5. Transitions

Une transition est un concept du langage de modélisation qui donne un effet d'une action sur le monde. C'est un changement ponctuel du monde qui est appliqué au point-situation courant par l'algorithme de simulation afin de construire un nouveau point-situation. Les transitions sont décrites de la même façon que les événements (ie, par des graphes propriétés du monde sans négation emboîtés à l'intérieur des concepts transitions avec deux types d'emboîtements: devient-vrai ou devient-faux). La figure 62 donne un exemple de transition dont la représentation est similaire à celle de l'événement de la figure 61.

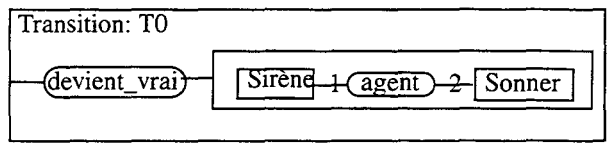


Figure 62. La transition "la sirène se met à sonner"

L'interprétation logique des transitions est cependant différente de celle des événements:

- Soit une transition T appliquée en i et décrite par un graphe propriété G sans négation avec la sémantique d'emboîtement devient-vrai, son interprétation logique est $\neg\phi(G, i-1) \rightarrow \phi(G, i)$
- Soit une transition T appliquée en i et décrite par un graphe propriété G sans négation avec la sémantique d'emboîtement devient-faux, son interprétation logique est $\phi(G, i-1) \rightarrow \neg\phi(G, i)$

L'interprétation logique de la transition de la figure 62, appliquée en i est ainsi la suivante:

$$\forall x1 \forall y1 \neg(\text{sirene}(x1, i-1) \wedge \text{agent}(x1, y1, i-1) \wedge \text{sonner}(y1, i-1)) \rightarrow$$

$$\exists x2 \exists y2(\text{sirene}(x2, i) \wedge \text{agent}(x2, y2, i) \wedge \text{sonner}(y2, i)) \quad (38)$$

La transition de la figure 62 est imprécise[1]. La raison pour laquelle la propriété devient vraie est indéterminée; on ne sait pas quels sont les sommets du graphe qui n'existaient pas et qui existent maintenant (sommets créés). De la même façon, si la sémantique d'emboîtement était devient faux, on ne connaîtrait pas les sommets qui existaient et qui n'existent plus ensuite (sommets supprimés).

Pour chaque sémantique d'emboîtement, nous proposons de distinguer les

1. L'événement de la figure 61 est également imprécis.

sommets créés ou supprimés en les mettant en surbrillance[1] (figure 63 et figure 64). Nous mettons ainsi précisément en évidence la différence entre les deux graphes décrivant le point-situation courant et le nouveau point-situation.

Les relations voisines d'un concept surbrillant sont également surbrillantes:

- Dans le cas de la suppression d'un concept, les relations voisines du concept sont obligatoirement supprimées.
- Dans le cas de la création d'un concept, il faut attacher ce concept aux relations mentionnées dans la transition afin que la propriété emboîtée dans la transition devienne vraie.

Il est à noter que les hypothèses du monde clos et du domaine clos dans les point-situations posent certaines contraintes sur les graphes décrivant les transitions.

- Dans le cas d'une transition devient-vrai, les concepts génériques surbrillants ne peuvent pas être des concepts concrets. Si tel était le cas, le point-situation créé par l'application de la transition contiendrait des concepts génériques. (Nous avons vu dans le paragraphe 5.1.4 que ça n'était pas possible). Les transitions (2) et (4) de la figure 63 ne sont ainsi pas autorisées.
- Les concepts abstraits surbrillants d'une transition devient-vrai peuvent éventuellement être surbrillants. Ils seront numérotés automatiquement dans le point-situation créé par l'application de la transition. Leurs types doivent cependant être placés juste au dessus du type absurde dans la hiérarchie des types de concepts. Si tel n'était pas le cas, il y aurait création d'une indétermination dans le nouveau point-situation.

La formule 39 donne l'interprétation logique d'une transition précise dont la sémantique d'emboîtement est devient vrai (cette formule est une spécialisation de $\neg\phi(G, i-1) \rightarrow \phi(G, i)$) et la formule 40 donne l'interprétation logique d'une transition précise dont la sémantique d'emboîtement est devient faux (cette formule est une spécialisation de $\phi(G, i-1) \rightarrow \neg\phi(G, i)$):

- Pns((x1,...,xj),i) est la conjonction des prédicats correspondant aux sommets de la partie non surbrillante de la transition dont les arguments atemporels sont des constantes ou les variables x1,...,xj et dont l'unique argument temporel est i (la partie non surbrillante de la transition contient j concepts génériques).
- Ps((x1,...,xk),(y1,...,yj),i) est la conjonction des prédicats correspondant aux sommets de la partie surbrillante de la transition dont les arguments atemporels sont des constantes ou les variables x1,...,xk ou les variables y1,...,yj et dont l'unique argument temporel est i (la partie surbrillante de la transition contient k concepts génériques et des relations de la partie surbrillante peuvent éventuellement être liées

1. Nous pourrions également distinguer les sommets créés ou supprimés dans les événements. Cette notation est cependant moins nécessaire dans le cas des événements. Les événements sont uniquement testés pendant la simulation tandis que les transitions permettent de construire de nouveaux point-situations. L'imprécision dans le cas des transitions amène ainsi la construction de nombreux point-situations dont certains peuvent ne pas être réalistes.

à certains des j concepts génériques de la partie non surbrillante).

- notPs((x1,...,xk),(y1,...,yj),i) est la conjonction des prédicats correspondant aux sommets de la partie surbrillante de la transition, mais chacun de ces prédicats est cette fois nié.

$$\forall a1 \ldots \forall aj \forall b1 \ldots \forall bk \neg(Pns((a1, \ldots, aj), i-1) \wedge Ps((b1, \ldots, bk), (a1, \ldots, aj), i-1)) \rightarrow$$

$$(\exists c1 \ldots \exists cj(Pns((c1, \ldots, cj), i-1)) \rightarrow \exists d1 \ldots \exists dj \exists e1 \ldots \exists ek(Pns((d1, \ldots, dj), i-1) \wedge$$

$$Pns((d1, \ldots, dj), i) \wedge notPs((e1, \ldots, ek), (d1, \ldots, dj), i-1) \wedge Ps((e1, \ldots, ek), (d1, \ldots, dj), i))) \quad (39)$$

$$\exists a1 \ldots \exists aj \exists b1 \ldots \exists bk(Pns((a1, \ldots, aj), i-1) \wedge Ps((b1, \ldots, bk), (a1, \ldots, aj), i-1)) \rightarrow$$

$$\forall c1 \ldots \forall cj \forall d1 \ldots \forall dk(Pns((c1, \ldots, cj), i-1) \wedge Ps((d1, \ldots, dk), (c1, \ldots, cj), i-1) \rightarrow$$

$$Pns((c1, \ldots, cj), i) \wedge notPs((d1, \ldots, dk), (c1, \ldots, cj), i)) \quad (40)$$

La transition de la figure 62 est ainsi équivalente à la disjonction de quatre transitions précises (figure 63) et chacune des formules 41 à 44 est une spécialisation de la formule 38:

- S'il existe un concept sirène et un concept sonner en i-1, on crée une relation agent entre ces deux concepts (1).

$$\forall a1 \forall a2 \neg(sirene(a1, i-1) \wedge sonner(a2, i-1) \wedge agent(a1, a2, i-1)) \rightarrow$$

$$(\exists c1 \exists c2(sirene(c1, i-1) \wedge sonner(c2, i-1)) \rightarrow$$

$$\exists d1 \exists d2(sirene(d1, i-1) \wedge sonner(d2, i-1) \wedge \neg agent(d1, d2, i-1) \wedge$$

$$sirene(d1, i) \wedge sonner(d2, i) \wedge agent(d1, d2, i))) \quad (41)$$

- S'il existe un concept sirène en i-1, on crée un nouveau concept sonner et on le lie par une relation agent au concept sirène (2).

$$\forall a1 \forall b1 \neg(sirene(a1, i-1) \wedge sonner(b1, i-1) \wedge agent(a1, b1, i-1)) \rightarrow$$

$$(\exists c1(sirene(c1, i-1)) \rightarrow$$

$$\exists d1 \exists e1(sirene(d1, i-1) \wedge \neg sonner(e1, i-1) \wedge \neg agent(d1, e1, i-1) \wedge$$

$$sirene(d1, i) \wedge sonner(e1, i) \wedge agent(d1, e1, i))) \quad (42)$$

- S'il existe un concept sonner en i-1, on crée un nouveau concept sirène et on le lie par une relation agent au concept sonner (3).

$$\forall a1 \forall b1 \neg(sonner(a1, i-1) \wedge sirene(b1, i-1) \wedge agent(b1, a1, i-1)) \rightarrow$$

$$(\exists c1(sonner(c1, i-1)) \rightarrow$$

$$\exists d1 \exists e1(sonner(d1, i-1) \wedge \neg sirene(e1, i-1) \wedge \neg agent(e1, d1, i-1) \wedge$$

$$\neg\text{sonner}(d1, i) \wedge \text{sirene}(e1, i) \wedge \text{agent}(e1, d1, i))) \quad (43)$$

- On crée un concept sirène et un concept sonner et on les lie par une relation agent (4).

$$\forall b1 \forall b2 \neg(\text{sonner}(b1, i-1) \wedge \text{sirene}(b2, i-1) \wedge \text{agent}(b2, b1, i-1)) \rightarrow$$

$$\exists e1 \exists e2(\neg\text{sonner}(e2, i-1) \wedge \neg\text{sirene}(e1, i-1) \wedge \neg\text{agent}(e1, e2, i-1) \wedge$$

$$\neg\text{sonner}(e2, i) \wedge \text{sirene}(e1, i) \wedge \text{agent}(e1, e2, i)) \quad (44)$$

Si la sémantique d'emboîtement de la transition de la figure 62 était devient-faux, la transition serait également équivalente à la disjonction de quatre transitions précises (figure 64):

- On supprime toutes les relations agent entre les concepts sirène et sonner (5).

$$\exists a1 \exists a2(\text{sirene}(a1, i-1) \wedge \text{sonner}(a2, i-1) \wedge \text{agent}(a1, a2, i-1)) \rightarrow$$

$$\forall c1 \forall c2(\text{sirene}(c1, i-1) \wedge \text{sonner}(c2, i-1) \wedge \text{agent}(c1, c2, i-1) \rightarrow$$

$$\text{sirene}(c1, i) \wedge \text{sonner}(c2, i) \wedge \neg\text{agent}(c1, c2, i)) \quad (45)$$

- On supprime tous les concepts sonner liés à un concept sirène par une relation agent (6).

$$\exists a1 \exists b1(\text{sirene}(a1, i-1) \wedge \text{sonner}(b1, i-1) \wedge \text{agent}(a1, b1, i-1)) \rightarrow$$

$$\forall c1 \forall d1(\text{sirene}(c1, i-1) \wedge \text{sonner}(d1, i-1) \wedge \text{agent}(c1, d1, i-1) \rightarrow$$

$$\text{sirene}(c1, i) \wedge \neg\text{sonner}(d1, i) \wedge \neg\text{agent}(c1, d1, i)) \quad (46)$$

- On supprime tous les concepts sirène liés à un concept sonner par une relation agent (7).

$$\exists a1 \exists b1(\text{sonner}(a1, i-1) \wedge \text{sirene}(b1, i-1) \wedge \text{agent}(b1, a1, i-1)) \rightarrow$$

$$\forall c1 \forall d1(\text{sonner}(c1, i-1) \wedge \text{sirene}(d1, i-1) \wedge \text{agent}(d1, c1, i-1) \rightarrow$$

$$\text{sonner}(c1, i) \wedge \neg\text{sirene}(d1, i) \wedge \neg\text{agent}(d1, c1, i)) \quad (47)$$

- On supprime tous les concepts sonner et sirène liés par une relation agent (8).

$$\exists b1 \exists b2(\text{sonner}(b1, i-1) \wedge \text{sirene}(b2, i-1) \wedge \text{agent}(b2, b1, i-1)) \rightarrow$$

$$\forall d1 \forall d2(\text{sonner}(d1, i-1) \wedge \text{sirene}(d2, i-1) \wedge \text{agent}(d2, d1, i-1) \rightarrow$$

$$\neg\text{sonner}(d1, i) \wedge \neg\text{sirene}(d2, i) \wedge \neg\text{agent}(d2, d1, i)) \quad (48)$$

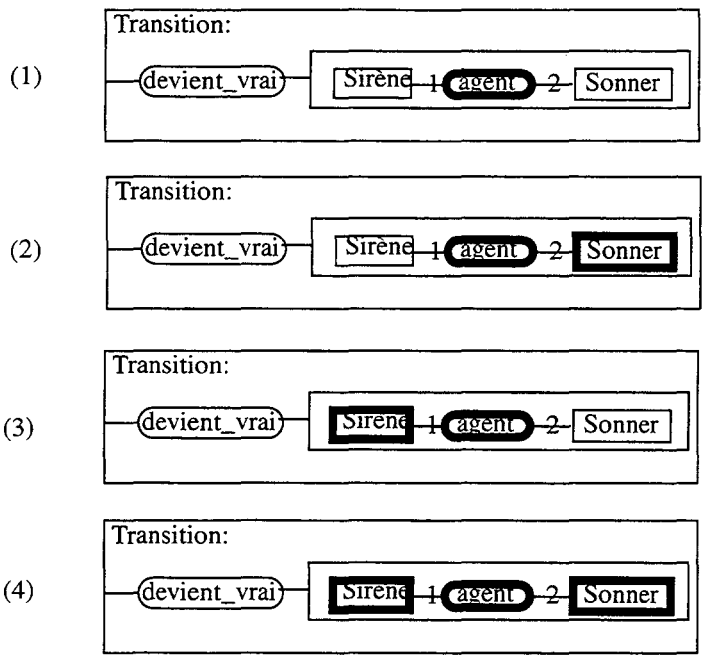


Figure 63. Quatre transitions précises "une sirène se met à sonner"

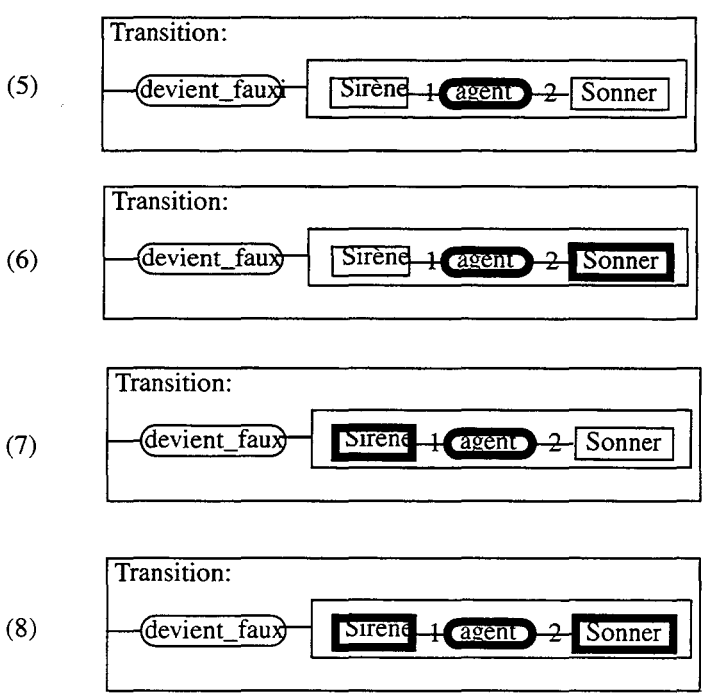


Figure 64. Quatre transitions précises "plus aucune sirène ne sonne"

5.6. Conclusion

Dans ce chapitre, nous avons proposé une représentation des point-situations, états, événements et transitions à partir de graphes propriétés du monde. Cette représentation pourrait être améliorée par des travaux de recherche supplémentaires sur l'ontologie du domaine et sur le formalisme des graphes conceptuels. Nous concluons en donnant quelques pistes pour ces travaux de recherche.

5.6.1. Enrichissement de l'ontologie du domaine

Pour enrichir l'ontologie du domaine, il serait intéressant d'exploiter des travaux qui ont été réalisés pour la définition d'ontologies génériques. Par exemple, dans l'ontologie de son outil d'acquisition des connaissances GGKAT ([MAR 95], [MAR 96]), P. Martin incorpore les 49000 types de concepts du dictionnaire sémantique Wordnet ([MIL 93]). Ce dictionnaire est une base de données lexicale inspirée par les théories psycholinguistiques de la mémoire humaine. Dans Wordnet, les noms, verbes et adjectifs anglais sont organisés en ensemble de synonymes, chacun de ces ensembles représentant un concept. Différentes relations lient alors ces concepts, dont les relations sorte-de ou partie-de. Un autre exemple d'ontologie générique contenant un important volume de connaissances de natures diverses est l'ontologie Cyc [LEN 95]. Cette ontologie est en construction depuis une douzaine d'années et sert de base à des applications de divers types (langage naturel, recherche d'information, partage de connaissances...). Elle satisfait deux critères principaux qui sont l'universalité (ie, n'importe quel concept imaginé par l'homme doit pouvoir être classé dans l'ontologie) et l'articulation (ie, distinguer correctement les concepts de l'ontologie est une condition nécessaire et suffisante à la bonne réalisation des applications).

5.6.2. Prise en compte de formes plus riches de négation

En ce qui concerne le formalisme de représentation, le pouvoir d'expression pourrait par exemple être augmenté dans les graphes propriétés du monde:

- en prenant en compte d'autres formes de négation telles qu'une négation sur les référents (par exemple, un employé différent de dupont, figure 65, formule 49) ou une négation sur les types (par exemple, un concept qui n'est pas un employé, figure 66, formule 50).

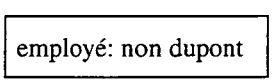


Figure 65. Négation sur les référents

$$\exists x(employe(x) \wedge x \neq dupont) \qquad (49)$$

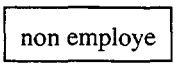


Figure 66. Négation sur des types de concept

$$\exists x \neg employe(x) \tag{50}$$

- en définissant une relation d'implication ou de disjonction, les figure 67 et figure 68 illustrent la façon dont J. Sowa définit de telles relations dans [SOW 84].

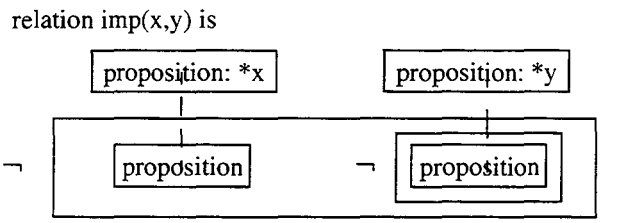


Figure 67. Relation imp

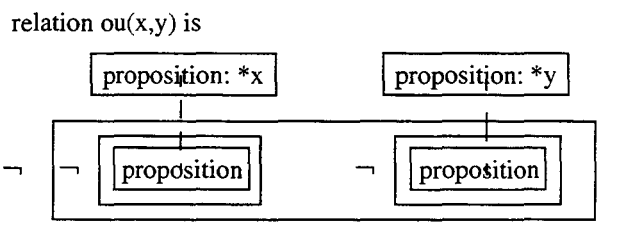


Figure 68. Relation ou

- en prenant en compte un référent universel, le graphe de la figure 69 serait alors équivalent à celui de la figure 53.

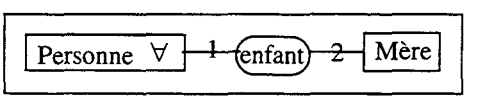


Figure 69. Référent universel

Il faut toutefois essayer de ne pas porter préjudice à la lisibilité et il reste en plus à considérer les aspects de raisonnement avec ces différentes extensions du formalisme des graphes conceptuels, ce que nous n'avons pas fait durant cette thèse. Nous avons uniquement abordé les aspects de raisonnement avec une représentation de la négation par une hypothèse du monde clos ou par des contextes négatifs tels que ceux de la figure 56 (cf chapitres 6 et 7).

5.6.3. Changements du monde plus expressifs

Finalement, en ce qui concerne la représentation des événements et des transitions, l'expressivité pourrait être augmentée en exprimant un changement du monde par deux propriétés: une propriété positive et une propriété négative, des liens de coréférence pouvant lier des concepts de chacune de ces propriétés [1]. Il est alors possible de représenter une transition imprécise équivalente à celle de la figure 62 (cf figure 70), ou une transition précise équivalente à la transition (1) de la figure 63 (cf figure 71), mais aussi de représenter d'autres transitions signifiant par exemple "une sirène se met à sonner mais la propriété "une sirène sonne" ne change pas forcément de valeur de vérité" (cf figure 72). Les inconvénients d'une telle représentation sont une moins grande lisibilité et surtout la difficulté d'écriture. Il faut par exemple s'assurer que la propriété positive est bien en contradiction avec la propriété négative sinon il n'y a pas réellement de changement du monde. L'expressivité est en revanche supérieure, elle sera d'ailleurs d'autant plus grande que nous prendrons en compte des formes plus complexes de négation.

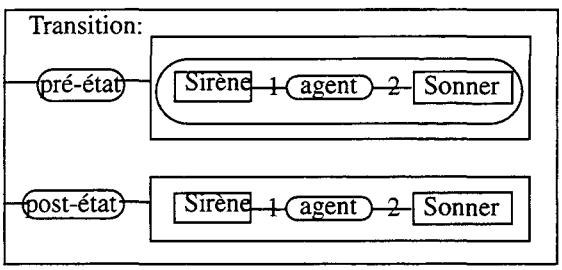


Figure 70. Une transition imprécise "une sirène se met à sonner alors qu'aucune sirène ne sonnait" représentée avec deux propriétés

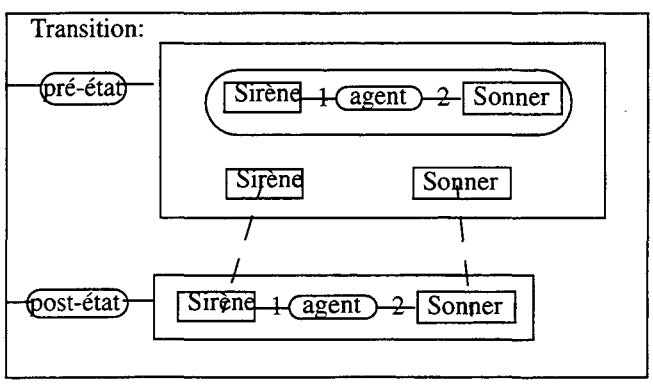


Figure 71. une transition précise "une sirène se met à sonner alors qu'aucune sirène ne sonnait" représentée avec deux propriétés

1. C'est d'ailleurs la façon dont nous avions choisi de représenter les transitions dans ([BOS 97a], [BOS 97b], [BOS 97c], [BOS 97d]). Nous y avons renoncé depuis à cause de la lisibilité et des problèmes posés par le raisonnement.

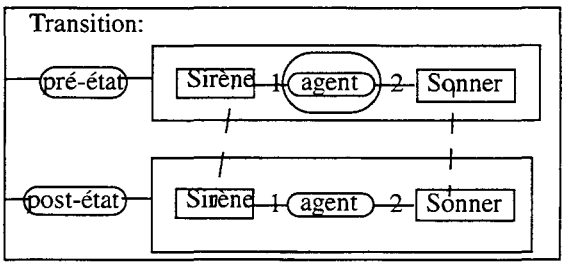


Figure 72. Une transition "une sirène se met à sonner"

Chapitre 6

Démonstration

Le premier type de raisonnement mis en jeu dans la simulation des comportements est la démonstration. Afin de vérifier l'exécutabilité des actions, il faut déterminer la valeur de vérité d'états et d'événements à partir de point-situations.

Dans ce chapitre, nous définissons d'abord l'opération de projection qui permet de démontrer un graphe conceptuel à partir d'un autre graphe conceptuel. Nous examinons alors des travaux de recherche qui proposent des algorithmes de réponses à des requêtes sur une base de connaissance, mais qui ne prennent en compte aucune forme de négation. Dans ces travaux, la base de connaissances contient des définitions de types ou des règles, qui peuvent être appliquées en chaînage avant ou arrière. Dans le cas du chaînage avant, l'algorithme de démonstration est directement basé sur la projection tandis que le chaînage arrière nécessite des opérations sur les graphes plus complexes. Finalement, nous présentons les algorithmes de démonstration d'états et d'événements actuellement utilisés par l'algorithme de simulation. Ces algorithmes utilisent l'opération de projection et considèrent certaines formes de négation.

6.1. Démontrer avec des graphes conceptuels

Nous définissons et caractérisons la projection, ainsi que son extension dans le cas des graphes emboîtés. Nous présentons alors des travaux bibliographiques qui proposent des algorithmes de réponse à des requêtes sur une base de connaissances.

6.1.1. La projection

Définition

Une ***projection*** d'un graphe conceptuel $H=(R_H, C_H, U_H, lab_H)$ dans un graphe conceptuel $G=(R_G, C_G, U_G, lab_G)$ est un couple d'applications $P=(f,g)$, $f: R_H \rightarrow R_G$, $g: C_H \rightarrow C_G$, qui

- conserve les arêtes et la numérotation des arêtes
  $\forall (r, c) \in U_H, (f(r), g(c)) \in U_G$ et $\forall c = H_i(r), g(c) = G_i(f(r))$
- peut restreindre les étiquettes des sommets
  $\forall r \in R_H$ avec $lab_G(f(r))=tr'$ et $lab_H(r)=tr$ alors $tr' \leq_r tr$
  $\forall c \in C_H$ avec $lab_G(g(c))=(tc',m')$ et $lab_H(c)=(tc,m)$ alors $tc' \leq_r tc$ et $m' \leq m$

Dans l'exemple de la figure 73, il existe une seule projection de H dans G.

L'application f est schématisée par des flèches en traits pleins et l'application g par des flèches en pointillé. Le résultat de la projection est alors le graphe P(H) avec P=(f,g).

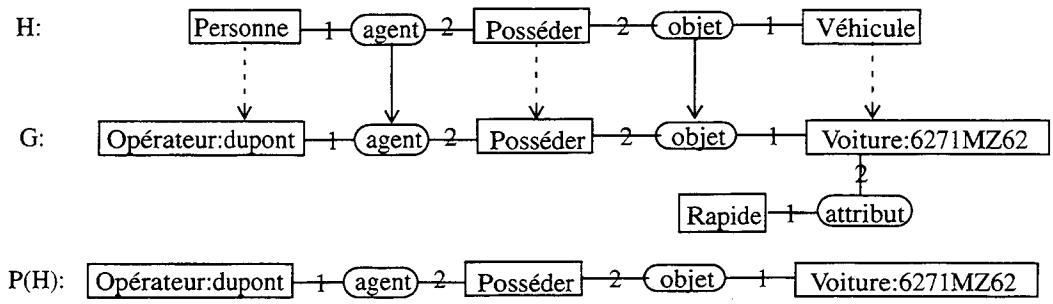


Figure 73. Le résultat d'une projection de H dans G

Propriétés

La projection correspond à un ***calcul de spécialisation*** entre deux graphes. Soient G et H deux graphes conceptuels, il existe une projection de H dans G si et seulement si $G \leq H$, c'est à dire que G peut être dérivé de H et éventuellement d'autres graphes par l'application d'un nombre fini de règles de spécialisation. On dit aussi que G est une spécialisation de H ou bien que H est une généralisation de G.

La projection correspond à la ***démonstration*** sur l'ensemble des formules logiques associées aux graphes conceptuels. Soient G et H deux graphes conceptuels sous forme normale définis sur un support S, $G \leq H$ si et seulement si $\phi(G), \phi(S) \vdash \phi(H)$ (=> consistance [SOW 84], <= complétude [CHE 92],[MUG 95]). Si G n'est pas sous forme normale, le résultat de complétude n'est pas valable. Un graphe conceptuel est sous forme normale s'il ne contient pas deux sommets ayant même marqueur individuel. La forme normale d'un graphe est ainsi obtenu en fusionnant les sommets concepts ayant même marqueur individuel.

Algorithmes

Un algorithme de projection par backtrack, permettant de fournir toutes les projections d'un graphe conceptuel quelconque dans un autre, a été implémenté dans la plate-forme CoGITo de gestion de graphes du LIRMM ([HAE 95]). Cet algorithme est présenté de façon détaillée dans [LEC 95]. Il tire parti du fait que le choix d'une image par f d'un sommet relation détermine les images par g des concepts voisins de cette relation. Il s'attache donc d'abord à déterminer l'application f (qui détermine également l'application g pour les concepts voisins des relations). Puis en cas de composantes connexes réduites à des uniques sommets concepts, l'algorithme détermine l'application g pour ces sommets concepts. Nous donnons ci-dessous un algorithme de projection par backtrack d'un graphe Q dans un graphe R, les différents résultats de projection (f,g) sont obtenus en backtrackant sur les points de choix.

Algorithme 2: Projeter (Q,R)

Début

$Q=(R_Q, C_Q, U_Q, lab_Q)$
$G=(R_G, C_G, U_G, lab_G)$
Pour tout $c \in C_Q$
$g(c) \leftarrow \varnothing$
Fpour
Pour tout $r \in R_Q$
$f(r) \leftarrow \varnothing$
Fpour
Pour tout $r \in R_Q$ avec c1 et c2 respectivement 1er et 2nd voisins de r
$lab_Q(r)=tr$
$lab_Q(c1)=(tc1,m1)$
$lab_Q(c2)=(tc2,m2)$
CHOISIR $r' \in R_G$ avec c1' et c2' respectivement 1er et 2nd voisins de r'
$lab_G(r')=tr'$
$lab_G(c1')=(tc1',m1')$
$lab_G(c2')=(tc2',m2')$
Si $tr' \leq_r tr$ et $tc1' \leq_c tc1$ et $tc2' \leq_c tc2$ et $m1' \leq m1$ et $m2' \leq m2$
Si $g(c1) = \varnothing$
$g(c1) \leftarrow c1'$
Fsi
Si $gc2 = \varnothing$
$g(c2) \leftarrow c2'$
Fsi
Si $g(c1) \neq c1'$ ou $g(c2) \neq c2'$
échec
Sinon
$f(r) \leftarrow r'$
Fsi
Sinon
échec
Fsi
Fpour
Pour tout $c \in C_Q$ sans relation voisine
$lab_Q(c)=(tc,m)$
CHOISIR $c' \in C_G$
$lab_G(c')=(tc',m')$
Si $tc' \leq_c tc$ et $m' \leq m$
Si $g(c) = \varnothing$
$g(c) \leftarrow c'$
Fsi
Si $g(c) \neq c'$
échec
Fsi
Fsi
Fpour
Si il existe un c tel que $g(c) = \varnothing$ ou il existe un r tel que $f(r) = \varnothing$
échec
Fsi
Retourner (f,g)

Fin

M. Chein et M.L. Mugnier ont démontré dans [CHE 92], que le problème de l'existence d'une projection d'un graphe conceptuel quelconque dans un autre est NP-

complet. Ce problème correspond en effet à un morphisme de graphes orientés. Ils proposent également un algorithme polynômial pour déterminer l'existence d'un graphe conceptuel sans cycles (avec une structure d'arbre) dans un graphe conceptuel quelconque. Cet algorithme est implémenté dans la plate-forme CoGITo ([HAE 95]).

Projection sur des graphes emboîtés

Dans le cas des graphes ***emboîtés***, la définition de la projection est la même que dans les cas des graphes simples, excepté pour la restriction des étiquettes des sommets:

- $\forall r \in R_H$ avec $lab_G(f(r))=tr'$ et $lab_H(r)=tr$ alors $tr' \leq_r tr$
  $\forall c \in C_H$ avec $lab_G(g(c))=(tc',m',d')$ et $lab_H(c)=(tc,m,d)$ alors $tc' \leq_r tc$ et $m' \leq m$ et si d est un ensemble de graphes alors d' est un ensemble de graphes et chaque graphe de d se projette sur un graphe de d', si d est la description générique (notée **) alors d' est quelconque.

Dans le cas des graphes ***emboîtés typés***, la restriction des étiquettes des sommets est la suivante:

- $\forall r \in R_H$ avec $lab_G(f(r))=tr'$ et $lab_H(r)=tr$ alors $tr' \leq_r tr$
  $\forall c \in C_H$ avec $lab_G(g(c))=(tc',m',d')$ et $lab_H(c)=(tc,m,d)$ alors $tc' \leq_r tc$ et $m' \leq m$ et si d est un ensemble d'emboîtements typés alors d' est un ensemble d'emboîtements typé et pour chaque emboîtement typé (ei,Gi) de d, il existe un emboîtement typé (e'j,G'j) de d' tel que Gi se projette dans G'j et $ej' \leq_e ei$, si d est la description générique (notée **) alors d' est quelconque.

6.1.2. Requêtes sur une base de connaissances

Dans cette partie, une base de connaissances est un ensemble de graphes conceptuels définis sur un support, plus éventuellement des définitions de types ou des règles de graphes. Poser une requête (sous la forme d'un graphe question) sur une base de connaissances consiste à rechercher si le graphe question peut être démontré à partir de la base de connaissance. Si c'est le cas, la réponse à la requête est "oui", plus un ensemble de graphes, spécialisations du graphe question.

Si une base de connaissances est uniquement constituée d'un ensemble de graphes conceptuels sous forme normale définis sur un support, la réponse à une requête peut s'effectuer uniquement grâce à l'opération de projection. Le graphe question peut être démontré à partir de la base s'il existe une projection de ce graphe dans la base. Chaque résultat de projection correspond alors à une spécialisation du graphe question.

Si la base de connaissances contient des définitions de types ou des règles, l'opération de projection seule n'est plus suffisante pour répondre à des requêtes.

Prise en compte de définitions de types

Dans ce paragraphe, une base de connaissance est constituée d'une base de faits (un ensemble de graphes conceptuels quelconques) et d'une base de définitions de types.

Un algorithme de réponse à une question, utilisant la projection classique et fonctionnant par saturation de la base de faits, est proposé dans ([CAR 94], [CAR 96]) (cf figure 74). La base de faits est fusionnée (ie, les sommets concepts ayant même marqueur individuel sont fusionnés), elle ne contient alors que des graphes sous forme normale. Chacun des concepts de la base de faits ou de la question, dont le type est défini, est expansé (ie, un concept est remplacé par le graphe de définition de son type). La question expansée est alors projetée sur les graphes de la base de fait fusionnée et expansée.

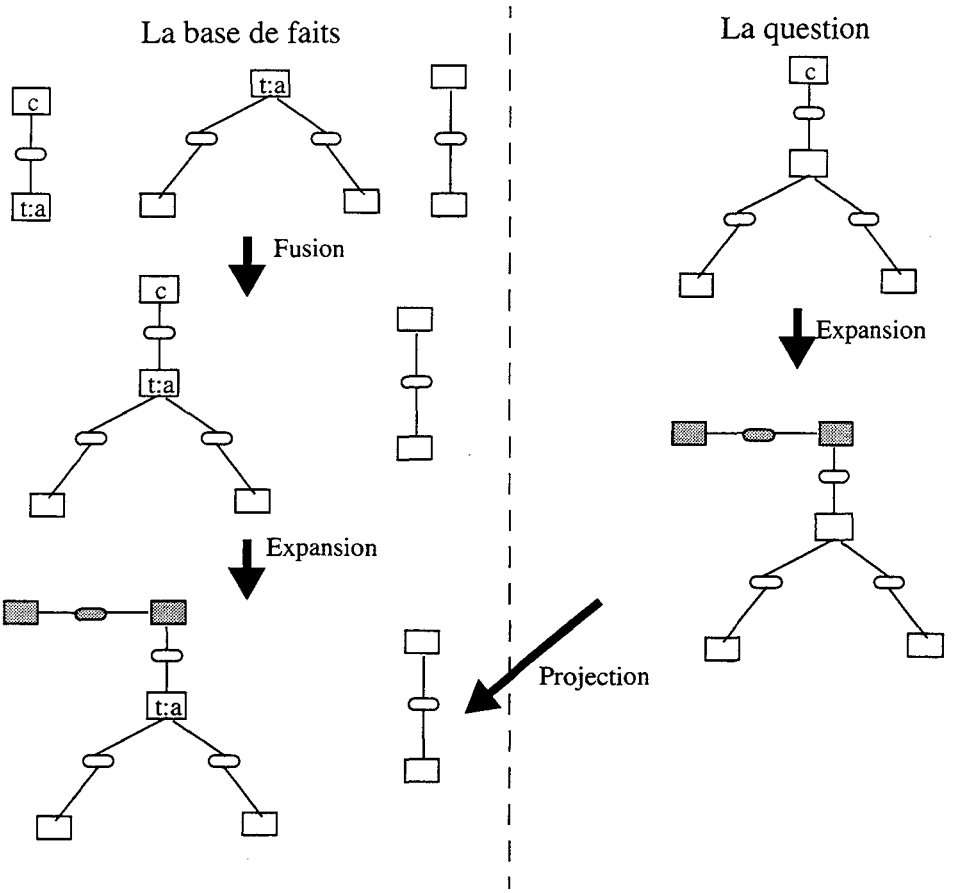


Figure 74. Réponse à une requête sur une base de connaissances contenant des définitions de types.

Un algorithme guidé par les questions, nommé BLUES+, permettant de laisser la base de faits intacte, est alors proposé dans ([CAR 94], [CAR 96]). La question est expansée puis éclatée en sous-questions. Chacune des sous-questions est alors D-projetée dans la base de faits et la base de définition. Les résultats des D-projections sont alors recomposés par fusion pour reconstituer la réponse finale.

Un graphe conceptuel question q est D-projetable sur l'union de la base de faits et de la base de définitions $BF \cup Bdef$ si et seulement si:

- il existe une projection de q dans BF

- ou il existe une projection de q dans un des graphes de définition de Bdef et le concept [t:*] avec t le type défini est D-projetable dans BF ∪ Bdef

Dans le cadre du système ROCK ([CAR 94], [CAR 96]), B. Carboneill a également étudié un ensemble d'heuristiques permettant de réduire la complexité de la recherche. Nous ne nous attardons cependant pas sur ce travail car ROCK n'est pas un système complet et nous avons besoin d'un mécanisme de démonstration qui soit correct et complet.

Prise en compte de règles de graphes

Dans ce paragraphe, une base de connaissance est constituée d'une base de faits (un ensemble de graphes conceptuels quelconques) et d'une base de règles.

Dans [SAL 96], une règle de graphes conceptuels est une règle d'inférence du type "Si G1, alors G2", où G1 et G2 sont des graphes conceptuels simples avec possibilité de liens de coréférence entre G1 et G2. E. Salvat définit une telle règle, notée $G1 \Rightarrow G2$, comme un couple d'abstractions n-aires ($\lambda x1...xnG1$, $\lambda x1...xnG2$). x1,...,xn sont les points d'attache de la règle, ils correspondent aux concepts coréférents de la règle. L'interprétation logique d'une règle de graphes ($\lambda x1...xnG1$, $\lambda x1...xnG2$) est la suivante:

$$\forall x_1 ... \forall x_n (\phi(\lambda x_1 ... x_n G1) \rightarrow \phi(\lambda x_1 ... x_n G2)) \quad (51)$$

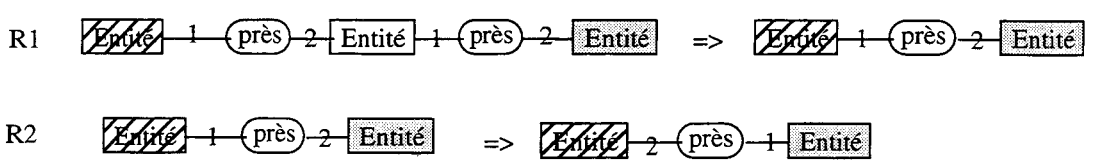


Figure 75. Deux exemples de règles de graphes

La figure 75 donne deux exemples de règles R1 et R2 exprimant la transitivité et la symétrie de la relation près. Elles correspondent en fait à des conditions suffisantes de l'existence d'une relation près. Les interprétations logiques des règles R1 et R2 sont respectivement les suivantes:

$$\forall x \forall y (\exists z (entite(x) \wedge entite(y) \wedge entite(z) \wedge pres(x, z) \wedge pres(z, x)) \rightarrow pres(x, y)) \quad (52)$$

$$\forall x \forall y (entite(x) \wedge entite(y) \wedge pres(x, y) \rightarrow pres(y, x)) \quad (53)$$

L'application de règles en chaînage avant permet d'enrichir explicitement la base de faits. Il est alors possible de trouver une réponse à une requête en projetant cette dernière dans la base de faits saturée par l'application de toutes les règles.

Une règle $R:G1 \Rightarrow G2$ s'applique en chaînage avant sur un graphe G s'il existe une projection P de G1 dans G. Le graphe résultant R(G) est obtenu à partir de G et G2 en fusionnant chaque xi de G1 avec l'image du xi de G2 par P. E. Salvat a montré que le mécanisme d'application des règles en chaînage avant est consistant est complet

vis à vis de la démonstration logique.

Par exemple, la base faits de la figure 76 est saturée par l'application des règles R1 et R2 de la figure 75. Le résultat de cette saturation est donné figure 77. On peut alors répondre à la requête de la figure 78 en projetant celle-ci sur la base de faits saturée.

L'inconvénient d'un mécanisme en chaînage avant est la saturation de la base de faits: quelles conditions sur la base de faits et quel ordre d'application des règles va garantir l'arrêt du processus d'application?

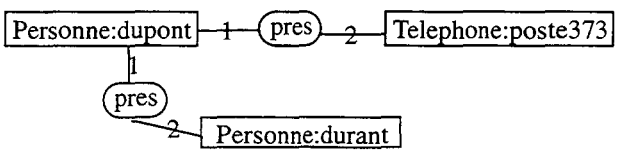


Figure 76. La base de faits

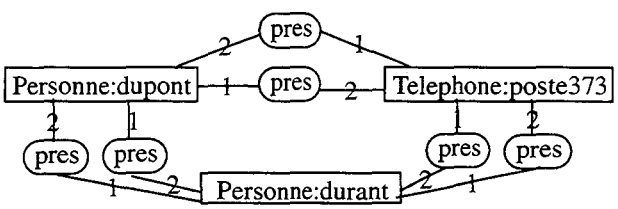


Figure 77. La base de faits saturée

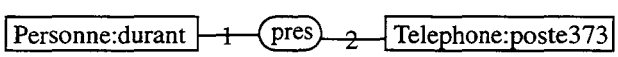


Figure 78. Une requête Q

Pour répondre à une requête par un mécanisme de chaînage arrière, la projection se révèle inadaptée. E. Salvat ([SAL 96]) définit une opération d'unification entre une requête et une règle, qui permet de déterminer si la règle peut produire une spécialisation de la requête. Pour rechercher l'unification, plutôt que de diviser la requête en sous-requêtes élémentaires composées d'une seule relation et ses concepts voisins, il propose de chercher les sous-graphes les plus grands possibles pouvant être traités comme un tout. De tels sous-graphes sont appelés des pièces (cf [SAL 96] pour plus de détails sur la façon de déterminer les pièces). S'il y a une unification entre une requête Q et une règle R, alors il est possible de construire une autre requête Q' telle que R(Q') (ie, l'application de R sur Q' en chaînage avant) soit une spécialisation de Q. Il faut alors essayer de répondre à la requête Q' en l'unifiant avec une autre règle ou avec un graphe de la base de faits. Un graphe fait est vu comme une règle dont l'hypothèse est le graphe vide (noté

$G_{\square}$). On dit que la résolution d'une requête se termine s'il est possible de construire par unifications une séquence de requêtes se terminant par $G_{\square}$. En gardant la trace de ces unifications successives, un graphe spécialisation de la requête initiale Q peut être reconstitué.

Si on veut répondre à la requête Q de la figure 78 à partir de la base de faits de la figure 76, il n'est cette fois plus nécessaire de modifier la base de faits. L'unification de Q avec R1 produit la requête intermédiaire Q1(cf figure 79). L'unification de Q1 avec la base de faits produit la requête intermédiaire Q2. L'unification de Q2 avec R2 produit la requête intermédiaire Q3. L'unification de Q3 avec la base de faits produit enfin $G_{\square}$.

E. Salvat a montré que le mécanisme d'application des règles en chaînage arrière est consistant est complet vis à vis de la démonstration logique. Il a étendu les règles de graphes et les mécanismes d'application associés, aux graphes emboîtés ([SAL 97]).

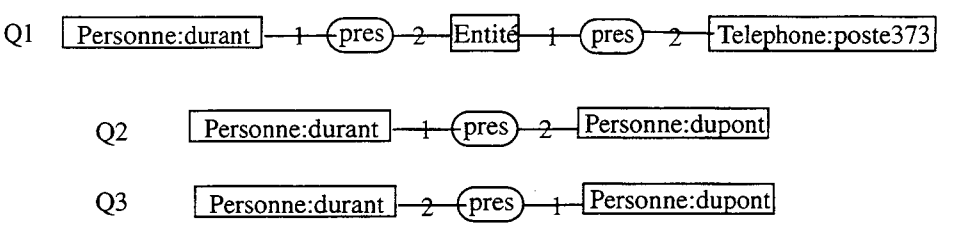


Figure 79. Requêtes intermédiaires

Synthèse

Que l'on considère une base de connaissances contenant des définitions de types ou une base de connaissances contenant des règles de graphes, la réponse à une requête peut se faire suivant deux principes:

- en chaînage avant: la base de faits est enrichie puis la réponse à la requête est trouvée grâce à une opération de projection; l'algorithme de démonstration ne prend pas en compte les définitions de type ou règles de graphes.
- en chaînage arrière: la base de faits n'est pas modifiée, un algorithme spécifique permet de directement trouver la réponse à la requête; l'algorithme de démonstration prend directement en compte soit les définitions de type, soit les règles de graphes.

Il est avantageux d'utiliser le chaînage arrière car on ne doit pas modifier la base de faits et on n'est ainsi pas confronté aux problèmes liés à l'arrêt du processus d'enrichissement de la base. Quant à l'avantage du chaînage avant, c'est d'obtenir une description complète des connaissances, directement lisible, sans avoir de nouveau à considérer les définitions de types ou les règles de graphes. De plus, dans le cas du chaînage avant, le principe de démonstration est directement basé sur la projection.

6.2. Démonstration des états et événements

Dans notre cas, la base de connaissances est constituée d'un point-situation et d'un ensemble de lois statiques (exprimées par des métaconcepts définitions associés à des types du domaine). Les requêtes sont les états et les événements. Contrairement aux approches précédemment présentées, une réponse à une requête peut être non car nous prenons en compte une forme de négation dans les point-situations. Cette forme de négation est le monde clos et les algorithmes de démonstration présentés dépendent de cette hypothèse. De plus, le principe de démonstration présenté ne prend pas en compte les lois du domaine. Nous nous positionnons ainsi dans une approche chaînage avant de réponse à une requête. Enfin, nous supposons que le point-situation initial, à partir duquel la simulation est lancée, est consistant[1]. Un point-situation est ***consistant*** s'il est consistant au sens logique habituel et s'il vérifie toutes les lois du domaine. Le rétablissement de consistance des autres point-situations, résultats de la simulation, sera réalisé par l'opération de mise à jour. Cette dernière sera explicitée dans le chapitre suivant.

6.2.1. Démonstration des états

Nos choix de représentation sont les suivants:

- dans les point-situations, la négation est représentée implicitement par une hypothèse du monde clos et une hypothèse du domaine clos spécifie que seuls les référents cités existent (eg, le point-situation de la figure 80 exprime que l'opérateur dupont est dans la salle de contrôle, qu'il n'existe pas d'autre opérateur et qu'il existe une pièce, le bureau du directeur, dans laquelle il n'y a personne).
- la négation est représentée explicitement par des contextes négatifs ovales dans les états (eg, l'état de la figure 81 exprime qu'il existe un opérateur qui n'est pas dans une pièce).

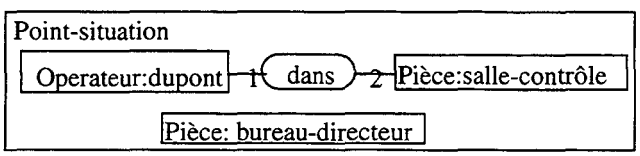


Figure 80. Représentation d'un point-situation

1. Une détection automatique d'inconsistance dans le point-situation initial peut être réalisée en vérifiant une par une les lois du domaine.

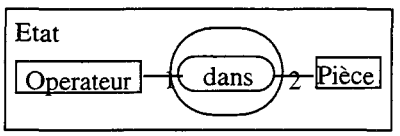


Figure 81. Représentation d'un état

L'algorithme 3 permet de prouver (ie, la valeur de vérité ***V*** est vraie) ou de réfuter (ie, la valeur de vérité ***V*** est fausse) un état ***E*** à partir d'un point-situation ***PS***. La valeur de vérité ne peut pas être indéterminée car on fait l'hypothèse du monde clos dans le point-situation (ie, ce qui n'est pas vrai est supposé faux). L'état E est composé d'une partie positive ***P*** (ie, l'ensemble des sommets en dehors des contextes) et d'un ensemble de contextes négatifs ***Ec***, à partir desquels sont construites les parties négatives ***N*** de l'état. L'ensemble ***S*** de solutions correspond aux différentes applications possibles, résultats de la démonstration de l'état. Chaque élément de S est une application de l'ensemble des sommets de la partie positive de E dans l'ensemble des sommets du graphe décrivant PS. L'algorithme de démonstration des états repose sur l'opération de projection et sur un principe de négation par échec. ***projeter*** renvoie toutes les solutions obtenues en backtrackant sur les points de choix de l'algorithme 2[1]

Algorithme 3: Démontrer(E,PS)

```
Début
    E=(P,Ec)
    S ← ∅
    Si  P ≠ ∅
        Sp ← projeter(P, PS)
        Si Sp ≠ ∅
            Si Ec ≠ ∅
                Pour tout rp ∈ Sp
                    rp · V ← vrai
                    Tant que rp · V ≠ faux et Ec ≠ ∅
                        soit c ∈ Ec
                        construire un graphe N composé des sommets contenus dans c
                        et de l'image par rp des concepts, non contenus dans c,
                        voisins des relations contenues dans c
                        Sn ← projeter(N, PS)
                        Si Sn ≠ ∅
                            rp · V ← faux
                        Fsi
                        Ec ← Ec – c
                    Ftant que
                    Si rp · V ≠ faux
                        S ← S + rp
                    Fsi
```

1. En réalité, l'algorithme de projection que nous utilisons n'est pas strictement le même que l'algorithme 2 car nous permettons d'emboîter des graphes dans des concepts de type proposition. La projection utilisée est une projection sur des graphes emboîtés positifs non typés. Afin de déterminer si un concept c1 de type proposition est l'image par une projection p d'un concept c2 de type proposition, il faut déterminer récursivement si le graphe emboîté dans c1 est l'image par p du graphe emboîté dans c2.

```
                    Fpour
                sinon
                 S ← Sp
                Fsi
            Fsi
            Si  S ≠ ∅
                    V ← vrai
            sinon
                    V ← faux
            Fsi
        sinon
         V ← vrai
            Tant que  V ≠ faux  et  Ec ≠ ∅
                soit  c ∈ Ec  et N le graphe contenu dans c
                Sn ← projeter(N, PS)
                Si Sn ≠ ∅
                        V ← faux
                Fsi
                 Ec ← Ec – c
            Ftant que
        Fsi
        Retourner (V,S)
Fin
```

Il est à noter que l'algorithme 3 se limite à donner la valeur de vérité d'un état et éventuellement un ensemble de spécialisations, résultats de la démonstration de cet état. Il faut ensuite que l'algorithme de simulation:

- propage les coréférences dans le reste du comportement (ie, instancier le comportement).
- interprète la signification de plusieurs spécialisations, résultats de démonstration.

Par exemple, dans le comportement de la figure 22, supposons que dupont soit un opérateur et qu'il soit dans la salle de contrôle, alors la précondition du comportement sera spécialisée en remplaçant le marqueur générique du concept opérateur par le marqueur individuel dupont. Ce sera cette précondition spécialisée qui sera alors testée pour vérifier si le comportement est applicable. L'algorithme de simulation a "propagé" la coréférence de la condition vers la précondition. La façon dont sont propagées les coréférences sera détaillée dans la troisième partie de la thèse.

Si au moins un des concepts de la partie positive de l'état est générique, il peut y avoir plusieurs spécialisations, résultats de la démonstration de l'état. Plusieurs spécialisations sont interprétées différemment suivant la nature de l'état qui est testé. Ces différentes interprétations sont données par la sémantique de l'ontologie comportementale. Dans le cas du comportement de la figure 22, s'il y a deux opérateurs qui sont dans la salle de contrôle, les deux doivent exécuter le comportement. En revanche, si l'opérateur dupont possède deux téléphones, soit il utilisera le premier, soit le second pour exécuter son comportement; on a ainsi une indétermination. La façon dont sont interprétées plusieurs spécialisations, résultats de démonstration, sera également détaillée dans la troisième partie de la thèse.

La figure 82 illustre comment l'algorithme 3 permet de démontrer l'état de la

figure 81 à partir du point-situation de la figure 80.

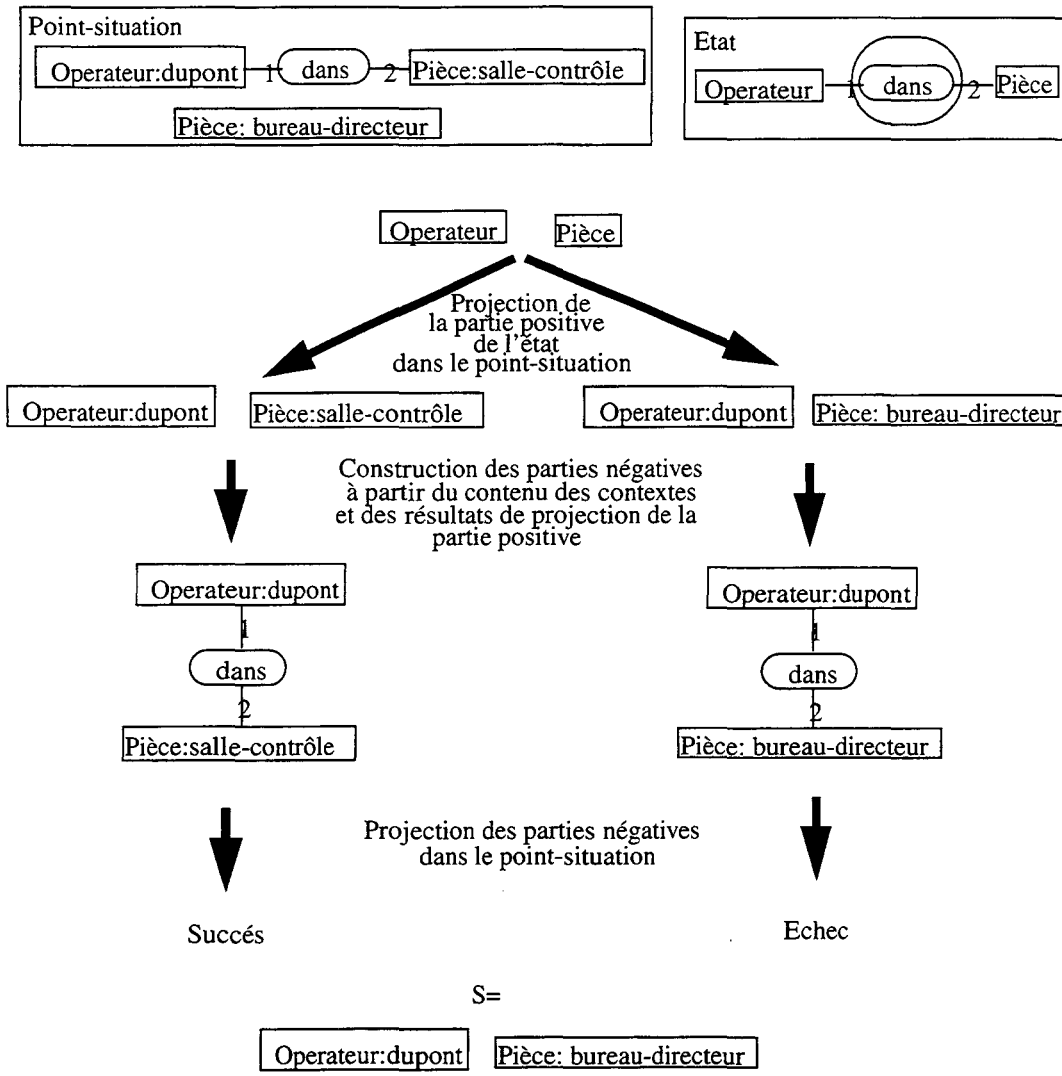


Figure 82. Application de l'algorithme de démonstration d'un état sur un exemple

6.2.2. Démonstration des événements

Nos choix de représentation sont les suivants:

- les point-situations sont représentés comme nous l'avons décrit dans 6.2.1, la démonstration d'un événement nécessite cependant de considérer deux point-situations consécutifs (par exemple, ceux de la figure 83)
- des graphes propriétés du monde positifs (ie, sans aucune forme de négation) sont emboîtés à l'intérieur de l'événement avec deux types d'emboîtements: devient-vrai ou devient-faux (cf par exemple, événement de la figure 84).

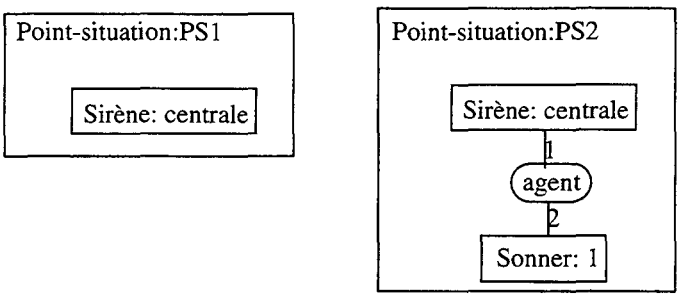


Figure 83. Deux point-situations consécutifs

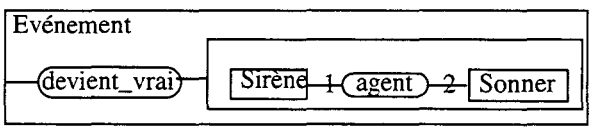


Figure 84. Représentation d'un événement

L'algorithme 4 permet de prouver (ie, la valeur de vérité ***V*** est vraie) ou de réfuter (ie, la valeur de vérité ***V*** est fausse) un événement basique ***Ev*** à partir de deux point-situations ***PS1*** et ***PS2***. Un événement basique Ev est décrit par un graphe propriété du monde ***G*** qui devient vrai (***Te=FV***) ou qui devient faux (***Te=VF***). L'ensemble ***S*** de solutions correspond aux différentes applications possibles, résultats de la démonstration de l'événement. Chaque élément de S est une application de l'ensemble des sommets du graphe décrivant Ev dans l'ensemble des sommets du graphe décrivant PS. Comme l'algorithme de démonstration des états, cet algorithme repose sur l'opération de projection et sur un principe de négation par échec.

Algorithme 4: Démontrer(Ev,PS1,PS2)
Début

```
Ev=(Te,G)
S ← ∅
V ← vrai
S1 ← projeter(G, PS1)
S2 ← projeter(G, PS2)
Si Te=FV
        Si S1 = ∅ et S2 ≠ ∅
                S ← S2
        sinon
                V ← faux
        Fsi
Fsi
Si Te=VF
        Si S1 ≠ ∅ ou S2 = ∅
                S ← S1
        sinon
                V ← faux
        Fsi
```

Fsi
Retourner (V,S)
Fin

La figure 85 illustre comment l'algorithme 4 permet de démontrer l'événement de la figure 84 à partir des point-situations de la figure 83. Comme dans le cas de la démonstration des états, il faut également que l'algorithme de simulation propage les coréférences dans le reste du comportement et interprète différentes spécialisations, résultats de la démonstration de l'événement.

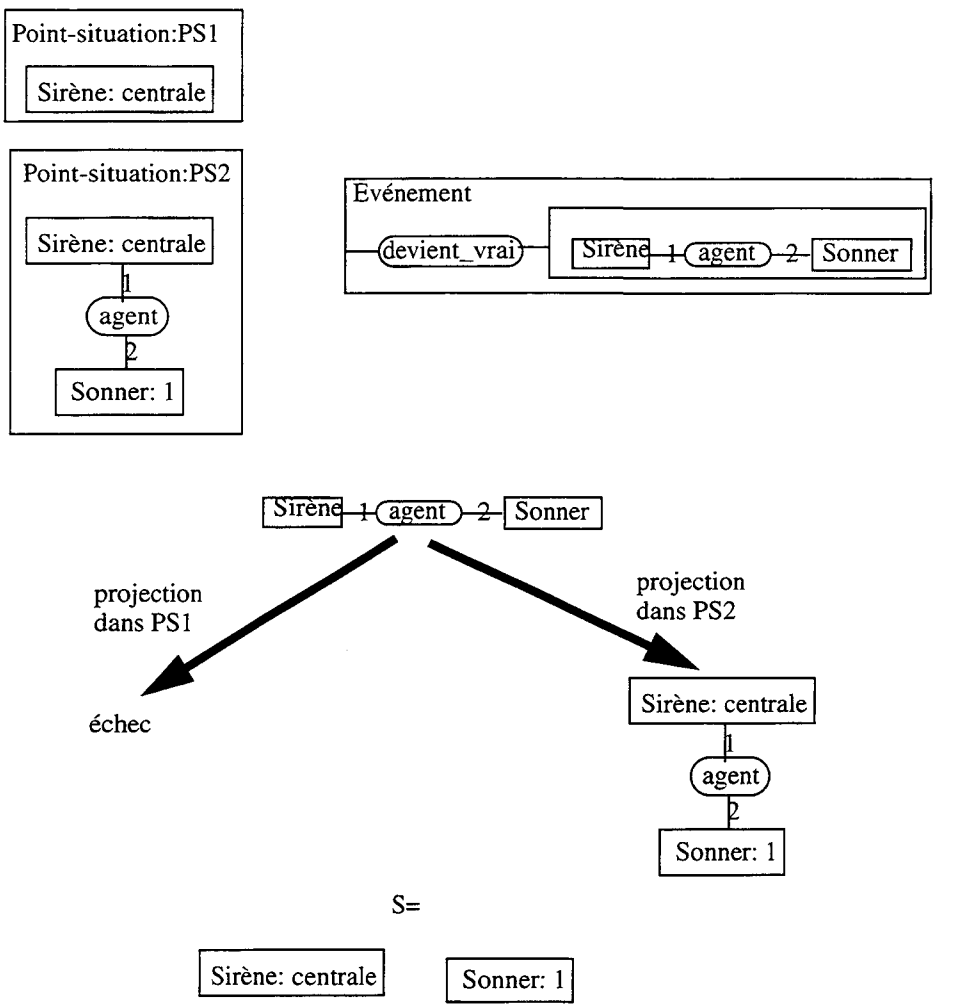


Figure 85. Application de l'algorithme de démonstration d'un événement sur un exemple

6.3. Conclusion

	Monde clos dans les point-situations	Monde ouvert dans les point-situations
Démonstration sans prise en compte des lois statiques (chaînage avant)		
Démonstration avec prise en compte des lois statiques (chaînage arrière)		

Figure 86. Diverses hypothèses

Le tableau de la figure 86 résume les hypothèses qui peuvent être faites concernant la représentation des point-situations et le principe de démonstration. La case grisée indique les hypothèses que nous posons actuellement.

La première faiblesse de notre approche est la représentation de la négation dans les point-situations par une hypothèse du monde clos. Bien que cette hypothèse permette d'économiser l'écriture des propriétés fausses et d'implémenter facilement des opérations de raisonnement basées sur la projection, elle oblige à décrire le monde de façon complète. Pour gérer de l'indétermination, il est nécessaire de traiter plusieurs point-situations en parallèle.

La deuxième faiblesse de notre approche est la non prise en compte des lois statiques dans le principe de démonstration. Bien qu'il soit ainsi possible de baser la démonstration sur l'opération de projection, il est nécessaire d'apporter beaucoup de modifications aux point-situations à chaque prise en compte d'un nouvel effet et de démarrer la simulation à partir d'un point-situation consistant.

Des travaux récents ont exploré des systèmes de démonstration n'exploitant pas uniquement l'opération de projection. Ces travaux ont pour objectifs d'obtenir des systèmes de démonstration implémentables:

- Dans [KER 97], G. Kerdilès et E. Salvat enrichissent le langage des graphes conceptuels simples avec l'implication et la négation et proposent un système de déduction analytique efficace qui combine les tableaux analytiques ([SCH 90]) et la projection.
- Dans [LAP 97], S. Lapalut étend le langage des graphes conceptuels simples en imposant une forme normale pour l'utilisation des contextes négatifs de J. Sowa et propose des algorithmes implantant des machines de grammaires de graphes permettant de répondre à des requêtes sur les graphes étendus.

De tels travaux offrent des perspectives intéressantes pour augmenter le

pouvoir d'expression dans les point-situations ou pour prendre en compte les lois statiques dans le principe de démonstration. Mais, il restera à définir l'algorithme de mise à jour correspondant. Notre problématique n'est en effet pas uniquement la démonstration des états et des événements, mais aussi la mise à jour des point-situations avec les effets des actions.

Chapitre 7

Mise à jour

La mise à jour consiste, à partir d'une description courante du monde et d'une information nouvelle signalant un changement du monde, à prédire la description du monde résultante. Par exemple, si l'interrupteur d'une lampe se trouve sur la position off et qu'une personne appuie sur l'interrupteur, alors dans la description du monde prédite, l'interrupteur se trouvera sur la position on et la lampe sera allumée. Au cours de la simulation des comportements, les changements du monde proviennent de l'exécution des actions. Un point-situation initial permet de démarrer la simulation. Les autres point-situations des séquences résultats sont prédits par l'algorithme de simulation grâce à des opérations de mise à jour. Chaque mise à jour correspond à la prise en compte d'un effet d'une action représenté par une transition.

Une mise à jour est différente d'une révision ([KAT 91]), qui, elle, consiste, à partir d'un ensemble de croyances sur le monde et d'une information nouvelle signalant un changement de croyance, à prédire l'ensemble des croyances résultant. Dans le cas d'une mise à jour, le monde évolue alors que dans le cas d'une révision, ce sont les croyances sur un monde statique qui évoluent.

Dans ce chapitre, nous décrivons d'abord des approches de mise à jour basées sur des représentations différentes du monde et des changements du monde. Il découle de cette étude bibliographique que la mise à jour nécessite la représentation d'un certain nombre de connaissances sur la façon dont le monde évolue. Nous introduisons la notion de loi dynamique grâce à laquelle nous représentons de telles connaissances sur l'évolution du monde. Finalement, nous détaillons un principe de mise à jour le monde, les changements du monde et les lois dynamiques étant représentées avec des graphes conceptuels.

7.1. Différentes approches de mise à jour

Pour chacune des approches de mise à jour présentée, nous décrivons la façon dont le monde et les changements du monde sont représentés, la forme des résultats de mise à jour et s'il en existe un l'algorithme correspondant. Nous concluons en comparant les différentes approches. Les critères de comparaison sont l'existence ou non d'un algorithme de mise à jour, la difficulté de modélisation des actions et le réalisme des résultats.

7.1.1. Approche STRIPS

L'approche STRIPS ([FIK 88]) suppose que le monde est décrit par une théorie de la logique des prédicats du premier ordre. Dans les modèles d'actions, les effets sont décrits par une liste d'ajouts (add-list) représentant un ensemble de formules du premier ordre devant être ajoutées au monde courant et par une liste de retraits (del-list) représentant un ensemble de formules du premier ordre devant devenir fausses, c'est à dire devant être retirées du monde courant. L'hypothèse de STRIPS consiste alors à dire que toutes les formules n'étant dans aucune des deux listes ne changent pas de valeur de vérité au cours de la mise à jour.

L'opération de mise à jour est alors simple, puisqu'il suffit de changer la valeur de vérité de certaines formules. Dans l'exemple de la figure 87, Wf est la mise à jour de Wd, p(a) et p(b) changent de valeur de vérité et p(c) persiste de Wd à Wf car elle n'est ni dans la liste d'ajouts, ni dans la liste de retraits.

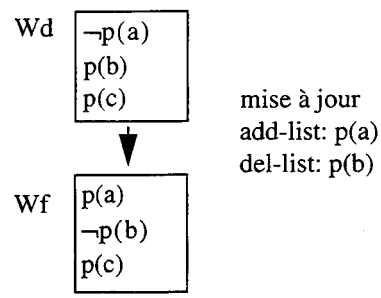


Figure 87. Un exemple de mise à jour avec des opérateurs STRIPS

7.1.2. Approches du changement minimal

Les approches basées sur le changement minimal considèrent une théorie S du premier ordre décrivant le monde courant, un ensemble de formules C décrivant les effets de l'action sous forme d'états résultats et un ensemble de formules protégées P représentant les lois du domaine. L'objectif de ces approches est de calculer le monde "le plus proche" du monde courant dans lequel les effets de l'action sont vrais, ceci, en ne modifiant pas les formules protégées.

PWA

M.L. Ginsberg et D.E. Smith ([GIN 88a]) proposent une méthode, nommée PWA (Possible Worlds Approach), permettant de calculer une nouvelle théorie T (appelée monde possible) proche de S et contenant C. Tout sous-ensemble $T \subseteq S \cup C$ est un monde possible si et seulement si:

1) $C \subseteq T$, toutes les formules de C sont vraies dans la nouvelle théorie

2) $P \cap S \subseteq T$, les formules protégées continuent à être vraies

3)T est consistant

4)T est maximal au sens de l'inclusion signifiant que T est la théorie "la plus proche" de S, le critère de minimalité du changement est ainsi défini en terme de nombre de formules ôtées de la théorie courante.

M.L. Ginsberg et D.E. Smith proposent un algorithme permettant de calculer les différents mondes possibles, résultats d'une mise à jour. Soit q une formule de C et $S_1,...,S_n$ n ensembles de faits de S, permettant de démontrer $\neg q$, l'algorithme de calcul des mondes possibles procède en deux étapes:

- il calcule tous les mondes W de la forme $W=S \cup C-\{s_1,...,s_n\}$ avec $s_i \in S_i$
- il ne garde que les W qui ne sont pas sous-ensembles d'un autre W

PMA

M. Winslett ([WIN 88]) a cependant montré dans un article critiquant PWA que la minimisation du changement en terme de nombre de formules n'est pas satisfaisante car deux théories ayant le même modèle peuvent ne pas être mises à jour de la même façon étant donné le même ensemble d'effets. M. Winslett définit alors une nouvelle approche de mise à jour nommée PMA (Possible Models Approach) en cherchant à minimiser une distance définie entre ensembles de modèles et non entre ensembles de formules. Les modèles considérés sont des modèles de Herbrand, les théories considérées étant du premier ordre (cf [ALL 94] et [DEL 86] pour la définition d'un modèle de Herbrand).

Dans l'approche PMA, l'ensemble des modèles S_T de la nouvelle théorie T est la réunion des ensembles Incorporate (C,M) pour M modèle de S, avec Incorporate(C,M) l'ensemble des modèles M' tels que:

- C et P sont vraies dans M'
- M' diffère de M par le moins d'atomes possibles

M. Winslett ne propose cependant pas d'algorithme permettant de construire la nouvelle théorie T à partir de S_T. La méthode PMA reste donc simplement au niveau de la théorie des modèles.

Modèles explicites de transitions

Dans [COR 94], M.O. Cordier et P. Siegel proposent de représenter des connaissances supplémentaires qui vont permettre de donner de meilleurs résultats de mise à jour que la PMA. Ces connaissances sont des modèles explicites de transitions. Un modèle de transition est un ensemble de transitions, chacune étant représentée par un couple de formules logiques <F1,F2>. Ils considèrent deux types de transitions: des transitions certaines TC:<F1,F2> et des transitions attendues TA:<F1,F2>. Les transitions attendues sont ordonnées dans un ordre partiel suivant leur certitude. Un tel modèle de transition permet d'exprimer:

-des formules protégées TC:<True,F>

-des formules immuables TC:<F,F>

-des formules persistantes TA:<F,F>

-des formules contingentes TC:<F1,F2>

-des formules par défaut TA:<True,F>

L'ordre sur les transitions attendues permet par exemple d'exprimer des priorités sur la persistance des formules. Le critère de minimalité de distance entre modèles prend alors en compte ces priorités.

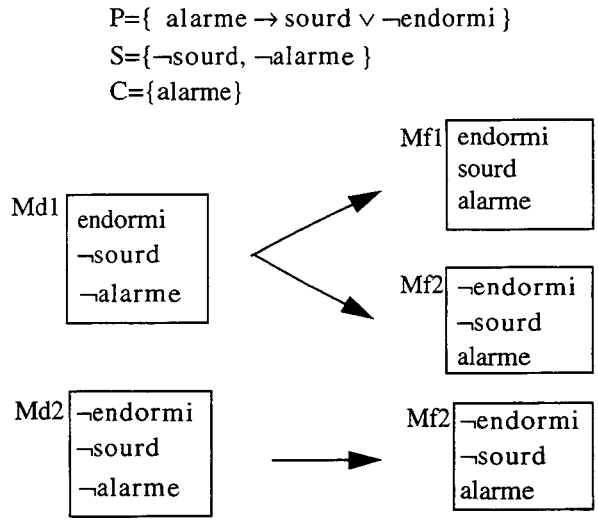


Figure 88. Un exemple de mise à jour avec PMA sans priorités sur la persistance des faits

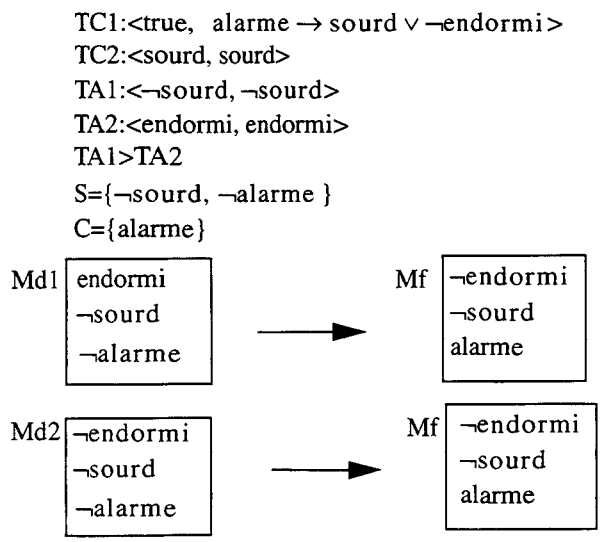


Figure 89. Prise en compte de modèles explicites de transitions

Dans l'exemple de la figure 88, une alarme est déclenchée dans l'usine. Il existe un employé de l'usine qui n'est pas sourd, mais on ne sait pas s'il est endormi ou non. Une loi du domaine stipule cependant que lorsque l'alarme est déclenchée, un employé est soit sourd, soit éveillé. Avec la PMA, il y a deux modèles possibles résultats de mise à jour: Mf1 et Mf2. Avec une représentation explicite des modèles de transition, il n'y a qu'un seul modèle résultat de mise à jour car il a été possible de représenter que la non surdité est plus persistante que l'endormissement (cf figure 89).

Distinction faits de base, faits déduits

F. Garcia ([GAR 93]) structure la description du monde en différenciant faits de base et faits déduits. Dans sa description du monde, une propriété du monde est descriptible par un ensemble de variables d'état (comme la température ou la pression en physique). Chaque variable d'état est décrite par un prédicat, appelé fait de base. Les faits déduits sont alors des faits définis à partir de faits de base par des relations de déduction. F. Garcia propose de rechercher la minimalité du changement relativement aux faits de base, sans tenir compte des faits déduits.

Dans l'exemple de la figure 90, les faits de base sont exprimés avec des prédicats "dans" et les faits déduits avec des prédicats "près". Le prédicat "près" est défini par la loi du domaine qui stipule que deux entités sont proches l'une de l'autre si elles sont dans la même pièce. Une autre loi du domaine énonce une contrainte de cohérence sur les prédicats dans: une même entité ne peut pas être dans deux pièces différentes. Dans le monde de départ, dupont est dans la pièce p1 et il y a un téléphone dans la pièce p2. Sachant qu'il y a trois pièces dans le monde p1, p2 et p3, dupont n'est ni dans p2, ni dans p3 et il n'est ainsi pas près du téléphone. La nouvelle information est que dupont quitte la pièce p1. En calculant la minimalité du changement uniquement sur les faits de base, il y a deux résultats de mise à jour Mf1 et Mf2: soit dupont est dans p2, soit dupont est dans p3. Avec la PMA, il n'y aurait eu qu'un seul résultat de mise à jour Mf2 car le calcul de minimalité aurait été réalisé sur tous les faits décrivant le monde. Ce résultat de PMA ne semble pas réaliste car ce n'est pas parce qu'il y a un téléphone dans p2 que dupont va plutôt se rendre dans p3. Les modèles Mf1 et Mf2 doivent être considérés à égalité.

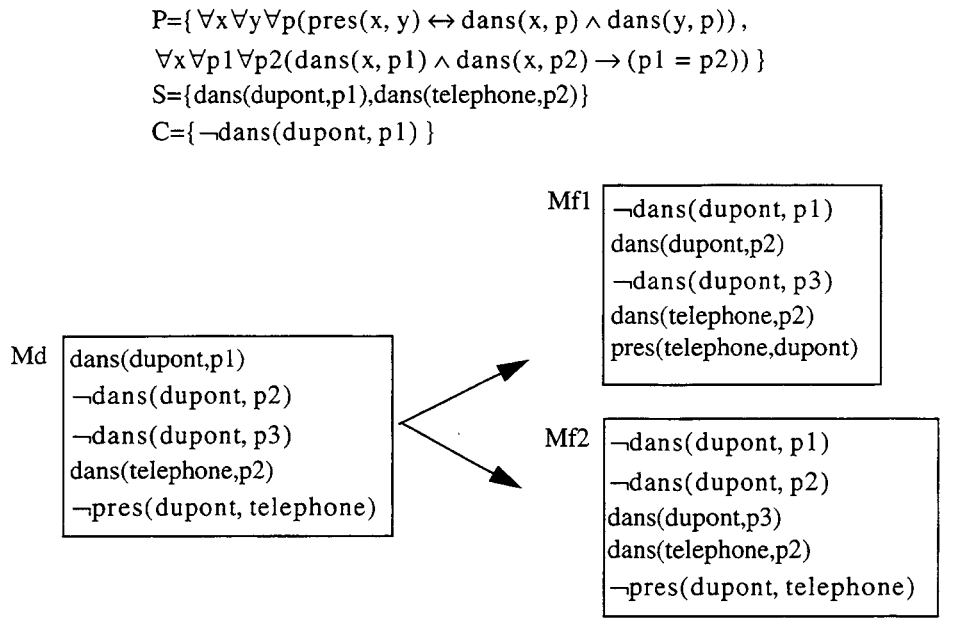


Figure 90. Un exemple de mise à jour avec distinction faits de base, faits déduits

7.1.3. Conclusion

Les approches STRIPS et PWA sont plus satisfaisantes que la PMA car chacune d'elle propose un algorithme qui permet de calculer une nouvelle description du monde représentée dans le même langage que la description courante du monde. Dans le cas de STRIPS, cet algorithme est simple puisqu'il consiste à ajouter ou retirer des formules de la description courante du monde. Cependant, la construction des modèles d'actions STRIPS est difficile pour des domaines complexes. Les modèles d'action doivent non seulement expliciter les effets principaux de l'action mais aussi ses effets secondaires et contextuels. Ces derniers doivent en plus être définis de façon à garantir la consistance de la description du monde après l'exécution de l'action. En ce qui concerne la modélisation des actions, les approches du changement minimal sont plus satisfaisantes que l'approche STRIPS car seuls les effets principaux de l'action doivent être explicités. PMA semble donner des résultats plus réalistes que PWA car le critère de minimalité porte sur les modèles et non sur les formules. Cependant, il est facile de trouver des exemples contre-intuitifs de mise à jour avec PMA. Le problème est en fait qu'il manque de l'information sur la façon dont le monde évolue. Les approches donnant de meilleurs résultats que PMA reposent sur un ensemble de lois du domaine plus d'autres connaissances telles que des modèles explicites de transitions ou une différenciation des faits qui décrivent le monde. Dans notre cas, les connaissances supplémentaires sur l'évolution du monde sont exprimées dans des lois dynamiques du domaine, que nous allons décrire maintenant.

7.2. Les lois dynamiques

L'opération de mise à jour nécessite de prendre en compte l'évolution prévisible du monde. Les approches de changement minimal ne s'appuient que sur une unique information sur l'évolution du monde: le principe de persistance qui stipule que si un changement ne doit pas se produire, il ne se produit pas. Nous avons cependant observé que sans information supplémentaire sur l'évolution du monde, les résultats de mise à jour obtenus sont souvent contre-intuitifs. Comme [COR 94], nous préconisons donc la représentation d'un ***modèle explicite d'évolution du monde***. Notre deuxième objectif est que ce modèle d'évolution permette d'implémenter facilement un ***algorithme de mise à jour*** tels que ceux de [FIK 88] et [GIN 88a].

Les lois du domaine sont exprimées par les métaconcepts, qui définissent les types de concept et de relation du domaine. Ce sont des conditions représentées par des graphes, qui doivent être vérifiées dans chacun des point-situations d'une évolution du monde (ie, à chaque instant de l'évolution du monde). Elles n'expriment cependant pas quelles sont les propriétés du monde qui évoluent et comment elles évoluent. Elles peuvent ainsi être qualifiées de lois ***statiques*** du domaine. Par rapport aux approches de changement minimal précédemment présentées, ces lois statiques sont équivalentes à l'ensemble des formules protégées.

Nous préconisons de modéliser l'évolution du monde par des lois ***dynamiques*** du domaine, associées à un ***principe de persistance***. Les lois dynamiques décrivent comment doivent évoluer les propriétés non immuables entre deux instants consécutifs, ceci en accord avec les lois statiques du domaine. Le principe de persistance stipule alors qu'une propriété, qui n'est pas obligée d'évoluer par l'application d'une transition ou d'une loi dynamique, n'évolue pas.

Dans un premier temps, nous expliquons plus précisément la sémantique d'une loi dynamique. Puis, nous décrivons le rapport entre lois dynamiques et lois statiques.

7.2.1. Sémantique d'une loi dynamique

Une loi dynamique exprime que si une propriété du monde (la condition de la loi, représentée par un état) est vérifiée en deux instants consécutifs de l'évolution du monde, alors un changement du monde (le déclenchement de la loi, représenté par un événement) qui se produit entre ces deux instants entraîne qu'un changement du monde (l'effet de la loi, représentée par une transition) se produise entre les deux mêmes instants. Par exemple, si une personne est entendante, alors une sirène qui sonne réveille cette personne (figure 91). L'état condition est décrit par un graphe pouvant contenir des négations exprimées avec des contextes ovales, emboîté dans le concept état. L'événement déclenchement est décrit par un graphe sans négation emboîté dans le concept événement avec deux types possibles d'emboîtements: vf et fv. La transition effet est décrite par un graphe sans négation emboîté dans le concept transition avec deux types possibles d'emboîtements: vf et fv. Afin d'exprimer une loi dynamique précise, des sommets peuvent être mis en surbrillance dans le graphe emboîté dans la transition.

La loi dynamique est considérée comme un métaconcept, c'est à dire que c'est une abstraction n-aire $\lambda x1,...,xn$ LD. $x1,...,xn$ sont des concepts du graphe emboîté dans la transition coréférents avec des concepts des graphes emboîtés dans l'état ou l'événement, ils sont interprétés par des variables quantifiées universellement.

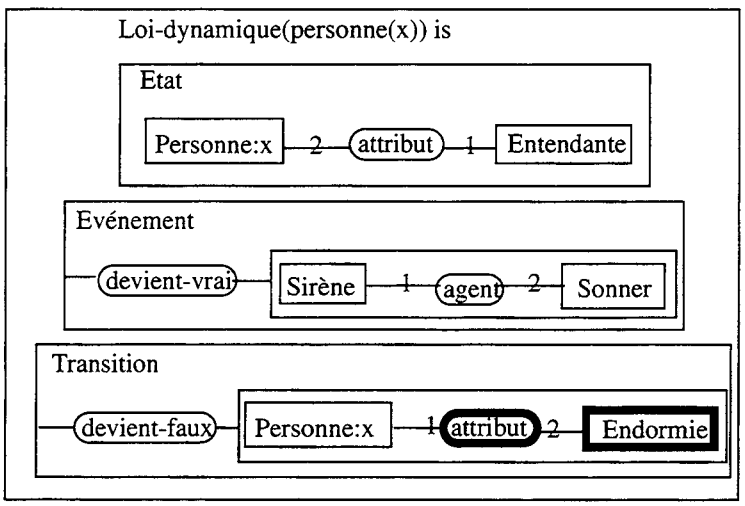


Figure 91. Exemple de loi dynamique

L'interprétation logique à n'importe quel instant clé i de l'évolution du monde, d'une loi dynamique $\lambda x1,...,xn$ LD décrite par un état [Etat:Et], un événement [Evénement:TEv:Ev] et une transition [Transition:TTr:Tr] est, suivant les types d'emboîtements TEv et TTr:

- TEv=vf et TTr=vf

$$\forall x1...\forall xn(\phi(\lambda x1...xnEt, (i-1, i)) \wedge \phi(\lambda x1...xnEv, i-1) \wedge \neg\phi(\lambda x1...xnEv, i) \wedge \phi(\lambda x1...xnTr, i-1) \rightarrow \neg\phi(\lambda x1...xnTr, i)) \quad (54)$$

- TEv=fv et TTr=vf

$$\forall x1...\forall xn(\phi(\lambda x1...xnEt, (i-1, i)) \wedge \neg\phi(\lambda x1...xnEv, i-1) \wedge \phi(\lambda x1...xnEv, i) \wedge \phi(\lambda x1...xnTr, i-1) \rightarrow \neg\phi(\lambda x1...xnTr, i)) \quad (55)$$

- TEv=vf et TTr=fv

$$\forall x1...\forall xn(\phi(\lambda x1...xnEt, (i-1, i)) \wedge \phi(\lambda x1...xnEv, i-1) \wedge \neg\phi(\lambda x1...xnEv, i) \wedge \neg\phi(\lambda x1...xnTr, i-1) \rightarrow \phi(\lambda x1...xnTr, i)) \quad (56)$$

- TEv=fv et TTr=fv

$$\forall x1...\forall xn(\phi(\lambda x1...xnEt, (i-1, i)) \wedge \neg\phi(\lambda x1...xnEv, i-1) \wedge \phi(\lambda x1...xnEv, i) \wedge \neg\phi(\lambda x1...xnTr, i-1) \rightarrow \phi(\lambda x1...xnTr, i)) \quad (57)$$

L'interprétation logique de la loi dynamique de la figure 91 est ainsi la suivante:

$$\forall x(\exists y(\text{personne}(x, (i-1, i)) \wedge \text{attribut}(y, x, (i-1, i)) \wedge \text{entendante}(y, (i-1, i))) \wedge$$

$$\neg(\exists a \exists b(\text{sirene}(a, i-1) \wedge \text{agent}(a, b, i-1) \wedge \text{sonner}(b, i-1))) \wedge$$

$$\exists c \exists d(\text{sirene}(c, i) \wedge \text{agent}(c, d, i) \wedge \text{sonner}(d, i)) \wedge$$

$$\exists z(\text{attribut}(z, x, i-1) \wedge \text{endormie}(z, i-1)) \rightarrow \neg(\exists u(\text{attribut}(u, x, i) \wedge \text{endormie}(u, i)))) \quad (58)$$

Notons que l'interprétation logique des lois dynamiques est similaire à l'interprétation logique des règles de graphes décrites dans [SAL 96] par la présence de quantificateurs universels et d'une implication logique, mais en diffère par la présence d'arguments temporels dans les prédicats et de négations.

7.2.2. Lois dynamiques et lois statiques

Une loi dynamique est toujours issue d'une loi statique; elle correspond à un rétablissement de cohérence exprimé par une transition quand une loi statique est violée par l'occurrence d'un événement dans un état. Plusieurs lois dynamiques sont issues d'une même loi statique. Pour construire l'ensemble des lois dynamiques correspondant à une loi statique, il faut générer tous les événements qui, s'ils se produisent dans un état donné, vont violer la loi statique. Pour chacun de ces couples (événement,état), il reste alors à générer les transitions permettant de restaurer la cohérence.

Le nombre de lois dynamiques issues d'une loi statique exprimée en logique propositionnelle dépend du nombre de propositions de la loi statique. Par exemple, neuf lois dynamiques (figure 92) peuvent être construites à partir de la loi statique suivante:

$$\text{personne} - \text{entendante} \wedge \text{personne} - \text{endormie} \wedge \text{sirene} - \text{sonne} \rightarrow \perp \quad (59)$$

	Etat	Evénement	Transition
1		personne – entendante ∧ personne – endormie	¬(sirene – sonne)
2		personne – endormie ∧ l sirene – sonne	¬(personne – entendante)
3		personne – entendante ∧ sirene – sonne	¬(personne – endormie)
4	personne-endormie	personne-entendante	¬(sirene – sonne)
5	sirene-sonne	personne-entendante	¬(personne – endormie)

	Etat	Evénement	Transition
6	personne-entendante	personne-endormie	¬(sirene – sonne)
7	sirene-sonne	personne-endormie	¬(personne – entendante)
8	personne-entendante	sirene-sonne	¬(personne – endormie)
9	personne-endormie	sirene-sonne	¬(personne – entendante)

Figure 92. Lois dynamiques en logique propositionnelle

Mais en réalité, très peu de lois dynamiques correspondent à des évolutions possibles du monde. Certaines propriétés du monde sont immuables, c'est à dire que rien ne peut les faire évoluer dans le contexte de la simulation. Par exemple, une personne sourde reste toujours sourde. D'autres propriétés ne sont pas immuables, mais restent persistantes quand certains événements se produisent (ie, elles sont indépendantes vis-à-vis de l'événement). Par exemple, une personne ne devient pas sourde parce que la sirène sonne. En revanche, elle pourrait devenir sourde à cause d'un sinistre tel qu'une explosion, la non-surdité n'est en effet pas immuable. Parmi les lois de la figure 92, seule la loi 8 correspond à une évolution possible du monde.

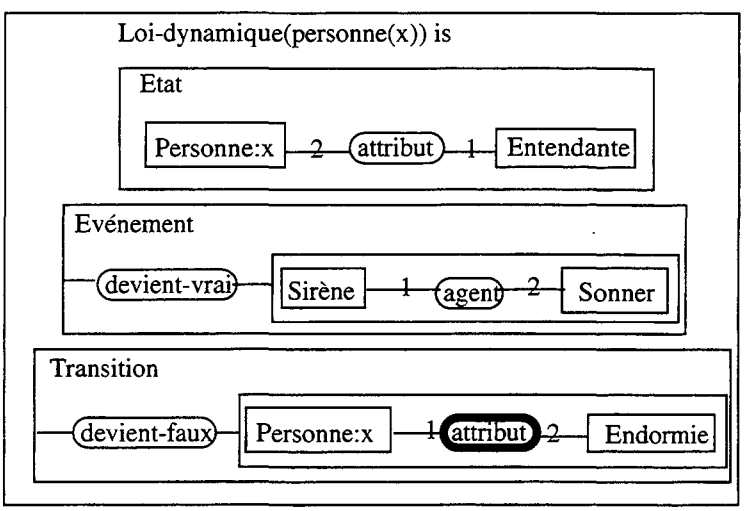


Figure 93. Une évolution du monde impossible

Avec le formalisme des graphes conceptuels, le nombre de lois dynamiques issues d'une loi statique est plus grand qu'avec la logique propositionnelle (ou qu'avec la logique des prédicats), car une propriété est représentée par un graphe (ie, conjonction de prédicats), et non par une seule proposition ou un seul prédicat. Par exemple, la loi 8 de la figure 92 exprimée en logique propositionnelle est équivalente à deux lois exprimées dans le formalisme des graphes conceptuels, suivant la façon dont la cohérence est rétablie (celle de la figure 91 et celle de la figure 93). Seule la loi de la figure 91 correspond cependant à une évolution possible du monde. Un attribut d'endormissement

ne peut en effet pas persister sans être lié à une entité.

En résumé, les lois dynamiques sont plus directement utilisables par l'algorithme de mise à jour que les lois statiques car elles expriment des rétablissements de cohérences entre des instants consécutifs de l'évolution du monde. Et elles sont plus riches en information que les lois statiques car elles correspondent à des évolutions possibles du monde (elles prennent en compte l'aspect immuable des propriétés du monde et l'indépendance de certaines propriétés vis-à-vis d'événements).

Il serait intéressant de concevoir un programme d'assistance à la construction des lois dynamiques à partir des lois statiques. Ce programme combinerait génération automatique des lois dynamiques et choix de l'expert. Nous n'avons pas conçu un tel programme durant cette thèse. Actuellement, les lois dynamiques sont saisies comme des métaconcepts. Nous nous sommes focalisés sur le principe de mise à jour, qui consiste à appliquer en chaînage avant des transitions et des lois dynamiques. Nous allons détailler ce principe maintenant.

7.3. Mise à jour des point-situations

L'opération de mise à jour est divisée en deux étapes: elle consiste à construire un nouveau point-situation à partir du point-situation courant avec les effets de l'action exprimés par des transitions, puis elle consiste à prendre en compte les lois dynamiques afin de rétablir la consistance du nouveau point-situation vis-à-vis des lois du domaine si c'est nécessaire. Cette deuxième étape revient à considérer des effets secondaires et contextuels de l'action indirectement exprimés dans des lois du domaine.

Dans un premier temps, nous détaillons le principe d'application d'une transition, puis le principe d'application d'une loi dynamique. Nous décrivons alors complètement l'opération de mise à jour qui combine applications de transitions et de lois dynamiques.

7.3.1. Application d'une transition

Nos choix de représentation sont les suivants:

- les point-situations sont représentés comme nous l'avons décrit dans 6.2.1 (par exemple, cf celui de la figure 94).
- des graphes propriétés du monde positifs (ie, sans aucune forme de négation) sont emboîtés à l'intérieur de la transition avec deux types d'emboîtements: devient-vrai ou devient-faux; les sommets créés ou supprimés sont mis en surbrillance à l'intérieur de la transition (cf par exemple, transition de la figure 95).

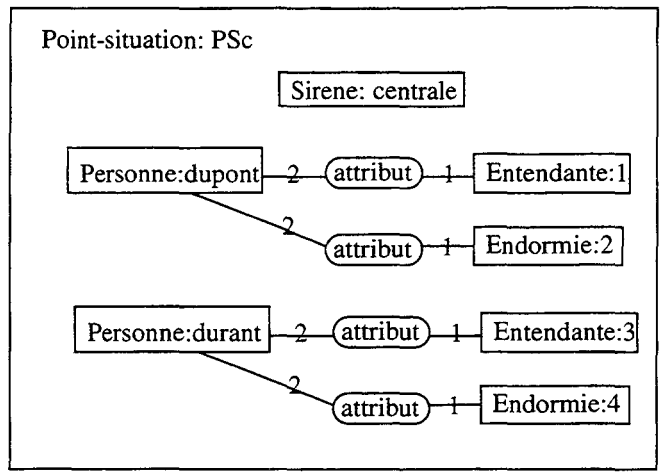


Figure 94. Le point-situation courant

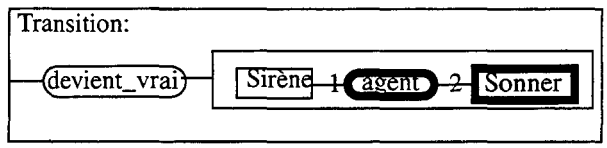


Figure 95. L'effet de l'action exprimé par une transition

L'algorithme 5 permet d'appliquer une transition précise ***T*** (contenant donc des sommets surbrillants) sur un point-situation-courant ***PSc*** afin de construire un nouveau point-situation ***PSn***. Une transition est décrite par un graphe propriété ***P*** qui doit devenir vrai (***Te=devient-vrai***) ou faux (***Te=devient-faux***). Le graphe propriété P contient un ensemble de sommets non surbrillants ***Pns*** et un ensemble de sommets surbrillants ***Ps*** (sommets qui doivent être créés ou supprimés).

Dans le cas de l'emboîtement devient-vrai, si la propriété P est fausse dans le point-situation courant, alors la transition doit être appliquée. La partie non surbrillante Pns de P est projetée dans le point-situation courant. Une application rpns, résultat de la projection de Pns dans PSc, est appliquée à P afin de construire une spécialisation de P. Dans cette spécialisation, les concepts génériques sont uniquement des concepts abstraits (pas de concepts génériques concrets dans la partie surbrillante d'une transition devient-vrai); leurs référents génériques sont numérotés. Le graphe obtenu est alors ajouté au point-situation courant. Finalement, une opération de normalisation permet de joindre les concepts avec des référents individuels similaires. En backtrackant sur les résultats de projection de Pns dans le point-situation courant (retour arrière sur CHOISIR dans l'algorithme 5), plusieurs nouveaux point-situations peuvent être construits; il peut ainsi y avoir indétermination.

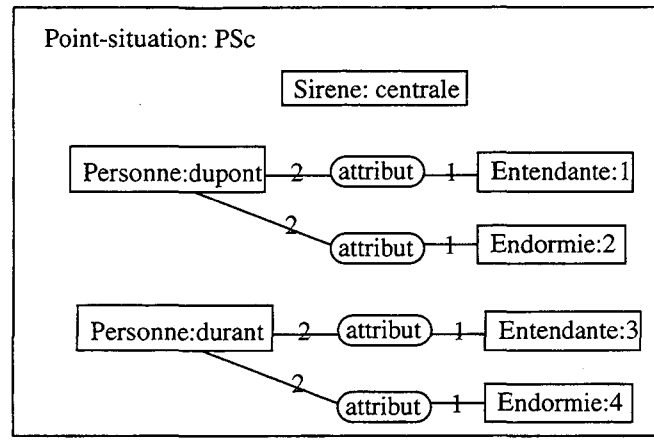
Dans le cas de l'emboîtement devient-faux, si la propriété P est vraie dans le point-situation courant, alors la transition doit être appliquée. Pour chaque rp (résultat de la projection de P dans PSc), chacun des sommets de rpPs (application de rp à la partie

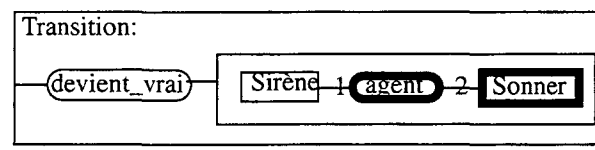
surbrillante Ps de P) est retiré du point-situation courant. Un seul nouveau point-situation peut être obtenu, il n'y a pas d'indétermination.

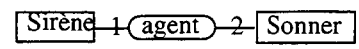
Algorithme 5: Appliquertransition(T,PSc)

```
Début
    T=(Te,P)
    P=(Pns,Ps)
     Sp ← projeter(P, PSc)
    Si Te=devient-vrai
        Si Sp ≠ ∅
            PSn ← PSc
        sinon
            Spns ← projeter(Pns, PSc)
            CHOISIR rpns ∈ Spns
            N ← numeroter(rpnsP) % numérotation des concepts génériques abstraits
            PSi ← ajouter(N, PSc) %somme de deux graphes disjoints
             PSn ← normaliser(PSi) %joints des concepts avec même référents individuels
        Fsi
    sinon si Te=devient-faux
        Si Sp = ∅
            PSn ← PSc
        sinon
            PSi ← PSc
            Pour tout rp ∈ Sp
                PSi ← retirer(PSi, rpPs) %oter un ensemble de sommets d'un graphe
            Fpour
            PSn ← PSi
        Fsi
    Fsi
    Retourner PSn
Fin
```

La figure 96 illustre comment l'algorithme 5 applique la transition de la figure 95 sur le point-situation de la figure 94.







projection de la propriété de la transition dans le point-situation courant

échec => la transition est appliquée

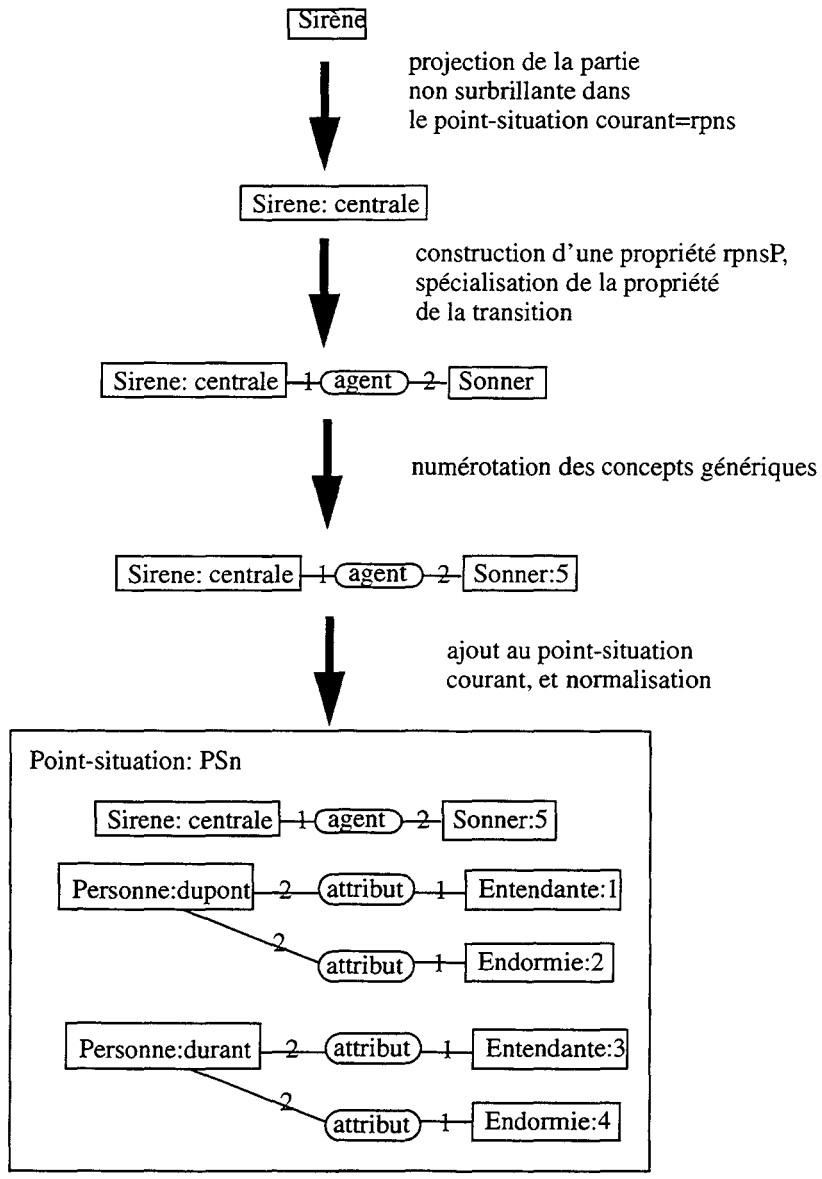


Figure 96. Application d'une transition sur un exemple

7.3.2. Application d'une loi dynamique

Nos choix de représentation sont les suivants:

- les point-situations sont représentés comme nous l'avons décrit dans 6.2.1, l'application d'une loi dynamique nécessite cependant de considérer deux points-situations PSc et PSn, PSn ayant été construit par l'application d'une transition sur PSc (par exemple, cf ceux des figure 94 et figure 96).
- une loi dynamique est une λ-abstraction n-aire décrite par un état, un événement et une transition (figure 91).

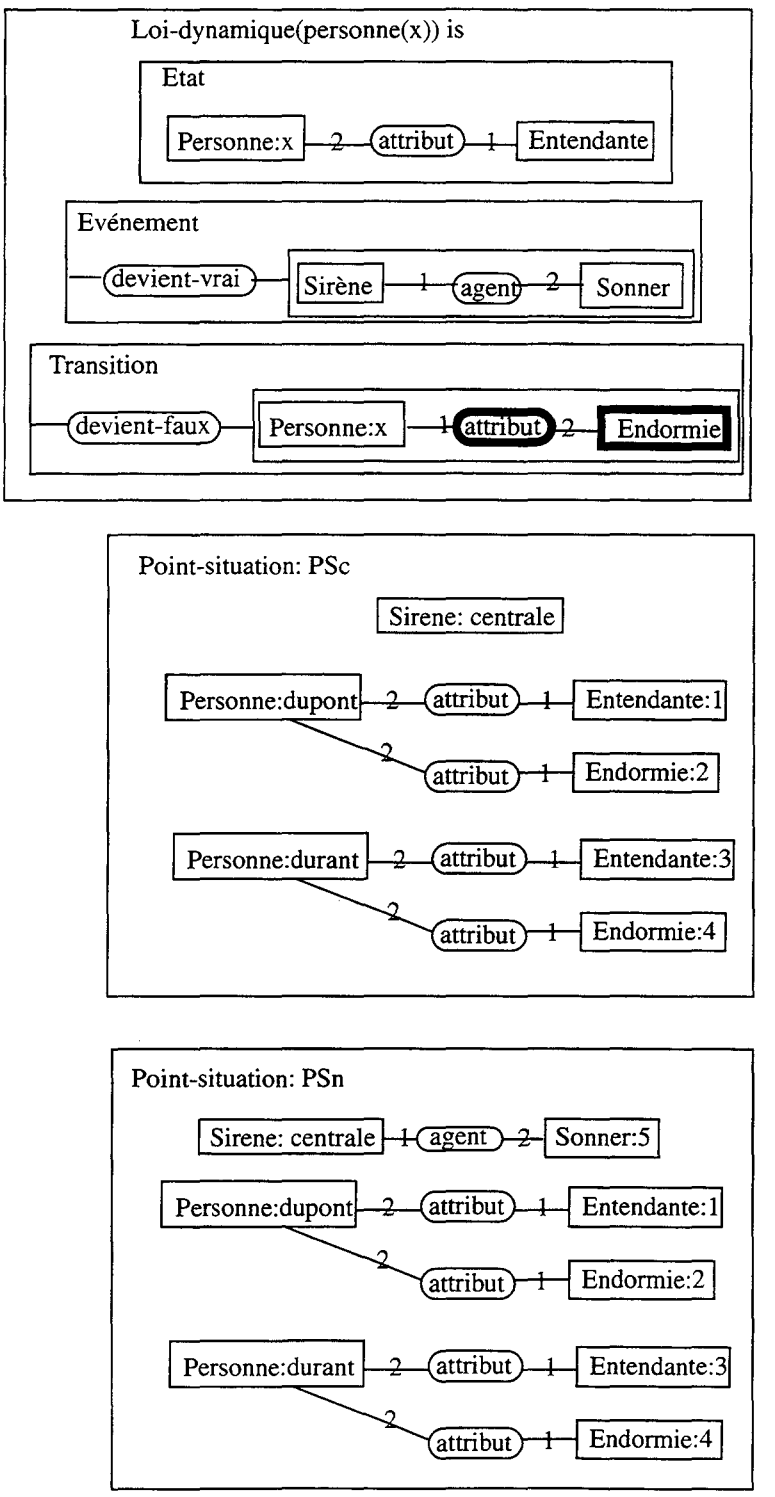
L'algorithme 6 permet d'appliquer une loi dynamique ***LD*** à partir de deux point-situations: le point-situation courant ***PSc*** et le nouveau point-situation ***PSn***, construit par l'application d'une transition sur PSc. La loi dynamique est décrite par un état ***Et***, un événement ***Ev*** et une transition ***T***.

Si l'état et l'événement peuvent être démontrés à partir de PSc et PSn, la transition est appliquée. Avant d'être démontré, l'événement est instancié avec une des applications, résultat de la démonstration de l'état E à partir de PSn. Et avant d'être appliquée, la transition est instanciée avec une composition de deux applications: l'une est un résultat de la démonstration de l'état E à partir de PSn, l'autre est un résultat de la démonstration de l'événement Ev à partir de PSc et PSn. Plusieurs transitions instanciées différemment sont appliquées itérativement sur le même point-situation.

Algorithme 6: Appliquerloi(LD,PSc,PSn)

```
Début
    LD=(Et,Ev,T)
    (V1, S1) ← demontrer(Et, PSc)
    (V2, S2) ← demontrer(Et, PSn)
    SEt ← S1 ∩ S2
    Si V1=vrai et V2=vrai
        S ← ∅
        Pour tout s ∈ SEt
            Ev=(Te,G)
            instancier(G, s)
            S1 ← projeter(G, PSc)
            S2 ← projeter(G, PSn)
            Si Te=fv
                Si S1 = ∅ et S2 ≠ ∅
                    S ← S + S2os
                Fsi
            sinon si Te=vf
                Si S1 ≠ ∅ et S2 = ∅
                    S ← S + S1os
                Fsi
            Fsi
        Fpour
        Si S ≠ ∅
            PSi ← PSn
            Pour tout s ∈ S
                T=(Te,G)
                instancier(G, s)
                S1 ← projeter(G, PSc)
                PSi ← appliquertransition(T, PSi)
            Fpour
            PSn' ← PSi
        sinon
            PSn' ← PSn
        Fsi
    sinon
        PSn' ← PSn
    Fsi
    Retourner PSn'
Fin
```

La figure 97 illustre comment l'algorithme 6 applique la loi dynamique de la figure 91 sur les point-situations des figure 94 et figure 96.



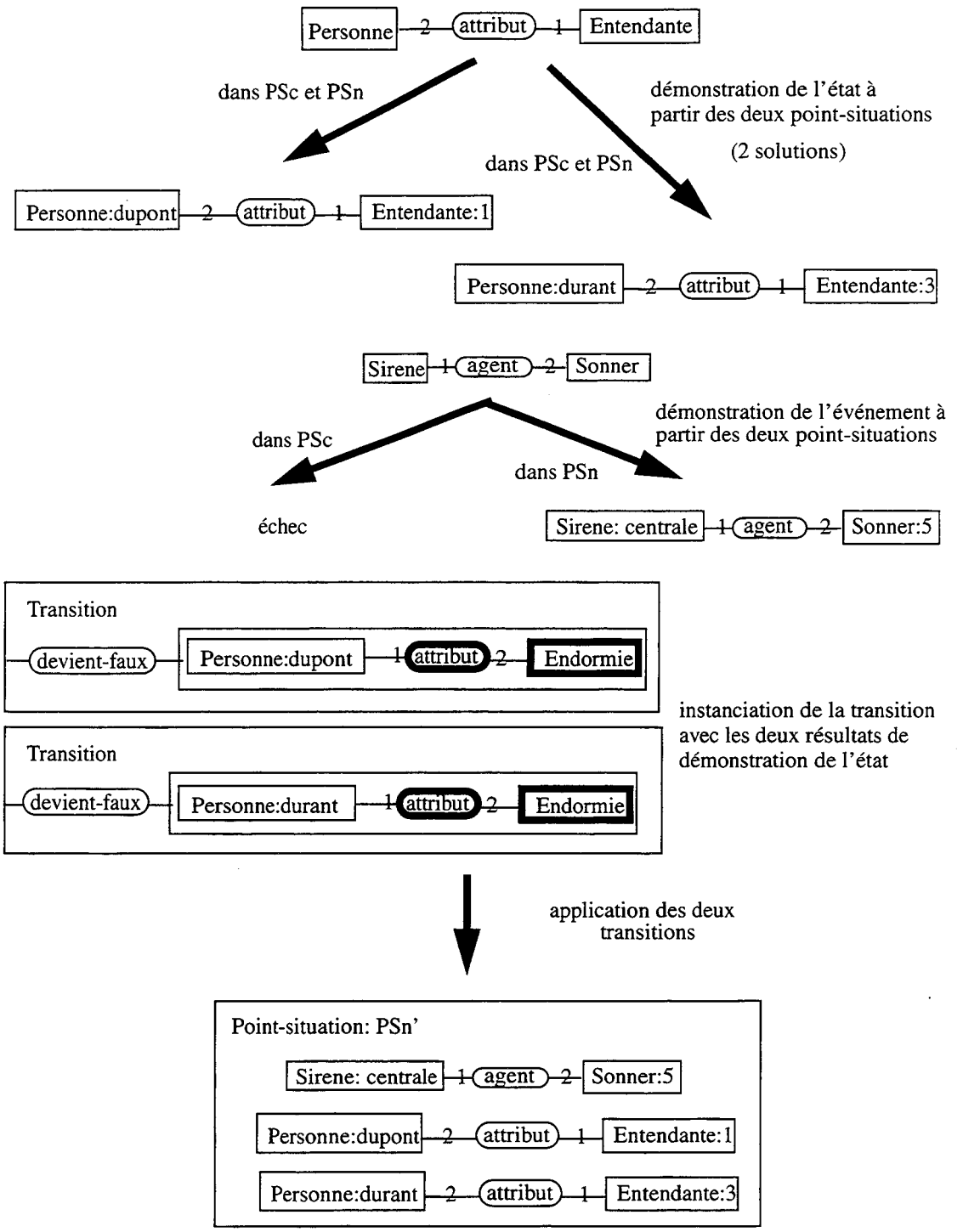


Figure 97. Application d'une loi dynamique sur un exemple

7.3.3. Mise à jour

La mise à jour consiste à construire un nouveau point-situation à partir du point-situation courant avec les effets de l'action exprimés par des transitions, puis à

prendre en compte les lois dynamiques afin de rétablir la consistance du nouveau point-situation vis-à-vis des lois du domaine si c'est nécessaire. L'algorithme 7 permet d'appliquer l'ensemble de toutes les lois dynamiques ***LLD*** à partir de deux point-situations: le point-situation courant ***PSc*** et le nouveau point-situation ***PSn***, construit par l'application d'une transition sur PSc. Les lois dynamiques sont appliquées jusque saturation, c'est-à-dire jusqu'à ce qu'il n'y en ait plus aucune d'applicable. Tous les ordres d'application des lois dynamiques donnant des résultats différents de simulation sont considérés.

Algorithme 7: Appliquertouteslois(LLD,PSc,PSn)

```
Début
    L ← LLD
    Tant que L ≠ ∅
        CHOISIR LD ∈ L
        PSn' ← appliquerloi(LD, PSc, PSn)
        Si PSn' ≠ PSn
            PSn ← PSn'
            L ← LLD
        sinon
            L ← L - LD
        Fsi
    Fintantque
Fin
```

Nous expliquons maintenant pourquoi nous considérons tous les ordres d'application des lois dynamiques. Si plusieurs lois sont applicables en même temps, il faut choisir un ordre d'application de ces lois. Deux lois dynamiques peuvent être applicables en même temps si:

- elles sont indépendantes (figure 98)
- elles proviennent d'une même loi statique et correspondent à deux rétablissements de cohérence possibles; elles sont alors concurrentes (figure 99)

Des lois indépendantes peuvent être appliquées dans n'importe quel ordre; les deux point-situations obtenus seront similaires. Le point-situation de la figure 100 est ainsi obtenu à partir des point-situations des figure 94 et figure 96 par l'application des lois de la figure 98 dans des ordres différents.

En revanche, suivant l'ordre dans lequel des lois concurrentes sont appliquées, les point-situations obtenus seront différents (ie, ils appartiendront à deux évolutions différentes du monde). L'application des lois dynamiques de la figure 99 sur les point-situations des figure 94 et figure 96 donne suivant l'ordre d'application un des point-situations de la figure 101.

Il faut ainsi essayer tous les ordres d'application des lois dynamiques afin de prendre en compte d'éventuelles lois concurrentes.

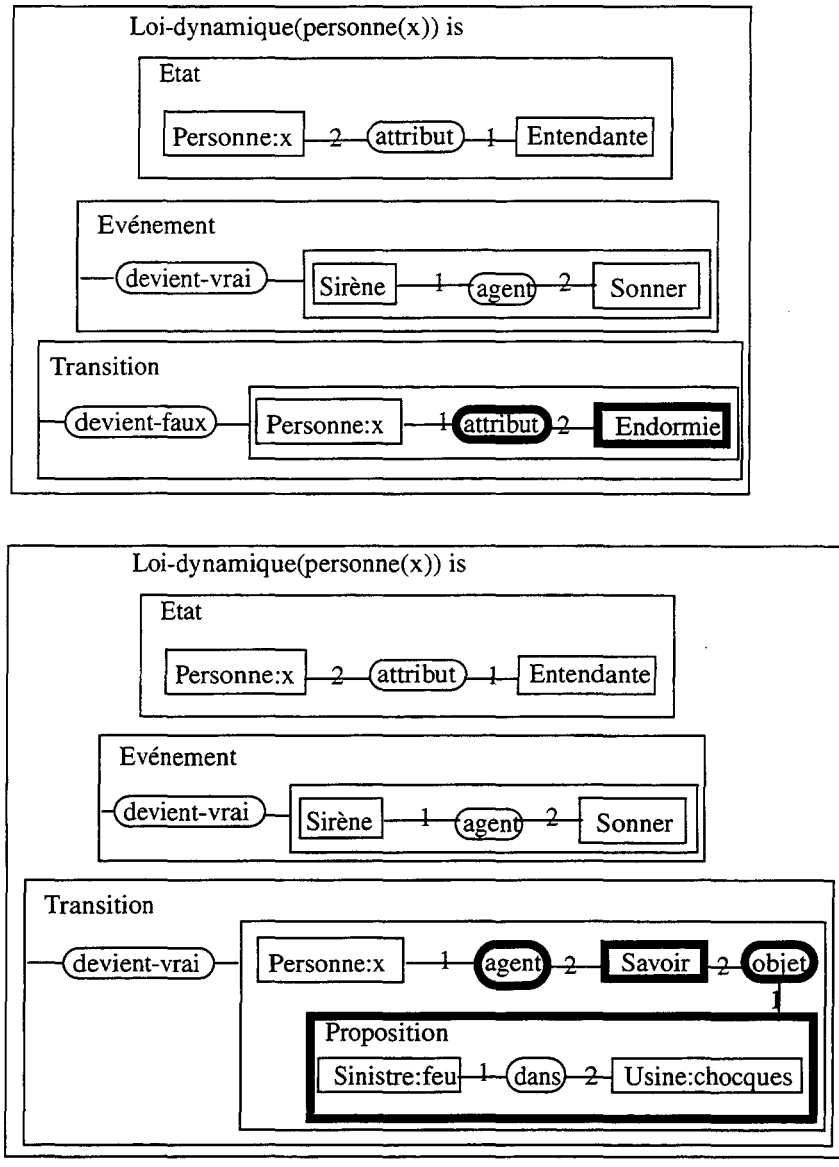


Figure 98. Lois dynamiques indépendantes

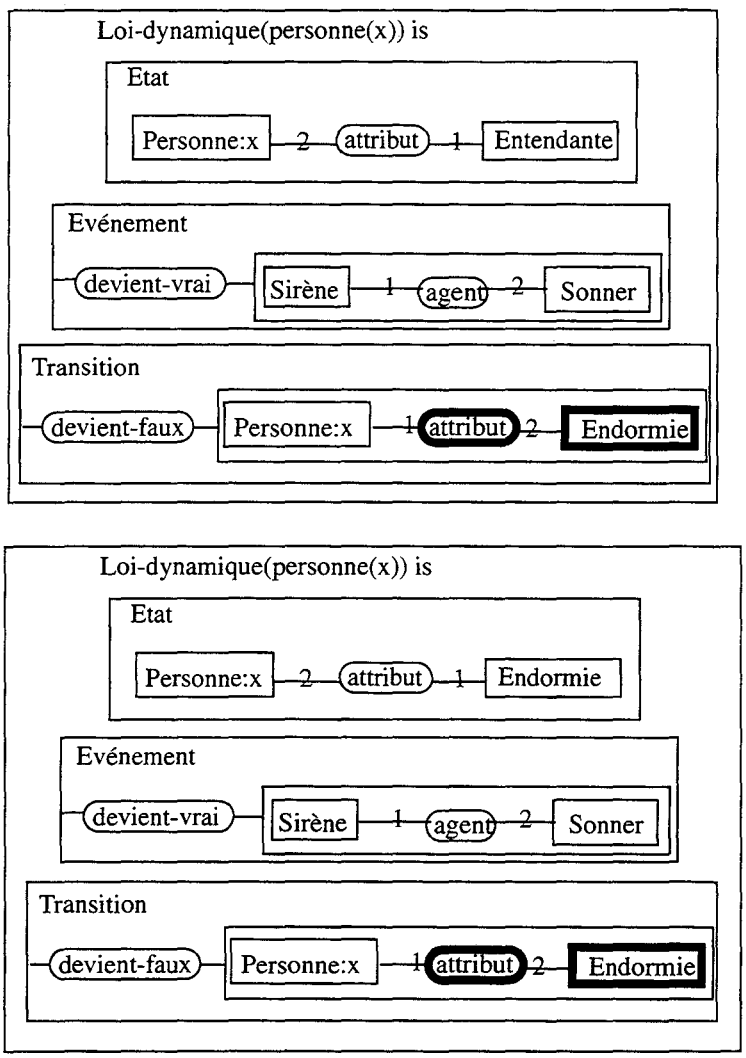


Figure 99. Lois dynamiques concurrentes

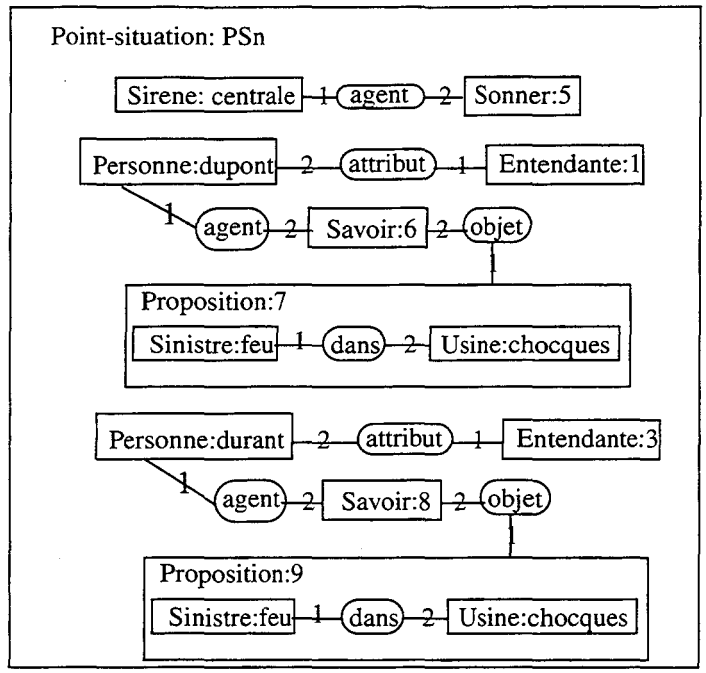


Figure 100. Un point-situation issu de l'application dans des ordres différents de 2 lois indépendantes

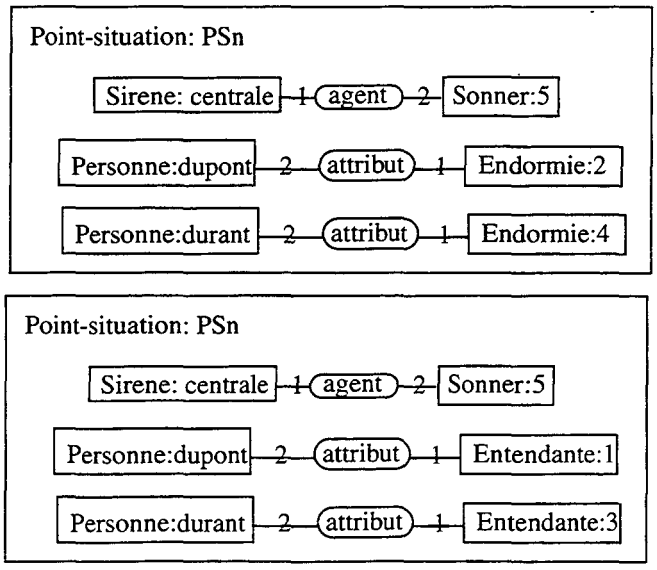


Figure 101. 2 point-situations issus de l'application dans des ordres différents de 2 lois concurrentes

7.4. Conclusion

L'algorithme de mise à jour présenté repose sur les hypothèses suivantes:

- monde clos dans les point-situations
- non prise en compte des lois statiques dans le principe de démonstration
- utilisation d'un modèle explicite d'évolution du monde, les lois dynamiques

Un principe de démonstration avec prise en compte des lois statiques permettrait d'alléger l'opération de mise à jour. Il est cependant à noter qu'un tel principe de démonstration ne dispenserait pas totalement de l'opération de mise à jour. Les lois dynamiques pourraient toujours être utiles pour rétablir la cohérence d'un nouveau point-situation vis-à-vis des lois du domaine. Imaginons que nous ne fassions pas l'hypothèse du monde clos, que le principe de démonstration prenne en compte les lois statiques, que la formule 60 soit une loi statique et que l'événement "a devient vrai" se produise:

- si le point-situation ne contient pas d'information sur la valeur de vérité de b, il n'est pas nécessaire d'appliquer de loi dynamique, b devient implicitement faux par le principe de démonstration prenant en compte les lois statiques
- en revanche, si le point-situation contient le littéral $\neg b$, une loi dynamique doit être appliquée afin de supprimer $\neg b$ du point-situation

$$a \rightarrow b \tag{60}$$

Un moyen d'éviter la mise à jour serait de considérer un principe de démonstration prenant en compte les lois statiques et de distinguer des faits de base et des faits déduits. Les effets des actions ne serait que des faits de base et le principe de démonstration permettrait aussi bien de déterminer la valeur de vérité de faits de base que de faits déduits. Il serait alors cependant nécessaire de modéliser des connaissances supplémentaires sur le monde: les distinctions faits de base et faits déduits. Bien que de telles connaissances puissent sembler moins naturelles à acquérir que des modèles explicites d'évolutions du monde, une telle voie mériterait d'être explorée dans l'avenir.

Partie III

Simulation: interprétation de l'ontologie comportementale

Cette partie est consacrée à l'interprétation de l'ontologie comportementale par l'algorithme de simulation. Elle présente la sémantique formelle de l'ontologie comportementale exprimée en une logique temporelle réifiée puis elle décrit l'algorithme général de simulation.

A partir de la modélisation (comportements, modèles d'actions et lois dynamiques) et d'un point-situation initial correspondant à un cas concret, l'algorithme de simulation doit être capable de générer toutes les évolutions possibles du monde. Pour ce faire, l'algorithme doit être capable d'exécuter des actions dans un certain ordre. Plusieurs évolutions du monde différentes correspondent à l'exécution d'ensembles d'actions différents ou à l'exécution du même ensemble d'actions dans des ordres différents. Afin de choisir quelle action exécuter à un instant clé, l'algorithme doit interpréter les relations temporelles (avant, après...), les relations condition (précondition, si...) et la relation de décomposition (déroulement). Il repose ainsi sur la sémantique formelle de ces relations. On peut dire que la sémantique formelle de l'ontologie comportementale impose un ensemble de contraintes qui doivent être vérifiées par chaque évolution possible du monde, résultat de la simulation.

Le premier chapitre de cette partie décrit la sémantique formelle de l'ontologie comportementale. Nous avons choisi d'exprimer celle-ci en une logique temporelle réifiée qui sépare les arguments des prédicats temporels en objets temporels (instant, moment, intervalle...) et en objets atemporels (comportement, action, état...). L'algorithme de simulation est ensuite détaillé et il est illustré par la simulation d'un extrait des fiches réflexes du plan particulier d'intervention d'une usine chimique.

Chapitre 8

Sémantique formelle de l'ontologie comportementale

La sémantique formelle, définie dans ce chapitre, donne la formalisation des contraintes posées sur chaque séquence résultat de simulation par les concepts et les relations de l'ontologie comportementale utilisés dans une modélisation. L'intérêt d'une telle formalisation est de définir clairement et précisément l'ontologie comportementale afin de favoriser non seulement sa compréhension et sa réutilisabilité, mais aussi son interprétabilité par l'algorithme de simulation.

Nous définissons tout d'abord une logique temporelle réifiée dont la caractéristique principale est d'autoriser une représentation granulaire du temps. Nous donnons alors, dans cette logique, les contraintes posées sur chaque séquence résultat de simulation.

8.1. La logique temporelle

En intelligence artificielle, les représentations logiques symboliques du temps sont classifiées en trois approches différentes:

- celles de logique classique dans lesquelles le temps n'a pas de statut particulier, il est juste un argument supplémentaire des prédicats décrivant les propriétés du monde ([MCC 69], [KOW 86])
- celles de logique modale qui considèrent le temps comme une modalité
- celles de logique temporelle réifiée qui donnent un statut particulier aux arguments temporels en les distinguant des arguments atemporels ([ALL84])

Nous avons choisi une logique réifiée car elle rend explicite le découpage entre les objets temporels et objets atemporels. Cette logique est du premier ordre et sa particularité est d'autoriser une représentation granulaire du temps.

Dans ce paragraphe, nous définissons tout d'abord la notion de séquence sur laquelle la logique temporelle va devoir permettre de poser des contraintes. Nous énumérons alors les objets temporels et les objets atemporels de la logique, ainsi que les prédicats qui donnent les propriétés entre ces objets. Finalement, nous décrivons une façon d'exprimer les coréférences (ou instanciations) dans la logique.

8.1.1. Définition d'une séquence

Dans la première partie de la thèse, nous avons émis l'hypothèse d'une représentation granulaire du temps par un niveau grossier correspondant à la phase d'élicitation des connaissances et par un niveau précis correspondant à une analyse plus fine des connaissances. L'égalité de deux instants au niveau grossier se traduit par une durée imperceptible entre ces deux instants au niveau précis, un instant étant défini par la prise en compte d'un effet d'une action. La non égalité de deux instants au niveau grossier se traduit par une durée significative entre ces deux instants au niveau précis.

L'algorithme de simulation génère les résultats de simulation au niveau précis. Un résultat de simulation est une ***séquence*** d'instants, à chacun desquels est associé un point-situation. Une séquence d'instants est un ensemble d'instants totalement ordonné par la relation avant ou juste avant (les relations avant et juste avant sont celles de la figure 39, nous les notons également $<_s$ et $<_i$, le s signifiant significatif et le i imperceptible, nous notons la disjonction "$<_s$ ou $<_i$" par $<_p$, le p signifiant précis). Deux instants consécutifs d'une séquence peuvent ainsi être séparés par une durée significative ou par une durée imperceptible. Notons que les instants d'une séquence sont des instants clés, c'est-à-dire qu'aucun changement du monde ne peut se produire entre deux instants consécutifs.

Les résultats de simulation peuvent être convertis au niveau grossier. Pour ce faire, une séquence S est divisée en sous-ensembles M_i d'instants totalement ordonnés par la relation $<_i$. Chacun de ces sous-ensembles M_i est appelé un moment. La famille des M_i constitue une partition de S. Chaque moment correspond à plusieurs prises en compte d'effets d'actions (ie, plusieurs instants).

S'il existe un instant I_a d'un moment M_i et un autre instant I_b d'un moment M_j tels que $I_a <_s I_b$, alors tout instant I_k de M_i et tout instant I_l de M_j sont tels que $I_k <_s I_l$. Nous notons cette propriété $M_i <_g M_j$, avec $<_g$ la relation avant du niveau grossier (cf figure 40). Les moments d'une séquence sont totalement ordonnés par la relation $<_g$.

Imaginons deux actions A et B telles que A d1-f1.d2-f2 B. A a un effet début, un effet fin et un effet intermédiaire et B a un effet début. La simulation des actions A et B génère une séquence composée de quatre instants: I_1 est l'effet début de A, I_2 l'effet intermédiaire de A, I_3 l'effet fin de A et I_4 l'effet début de B. Les relations temporelles entre ces instants sont $I_1 <_s I_2$, $I_2 <_s I_3$ et $I_3 <_i I_4$. La séquence est ainsi composée de trois moments $M_1=\{I_1\}$, $M_2=\{I_2\}$ et $M_3=\{I_3,I_4\}$.

8.1.2. Les objets

Nous distinguons les objets temporels qui sont les éléments d'une séquence des objets atemporels qui sont des concepts de l'ontologie comportementale. Les objets temporels considérés sont les suivants:

- L'instant est l'objet temporel basique, il correspond à la prise en compte d'un effet d'une action.
- Un intervalle d'instants est un ensemble d'instants totalement ordonné par la relation $<_p$, il est défini par un couple d'instants (I_{inf}, I_{sup}) avec I_{inf} la borne

inférieure de l'ensemble d'instants et I_{sup} sa borne supérieure.

- Un moment est un ensemble d'instants totalement ordonné par la relation $<_i$, c'est un cas particulier d'intervalle d'instants.
- Un intervalle de moments est un ensemble de moments totalement ordonné par la relation $<_g$, il est défini par un couple de moments (M_{inf}, M_{sup}) avec M_{inf} la borne inférieure de l'ensemble de moments et M_{sup} sa borne supérieure.

Les objets atemporels considérés sont des états, des événements, des actions, des comportements, des blocs conditionnels, des blocs optionnels, des blocs disjonctifs, des déroulements et des intervenants de comportements.

8.1.3. Les prédicats

Un premier ensemble de prédicats admet comme arguments uniquement des objets temporels. Ces prédicats temporels sont les suivants:

- les relations temporelles $<_s$, $<_i$, $<_p$, $<_g$, $\leq_s$, $\leq_i$, $\leq_p$ et $\leq_g$
- le prédicat $\in$ indiquant l'appartenance d'un instant à un intervalle d'instants ou à un moment, ou l'appartenance d'un moment à un intervalle de moments
- le prédicat $\subset$ indiquant l'inclusion d'un intervalle (instants ou moments) dans un autre intervalle

Dans ces prédicats, les fonctions inf et sup indiquent respectivement l'instant borne inférieure et l'instant borne supérieure d'un moment.

Un deuxième ensemble de prédicats admet comme arguments des objets temporels et des objets atemporels. Les deux prédicats primitifs de cet ensemble de prédicats sont ceux qui définissent l'établissement d'un état en un instant (***hold***) et l'occurrence d'un événement en un instant (***occur***). Ces prédicats sont primitifs dans le sens où, avec les prédicats temporels, ils permettent de définir tous les autres prédicats.

Soit un état E et un point situation PS associé à un instant I_i, le prédicat hold, signifiant que E est établi en I_i, est défini de la manière suivante (demontrer est la fonction permettant de donner la valeur de vérité d'un état à partir d'un point-situation cf algorithme 3):

$$\text{hold}(E, I_i) \leftrightarrow ((\text{vrai}, S) \leftarrow \text{demontrer}(E, PS)) \quad (61)$$

Soit un événement Ev ainsi que deux point-situations PS1, PS2 associés respectivement aux instants I_{i-1} et I_i, le prédicat occur, signifiant que Ev se produit en I_i, est défini de la manière suivante (demontrer est la fonction permettant de donner la valeur de vérité d'un événement à partir de deux point-situations cf algorithme 4):

$$\text{occur}(Ev, I_i) \leftrightarrow ((\text{vrai}, S) \leftarrow \text{demontrer}(Ev, PS1, PS2)) \quad (62)$$

Le prédicat hold primitif permet de définir un prédicat hold secondaire donnant l'établissement d'un état sur un intervalle d'instants, le deuxième argument de hold est alors un intervalle d'instants:

$$\forall I_i \forall I_j(\text{hold}(E, (I_i, I_j)) \leftrightarrow \forall I_k,(I_i \leq_p I_k \leq_p I_j)(\text{hold}(E, I_k))) \quad (63)$$

Et le prédicat occur primitif permet de définir un prédicat occur secondaire donnant l'occurrence d'un événement en un moment, le deuxième argument de occur est alors un moment:

$$\forall M_i(\text{occur}(Ev, M_i) \leftrightarrow \exists I_j,(I_j \in M_i)(\text{occur}(Ev, I_j))) \quad (64)$$

Le prédicat occurfirst définit la première occurrence d'un événement en un moment:

$$\forall M_i(\text{occurfirst}(Ev, M_i) \leftrightarrow \text{occur}(Ev, Mi) \wedge \forall M_j,(M_j <_g M_i)(\neg\text{occur}(Ev, M_j))) \quad (65)$$

Le prédicat occurfirstafter définit la première occurrence d'un événement en un moment après un autre moment:

$$\forall M_i \forall M_j(\text{occurfirstafter}(Ev, M_i, M_j) \leftrightarrow$$

$$\text{occur}(Ev, M_i) \wedge \forall M_k,(M_j <_g M_k <_g M_i)(\neg\text{occur}(Ev, M_k))) \quad (66)$$

Le dernier prédicat considéré est le prédicat execute qui définit l'exécution d'une tâche (action ou processus) ou l'exécution d'un déroulement (cf 8.2).

8.1.4. Coréférences

Nous introduisons la notion d'instanciation qui va permettre d'exprimer les coréférences dans la formalisation.

Une ***instanciation*** est une application d'un ensemble de concepts instanciables Eins dans un ensemble de concepts individuels Eind qui à tout couple (type,référent)=(t,x) d'un concept de Eins associe un couple (type,référent)=(t',a) d'un concept de Eind avec $t' \leq_c t$ et a un référent individuel. L'instanciation par σ d'un objet atemporel O est notée σO.

Des concepts instanciables sont des concepts auxquels une instanciation peut être appliquée.

Les concepts instanciables dans un état sont les concepts génériques de la partie positive du graphe emboîté dans l'état.

Les concepts instanciables dans un événement sont les concepts génériques du graphe emboîté dans l'événement.

Les concepts instanciables dans une action sont les concepts génériques du graphe emboîté dans l'action, ce sont donc les paramètres de l'action.

Les concepts instanciables dans un déroulement sont les concepts instanciables de toutes les actions décrivant le déroulement.

Enfin, les concepts instanciables dans un comportement sont les concepts instanciables des états, des événements et du déroulement liés au comportement.

Deux objets atemporels peuvent avoir un ensemble de concepts communs; ce sont les concepts instanciables d'un de ces objets coréférencés aux concepts instanciables de l'autre objet. Nous utilisons les instanciations pour exprimer que les concepts

communs de deux objets doivent être instanciés de façon similaire dans une séquence, résultat de simulation.

Par exemple, imaginons qu'une action A et qu'un état E, précondition de A, aient des concepts communs. L'exécution dans une séquence S de l'action A instanciée par σ (notée σA) nécessite que l'état E instancié par σ (noté σE) soit vrai juste avant l'action. L'exécution de A d'un instant I_i jusqu'un instant I_j nécessite que E soit vrai depuis la fin du moment précédant I_i jusque I_i se note:

$$\forall M_k,(I_i \in M_k)(\text{execute}(A, (I_i, I_j)) \rightarrow \text{hold}(E, (\sup(M_{k-1}), I_i)))) \quad (67)$$

Mais il est nécessaire d'ajouter dans la formalisation que les concepts communs de A et E doivent être instanciés de façon similaire:

$$\forall\sigma\forall M_k,(I_i \in M_k)(\text{execute}(\sigma A, (I_i, I_j)) \rightarrow \text{hold}(\sigma E, (\sup(M_{k-1}), I_i)))) \quad (68)$$

8.2. Les contraintes

Les concepts et les relations de l'ontologie comportementale utilisés dans une modélisation imposent des contraintes sur chaque séquence résultat de simulation. La figure 102 donne la notation d'une telle contrainte (F est une formule de la logique temporelle précédemment définie, dont les arguments sont des objets temporels et les objets atemporels A1 et A2).

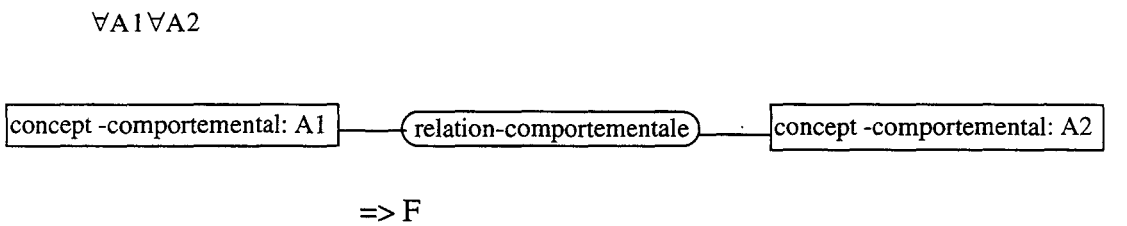


Figure 102. Syntaxe d'une contrainte

Dans la suite de ce chapitre, nous préférons cependant noter les contraintes linéairement de la façon suivante:

$$\forall A1\forall A2(\text{concept-comportemental}(A1) \wedge \text{concept-comportemental}(A2) \wedge \text{relation-comportementale}(A1, A2) \Rightarrow F) \quad (69)$$

8.2.1. Comportements

Conditions de contrôle

- Un comportement est exécuté seulement si une personne qui peut l'exécuter existe.

$$\forall C\forall P((\text{intervenant}(P, C) \wedge \text{comportement}(C) \wedge \text{personne}(P)) \Rightarrow$$

$$\forall\sigma\forall I_i\forall I_j\forall M_k,(I_i \in M_k)(\text{execute}(\sigma C, (I_i, I_j)) \rightarrow \text{hold}(\sigma P, \text{sup}(M_{k-1})))) \quad (70)$$

- Un comportement est exécuté seulement si ses conditions de contrôle sont vérifiées. Une condition de contrôle peut être exprimée par un état lié à une relation si ou par un événement lié à une relation sietlapremierefois ou sietchaquefois.

$$\forall C\forall E((\text{si}(C, E) \wedge \text{comportement}(C) \wedge \text{etat}(E)) \Rightarrow$$

$$\forall\sigma\forall I_i\forall I_j\forall M_k,(I_i \in M_k)(\text{execute}(\sigma C, (I_i, I_j)) \rightarrow \text{hold}(\sigma E, \text{sup}(M_{k-1})))) \quad (71)$$

$$\forall C\forall Ev((\text{sietchaquefois}(C, Ev) \wedge \text{comportement}(C) \wedge \text{evenement}(Ev)) \Rightarrow$$

$$\forall\sigma\forall I_i\forall I_j\forall M_k,(I_i \in M_k)(\text{execute}(\sigma C, (I_i, I_j)) \rightarrow \text{occur}(\sigma Ev, M_{k-1}))) \quad (72)$$

$$\forall C\forall Ev((\text{sietlapremierefois}(C, Ev) \wedge \text{comportement}(C) \wedge \text{evenement}(Ev)) \Rightarrow$$

$$\forall\sigma\forall I_i\forall I_j\forall M_k,(I_i \in M_k)(\text{execute}(\sigma C, (I_i, I_j)) \rightarrow \text{occurfirst}(\sigma Ev, M_{k-1}))) \quad (73)$$

- Un comportement est exécuté si une personne qui peut l'appliquer existe et si ses conditions de contrôle sont vérifiées. Dans le cas de la relation sietlapremierefois, s'il y a plusieurs instanciations possibles de la conjonction intervenant $\wedge$ événement $\wedge$ état, le comportement est exécuté avec une seule de ces instanciations. Le comportement ne peut ainsi être exécuté qu'une seule fois par un seul intervenant. Dans le cas de la relation sietchaquefois, s'il y a plusieurs instanciations possibles de la conjonction intervenant $\wedge$ événement $\wedge$ état, le comportement est exécuté avec chacune de ces instanciations. Le comportement peut ainsi être exécuté plusieurs fois par différents intervenants.

$$\forall C\forall Ev\forall E\forall P(\text{sietlapremierefois}(C, Ev) \wedge \text{si}(C, E) \wedge \text{intervenant}(P, C) \wedge$$

$$\text{comportement}(C) \wedge \text{evenement}(Ev) \wedge \text{etat}(E) \wedge \text{personne}(P) \Rightarrow$$

$$\forall M_i(\text{hold}(P, \text{sup}(M_i)) \wedge \text{hold}(E, \text{sup}(M_i)) \wedge \text{occurfirst}(Ev, M_i) \rightarrow$$

$$\exists I_j\exists!\sigma(\text{execute}(\sigma C, (\text{inf}M_{i+1}, I_j)) \wedge \text{hold}(\sigma P, \text{max}(M_i)) \wedge$$

$$\text{hold}(\sigma E, \text{max}(M_i)) \wedge \text{occurfirst}(\sigma Ev, M_i)))) \quad (74)$$

$$\forall C\forall Ev\forall E\forall P(\text{sietlapremierefois}(C, Ev) \wedge \text{si}(C, E) \wedge \text{intervenant}(P, C) \wedge$$

$$\text{comportement}(C) \wedge \text{evenement}(Ev) \wedge \text{etat}(E) \wedge \text{personne}(P) \Rightarrow$$

$$\forall\sigma\forall M_i(\text{hold}(\sigma P, \text{sup}(M_i)) \wedge \text{hold}(\sigma E, \text{sup}(M_i)) \wedge \text{occur}(\sigma Ev, M_i) \rightarrow$$

$$\exists I_j(\text{execute}(\sigma C, (\text{inf}M_{i+1}, I_j))))) \quad (75)$$

Préconditions

- Un comportement est exécuté seulement si ses préconditions sont vérifiées juste avant son exécution.

$$\forall C \forall E(\text{precondition}(E, C) \wedge \text{comportement}(C) \wedge \text{etat}(E) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j \forall M_k, (I_i \in M_k)(\text{execute}(\sigma C, (I_i, I_j)) \rightarrow \text{hold}(\sigma E, \sup(M_{k-1})))) \quad (76)$$

- S'il y a plusieurs instanciations possibles de la précondition, le comportement est exécuté avec une seule de ces instanciations.

$$\forall C \forall E(\text{precondition}(E, C) \wedge \text{comportement}(C) \wedge \text{etat}(E) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k, (I_i \in M_k)(\text{execute}(C, (I_i, I_j)) \rightarrow$$

$$\exists! \sigma(\text{hold}(\sigma E, \sup(M_{k-1})) \wedge \text{execute}(\sigma C, (I_i, I_j))))) \quad (77)$$

Déroulement d'un comportement

- L'exécution d'un comportement est équivalente à l'exécution de son déroulement.

$$\forall D \forall C(\text{deroulement}(D, C) \wedge \text{deroulement}(D) \wedge \text{comportement}(C) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j(\text{execute}(\sigma C, (I_i, I_j)) \leftrightarrow \text{execute}(\sigma D, (I_i, I_j)))) \quad (78)$$

8.2.2. Actions

Emboîtement des actions dans un déroulement

- Il est nécessaire d'exécuter toutes les actions d'un déroulement.

$$\forall A \forall D(\text{emboite}(A, D) \wedge \text{action}(A) \wedge \text{deroulement}(D) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j(\text{execute}(\sigma D, (I_i, I_j)) \rightarrow \exists I_k \exists I_l(\text{execute}(\sigma A, (I_k, I_l))))) \quad (79)$$

Contraintes temporelles sur l'exécution des actions

- Une action ne peut être exécutée que dans la plage temporelle délimitée par le déroulement emboîtant l'action et par les relations temporelles liées à l'action. Nous donnons uniquement les contraintes posées par les relations après (d2-f2-d1-f1) et juste après (d2-f2.d1-f1), les contraintes posées par les autres relations temporelles sont similaires.

$$\forall A \forall D(\text{emboite}(A, D) \wedge \text{action}(A) \wedge \text{deroulement}(D) \Rightarrow$$

$$\forall I_i \forall I_j \forall I_k \forall I_l(\text{execute}(D, (I_i, I_j)) \wedge \text{execute}(A, (I_k, I_l)) \rightarrow I_k \geq_p I_i \wedge I_l \leq_p I_j)) \quad (80)$$

$$\forall A \forall T (\text{d2-f2-d1-f1}(A, T) \wedge \text{action}(A) \wedge \text{tache}(T) \Rightarrow$$

$$\forall I_i \forall I_j \forall I_k \forall I_l (\text{execute}(T, (I_i, I_j)) \wedge \text{execute}(A, (I_k, I_l)) \rightarrow I_k >_s I_j)) \quad (81)$$

$$\forall A \forall T (\text{d2-f2.d1-f1}(A, T) \wedge \text{action}(A) \wedge \text{tache}(T) \Rightarrow$$

$$\forall I_i \forall I_j \forall I_k \forall I_l (\text{execute}(T, (I_i, I_j)) \wedge \text{execute}(A2, (I_k, I_l)) \rightarrow I_k >_i I_j)) \quad (82)$$

Synchronisation par un événement

- Le moment à partir duquel est attendu un événement qui synchronise un début d'action ou une fin d'action est déterminé par le déroulement emboîtant l'action et par les relations temporelles liées à l'action. Ce moment est associé à l'action par canddebmax dans le cas d'un début d'action synchronisé (dès-que) et par candfinmax dans le cas d'une fin d'action synchronisée (jusque). Nous rappelons qu'un début action (respectivement une fin d'action) synchronisé par un événement ne peut pas être lié à une relation temporelle qui positionne le début de l'action (respectivement la fin de l'action) juste après le début ou la fin d'une autre action.

$$\forall A \forall D (\text{emboite}(A, D) \wedge \text{action}(A) \wedge \text{deroulement}(D) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k, (I_i \in M_k)(\text{execute}(D, (I_i, I_j)) \rightarrow \text{canddeb}(A, M_k))) \quad (83)$$

$$\forall A \forall D (\text{emboite}(A, D) \wedge \text{action}(A) \wedge \text{deroulement}(D) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k, (I_i \in M_k)(\text{execute}(D, (I_i, I_j)) \rightarrow \text{candfin}(A, M_k))) \quad (84)$$

$$\forall T \forall A (\text{d2-f2-d1-f1}(A, T) \wedge \text{action}(A) \wedge \text{tache}(T) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k (\text{execute}(T, (I_i, I_j)) \wedge \text{canddeb}(A, M_k) \rightarrow \text{infM}_k >_s I_j)) \quad (85)$$

$$\forall A (\text{action}(A) \Rightarrow$$

$$\forall A (\text{action}(A) \Rightarrow \forall I_i \forall I_j \forall M_k, (I_i \in M_k)(\text{execute}(A, (I_i, I_j)) \rightarrow \text{candfin}(A, M_k))) \quad (86)$$

$$\forall D \forall A (\text{deroulement}(D, A) \wedge \text{action}(A) \wedge \text{deroulement}(D) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k, (I_{ij} \in M_k)(\text{execute}(D, (I_i, I_j)) \rightarrow \text{candfin}(A, M_k))) \quad (87)$$

$$\forall A (\text{action}(A) \Rightarrow$$

$$\forall M_i (\text{canddebmax}(A, M_i) \leftrightarrow \text{canddeb}(A, M_i) \wedge \forall M_k, (M_k >_g M_i)(\neg \text{canddeb}(A, M_k))) \quad (88)$$

$$\forall A(action(A) \Rightarrow$$

$$\forall M_i(candfinmax(A, M_i) \leftrightarrow candfin(A, M_i) \wedge \forall M_k,(M_k >_g M_i)(\neg candfin(A, M_k))) \quad (89)$$

- Une action dont le début est synchronisé par un événement est exécutée dès que cet événement se produit (plus précisément, au moment suivant). S'il y a plusieurs instanciations possibles de l'événement, l'action est exécutée avec une seule de ces instanciations et le déroulement dans lequel sont emboîtés l'action et l'événement est également exécuté avec une seule de ces instanciations.

$$\forall A \forall Ev(dès\text{-}que(A, Ev) \wedge action(A) \wedge evenement(Ev) \Rightarrow$$

$$\forall M_i \forall M_j(canddebmax(A, M_i) \wedge occurfirstafter(Ev, M_j, M_i) \rightarrow$$

$$\exists I_k \exists I_l \exists !\sigma,(I_k \in M_{j+1})(execute(\sigma A, (I_k, I_l)) \wedge occurfirstafter(\sigma Ev, M_j, M_i)))) \quad (90)$$

$$\forall A \forall Ev \forall D(dès\text{-}que(A, Ev) \wedge action(A) \wedge evenement(Ev) \wedge$$

$$emboite(Ev, D) \wedge deroulement(D) \Rightarrow$$

$$\forall M_i \forall M_j(canddebmax(A, M_i) \wedge occurfirstafter(Ev, M_j, M_i) \rightarrow$$

$$\exists I_k \exists I_l \exists !\sigma(execute(\sigma D, (I_k, I_l)) \wedge occurfirstafter(\sigma Ev, M_j, M_i)))) \quad (91)$$

- Une action dont la fin est synchronisée par un événement se termine dès que cet événement se produit (plus précisément, au moment suivant). S'il y a plusieurs instanciations possibles de l'événement, le déroulement dans lequel sont emboîtés l'action et l'événement est exécuté avec une seule de ces instanciations.

$$\forall A \forall Ev(jusque(A, Ev) \wedge action(A) \wedge evenement(Ev) \Rightarrow$$

$$\forall M_i \forall M_j(candfinmax(A, M_i) \wedge occurfirstafter(Ev, M_j, M_i) \rightarrow$$

$$\exists I_k \exists I_l,(I_l \in M_{j+1})execute(A, (I_k, I_l)))) \quad (92)$$

$$\forall A \forall Ev \forall D(jusque(A, Ev) \wedge action(A) \wedge evenement(Ev) \wedge$$

$$emboite(Ev, D) \wedge deroulement(D) \Rightarrow$$

$$\forall M_i \forall M_j(candfinmax(A, M_i) \wedge occurfirstafter(Ev, M_j, M_i) \rightarrow$$

$$\exists I_k \exists I_l \exists !\sigma(execute(\sigma D, (I_k, I_l)) \wedge occurfirstafter(\sigma Ev, M_j, M_i)))) \quad (93)$$

- Une action dont le début est synchronisé par un événement est exécutée seulement si cet événement se produit au moment avant l'action.

$$\forall A \forall Ev(dès\text{-}que(A, Ev) \wedge action(A) \wedge evenement(Ev) \Rightarrow$$

$$\text{occurfirstafter}(\sigma Ev, M_{l-1}, M_i))) \quad (94)$$

- Une action dont la fin est synchronisée par un événement se termine seulement si cet événement se produit au moment avant l'action.

$$\forall A \forall Ev(\text{jusque}(A, Ev) \wedge \text{action}(A) \wedge \text{evenement}(Ev) \Rightarrow$$

$$\forall \sigma \forall M_i \forall I_j \forall I_k \forall Ml, (I_j \in M_l)(\text{candfinmax}(A, M_i) \wedge \text{execute}(\sigma A, (I_j, I_k)) \rightarrow$$

$$\text{occurfirstafter}(\sigma Ev, M_{l-1}, M_i))) \quad (95)$$

Préconditions et conditions de bon déroulement

- Une action est exécutée seulement si ses préconditions sont vérifiées juste avant son exécution.

$$\forall A \forall E(\text{precondition}(E, A) \wedge \text{action}(A) \wedge \text{etat}(E) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j \forall M_k, (I_i \in M_k)(\text{execute}(\sigma A, (I_i, I_j)) \rightarrow \text{hold}(\sigma E, (\sup(M_{k-1}), I_i)))) \quad (96)$$

- S'il y a plusieurs instanciations possibles de la précondition, l'action est exécutée avec une seule de ces instanciations.

$$\forall A \forall E(\text{precondition}(E, A) \wedge \text{action}(A) \wedge \text{etat}(E) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k, (I_i \in M_k)(\text{execute}(A, (I_i, I_j)) \rightarrow$$

$$\exists ! \sigma(\text{hold}(\sigma E, (\sup(M_{k-1}), I_i)) \wedge \text{execute}(\sigma A, (I_i, I_j))))) \quad (97)$$

- Une action est exécutée correctement seulement si ses conditions de bon déroulement son vérifiées juste avant et durant toute son exécution.

$$\forall A \forall E(\text{conditionbonderoulement}(E, A) \wedge \text{action}(A) \wedge \text{etat}(E) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j \forall M_k, (I_i \in M_k)(\text{execute}(\sigma A, (I_i, I_j)) \rightarrow \text{hold}(\sigma A, (\sup M_{k-1}, I_j)))) \quad (98)$$

- S'il y a plusieurs instanciations possibles de la condition de bon déroulement, l'action est exécutée avec une seule de ces instanciations.

$$\forall A \forall E(\text{conditionbonderoulement}(E, A) \wedge \text{action}(A) \wedge \text{etat}(E) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k, (I_i \in M_k)(\text{execute}(A, (I_i, I_j)) \rightarrow$$

$$\exists ! \sigma(\text{hold}(\sigma E, (\sup M_{k-1}, I_j)) \wedge \text{execute}(\sigma A, (I_i, I_j))))) \quad (99)$$

Déroulement d'une action

- L'exécution d'une action est équivalente à l'exécution de son déroulement.

$$\forall D \forall A(\text{deroulement}(D, A) \wedge \text{action}(A) \wedge \text{deroulement}(D) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j(\text{execute}(\sigma A, (I_i, I_j)) \leftrightarrow \text{execute}(\sigma D, (I_i, I_j)))) \quad (100)$$

8.2.3. Blocs conditionnels

Emboîtement des blocs conditionnels dans un déroulement

- Il est nécessaire d'exécuter tous les blocs conditionnels d'un déroulement.

$$\forall BC \forall D(\text{emboite}(BC, D) \wedge \text{blocconditionnel}(BC) \wedge \text{deroulement}(D) \Rightarrow$$

$$\forall I_i \forall I_j(\text{execute}(\sigma D, (I_i, I_j)) \rightarrow \exists I_k \exists I_l(\text{execute}(\sigma BC, (I_k, I_l))))) \quad (101)$$

Contraintes temporelles sur l'exécution des blocs conditionnels

- Un bloc conditionnel ne peut être exécuté que dans la plage temporelle délimitée par le déroulement emboîtant le bloc et par les relations temporelles liées au bloc.

$$\forall BC \forall D(\text{emboite}(BC, D) \wedge \text{blocconditionnel}(BC) \wedge \text{deroulement}(D) \Rightarrow$$

$$\forall I_i \forall I_j \forall I_k \forall I_l(\text{execute}(D, (I_i, I_j)) \wedge \text{execute}(BC, (I_k, I_l)) \rightarrow I_k \geq_p I_i \wedge I_l \leq_p I_j)) \quad (102)$$

$$\forall BC \forall T(\text{d2-f2-d1-f1}(BC, T) \wedge \text{blocconditionnel}(BC) \wedge \text{tache}(T) \Rightarrow$$

$$\forall I_i \forall I_j \forall I_k \forall I_l(\text{execute}(T, (I_i, I_j)) \wedge \text{execute}(BC, (I_k, I_l)) \rightarrow I_k >_s I_j)) \quad (103)$$

$$\forall BC \forall T(\text{d2-f2.d1-f1}(BC, T) \wedge \text{blocconditionnel}(BC) \wedge \text{tache}(T) \Rightarrow$$

$$\forall I_i \forall I_j \forall I_k \forall I_l(\text{execute}(T, (I_i, I_j)) \wedge \text{execute}(BC, (I_k, I_l)) \rightarrow I_k >_i I_j)) \quad (104)$$

Condition exprimée par un événement

- Le moment à partir duquel est attendu un événement qui conditionne un bloc conditionnel est déterminé par le déroulement emboîtant le bloc et par les relations temporelles liées au bloc. Ce moment est associé au bloc conditionnel par le prédicat candmax.

$$\forall BC \forall D(\text{emboite}(BC, D) \wedge \text{blocconditionnel}(BC) \wedge \text{deroulement}(D) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k, (I_i \in M_k)(\text{execute}(D, (I_i, I_j)) \rightarrow \text{cand}(BC, M_k)))$$

$$\forall BC \forall T(\text{d2-f2-d1-f1}(BC, T) \wedge \text{blocconditionnel}(BC) \wedge \text{tache}(T) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k(\text{execute}(T, (I_i, I_j)) \wedge \text{cand}(BC, M_k) \rightarrow \text{inf}M_k >_s I_j)) \quad (105)$$

$$\forall BC(\text{blocconditionnel}(BC) \Rightarrow$$

$$\forall M_i(\text{candmax}(BC, M_i) \leftrightarrow \text{cand}(BC, M_i) \wedge \forall M_k,(M_k >_g M_i)(\neg\text{cand}(BC, M_k))) \quad (106)$$

- La partie alors (respectivement sinon) du bloc conditionnel est exécutée si l'événement conditionnant le bloc est vrai (respectivement faux) au moment précédant l'exécution du bloc. S'il y a plusieurs instanciations possibles de l'événement, la partie alors du bloc conditionnel sera exécutée avec une seule de ces instanciations et le déroulement dans lequel sont emboîtés le bloc conditionnel et l'événement sera également exécuté avec une seule de ces instanciations.

$$\forall BC \forall Ev(\text{si}(BC, Ev) \wedge \text{blocconditionnel}(BC) \wedge \text{evenement}(Ev) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k,(I_i \in M_k)(\text{execute}(BC, (I_i, I_j)) \wedge \text{occur}(Ev, M_{k-1}) \rightarrow$$

$$\exists!\sigma(\text{executealors}(\sigma BC, (I_i, I_j)) \wedge \text{occur}(\sigma Ev, M_{k-1})))) \quad (107)$$

$$\forall BC \forall Ev \forall D(\text{si}(BC, Ev) \wedge \text{blocconditionnel}(BC) \wedge \text{evenement}(Ev)$$

$$\wedge \text{emboite}(Ev, D) \wedge \text{deroulement}(D) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k,(I_i \in M_k)(\text{execute}(BC, (I_i, I_j)) \wedge \text{occur}(Ev, M_{k-1}) \rightarrow$$

$$\exists I_a \exists I_b \exists!\sigma(\text{execute}(\sigma D, (I_a, I_b)) \wedge \text{occur}(\sigma Ev, M_{k-1})))) \quad (108)$$

$$\forall BC \forall Ev(\text{si}(BC, Ev) \wedge \text{blocconditionnel}(BC) \wedge \text{evenement}(Ev) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k,(I_i \in M_k)(\text{execute}(BC, (I_i, I_j)) \wedge \neg\text{occur}(Ev, M_{k-1}) \rightarrow$$

$$\text{executesinon}(BC, (I_i, I_j)))) \quad (109)$$

- La partie alors (respectivement sinon) du bloc conditionnel est exécutée seulement si l'événement conditionnant le bloc est vrai (respectivement faux) au moment avant l'exécution du bloc.

$$\forall BC \forall Ev(\text{si}(BC, Ev) \wedge \text{blocconditionnel}(BC) \wedge \text{evenement}(Ev) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j \forall M_k,(I_i \in M_k)(\text{executealors}(\sigma BC, (I_i, I_j)) \rightarrow \text{occur}(\sigma Ev, M_{k-1}))) \quad (110)$$

$$\forall BC \forall Ev(\text{si}(BC, Ev) \wedge \text{blocconditionnel}(BC) \wedge \text{evenement}(Ev) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j \forall M_k,(I_i \in M_k)(\text{executesinon}(\sigma BC, (I_i, I_j)) \rightarrow \neg\text{occur}(\sigma Ev, M_{k-1}))) \quad (111)$$

- La partie alors d'un bloc conditionnel est exécutée dès que l'événement qui le

conditionne se produit. De façon plus précise, si l'événement se produit pour la première fois après le moment à partir duquel il est attendu et si la décision d'exécuter la partie sinon du bloc conditionnel n'a pas encore été prise, la partie alors du bloc conditionnel est exécutée au moment suivant l'occurrence de l'événement.

$$\forall BC \forall Ev(si(BC, Ev) \wedge blocconditionnel(BC) \wedge evenement(Ev) \Rightarrow$$

$$\forall M_i \forall M_j \forall I_k \forall I_l, (I_k <_p inf M_{j+1})(candmax(A, M_i) \wedge occurfirstafter(Ev, M_j, M_i) \wedge$$

$$\neg executesinon(BC, (I_k, I_l)) \rightarrow \exists I_a \exists I_b, (I_a \in M_{j+1})(executealors(BC, (I_a, I_b))))) \quad (112)$$

Condition exprimée par un état

- La partie alors (respectivement sinon) du bloc conditionnel est exécutée si l'état conditionnant le bloc est vrai (respectivement faux) juste avant l'exécution du bloc. S'il y a plusieurs instanciations possibles de l'état, le bloc conditionnel sera exécuté avec une seule de ces instanciations et le déroulement emboîtant le bloc conditionnel et l'état sera également exécuté avec une seule de ces instanciations.

$$\forall BC \forall E(si(BC, E) \wedge blocconditionnel(BC) \wedge etat(E) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k, (I_i \in M_k)(execute(BC, (I_i, I_j)) \wedge hold(E, (sup M_{k-1}, I_i)) \rightarrow$$

$$\exists ! \sigma(executealors(\sigma BC, (I_i, I_j)) \wedge hold(\sigma E, (sup M_{k-1}, I_i))))) \quad (113)$$

$$\forall BC \forall E \forall D(si(BC, E) \wedge blocconditionnel(BC) \wedge etat(E)$$

$$\wedge emboite(E, D) \wedge deroulement(D) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k, (I_i \in M_k)(execute(BC, (I_i, I_j)) \wedge hold(E, (sup M_{k-1}, I_i)) \rightarrow$$

$$\exists I_a \exists I_b \exists ! \sigma(execute(\sigma D, (I_a, I_b)) \wedge hold(\sigma E, (sup M_{k-1}, I_i))))) \quad (114)$$

$$\forall BC \forall E(si(BC, E) \wedge blocconditionnel(BC) \wedge etat(E) \Rightarrow$$

$$\forall I_i \forall I_j \forall M_k, (I_i \in M_k)(execute(BC, (I_i, I_j)) \wedge \neg hold(E, (sup M_{k-1}, I_i)) \rightarrow$$

$$executesinon(BC, (I_i, I_j)))) \quad (115)$$

- La partie alors (respectivement sinon) du bloc conditionnel est exécuté seulement si l'état conditionnant le bloc est vrai (respectivement faux) juste avant l'exécution du bloc.

$$\forall BC \forall E(si(BC, E) \wedge blocconditionnel(BC) \wedge etat(E) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j \forall M_k, (I_i \in M_k)(executealors(\sigma BC, (I_i, I_j)) \rightarrow hold(\sigma E, (sup M_{k-1}, I_i)))) \quad (116)$$

$$\forall BC \forall E(si(BC, E) \wedge blocconditionnel(BC) \wedge etat(E) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j \forall M_k, (I_i \in M_k)(executesinon(\sigma BC, (I_i, I_j)) \rightarrow \neg hold(\sigma E, (supM_{k-1}, I_i)))) \quad (117)$$

Déroulement d'un bloc conditionnel

- L'exécution de la partie alors du bloc conditionnel est équivalente à l'exécution du déroulement lié au bloc par la relation alors.

$$\forall BC \forall D(alors(BC, D) \wedge blocconditionnel(BC) \wedge deroulement(D) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j(executealors(\sigma BC, (I_i, I_j)) \leftrightarrow execute(\sigma D, (I_i, I_j)))) \quad (118)$$

- L'exécution de la partie sinon du bloc conditionnel est équivalente à l'exécution du déroulement lié au bloc par la relation sinon.

$$\forall BC \forall D(sinon(BC, D) \wedge blocconditionnel(BC) \wedge deroulement(D) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j(executesinon(\sigma BC, (I_i, I_j)) \leftrightarrow execute(\sigma D, (I_i, I_j)))) \quad (119)$$

- Si aucune relation sinon n'est liée au bloc conditionnel, l'exécution de la partie sinon du bloc est équivalente à l'exécution d'un déroulement vide.

$$\forall BC \forall D1(\neg sinon(BC, D1) \wedge blocconditionnel(BC) \wedge deroulement(D1) \Rightarrow$$

$$\forall I_i \forall I_j(executesinon(BC, (I_i, I_j)) \leftrightarrow$$

$$\exists D2(deroulement(D2) \wedge execute(D2, (I_i, I_j)) \wedge \forall A(\neg emboite(D2, A))))) \quad (120)$$

8.2.4. Blocs disjonctifs

Emboîtement des blocs disjonctifs dans un déroulement

- Il est nécessaire d'exécuter tous les blocs disjonctifs d'un déroulement.

$$\forall BD \forall D(emboite(BD, D) \wedge blocdisjonctif(BD) \wedge deroulement(D) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j(execute(\sigma D, (I_i, I_j)) \rightarrow \exists I_k \exists I_l(execute(\sigma BD, (I_k, I_l)))) \quad (121)$$

Contraintes temporelles sur l'exécution des blocs disjonctifs

- Un bloc disjonctif ne peut être exécuté que dans la plage temporelle délimitée par le déroulement emboîtant le bloc et par les relations temporelles liées au bloc.

$$\forall BD \forall D(emboite(BD, D) \wedge blocdisjonctif(BD) \wedge deroulement(D) \Rightarrow$$

$$\forall I_i \forall I_j \forall I_k \forall I_l(execute(D, (I_i, I_j)) \wedge execute(BD, (I_k, I_l)) \rightarrow I_k \geq_p I_i \wedge I_l \leq_p I_j)) \quad (122)$$

$$\forall BD \forall T(d2\text{-}f2\text{-}d1\text{-}f1(BD, T) \wedge blocdisjonctif(BD) \wedge tache(T) \Rightarrow$$

$$\forall I_i \forall I_j \forall I_k \forall I_l(execute(T, (I_i, I_j)) \wedge execute(BD, (I_k, I_l)) \rightarrow I_k >_s I_j)) \quad (123)$$

$$\forall BD \forall T(d2\text{-}f2.d1\text{-}f1(BD, T) \wedge blocdisjonctif(BD) \wedge tache(T) \Rightarrow$$

$$\forall I_i \forall I_j \forall I_k \forall I_l(execute(T, (I_i, I_j)) \wedge execute(BD, (I_k, I_l)) \rightarrow I_k >_i I_j)) \quad (124)$$

Déroulement d'un bloc disjonctif

- L'exécution d'un bloc disjonctif est équivalente à l'exécution de l'un des deux déroulements du bloc.

$$\forall BD \forall D1 \forall D2(deroulement(BD, D1) \wedge deroulement(BD, D2) \wedge$$

$$blocdisjonctif(BD) \wedge deroulement(D1) \wedge deroulement(D2) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j(execute(\sigma BD, (I_i, I_j)) \leftrightarrow (execute(\sigma D1, (I_i, I_j)) \vee execute(\sigma D2, (I_i, I_j))))) \quad (125)$$

8.2.5. Blocs optionnels

Emboîtement des blocs optionnels dans un déroulement

- Il est nécessaire d'exécuter tous les blocs optionnels d'un déroulement.

$$\forall BO \forall D(emboite(BO, D) \wedge blocoptionnel(BO) \wedge deroulement(D) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j(execute(\sigma D, (I_i, I_j)) \rightarrow \exists I_k \exists I_l(execute(\sigma BO, (I_k, I_l))) \quad (126)$$

Contraintes temporelles sur l'exécution des blocs optionnels

- Un bloc optionnel ne peut être exécuté que dans la plage temporelle délimitée par le déroulement emboîtant le bloc et par les relations temporelles liées au bloc.

$$\forall BO \forall D(emboite(BO, D) \wedge blocoptionnel(BO) \wedge deroulement(D) \Rightarrow$$

$$\forall I_i \forall I_j \forall I_k \forall I_l(execute(D, (I_i, I_j)) \wedge execute(BO, (I_k, I_l)) \rightarrow I_k \geq_p I_i \wedge I_l \leq_p I_j)) \quad (127)$$

$$\forall BO \forall T(d2\text{-}f2\text{-}d1\text{-}f1(BO, T) \wedge blocoptionnel(BO) \wedge tache(T) \Rightarrow$$

$$\forall I_i \forall I_j \forall I_k \forall I_l(execute(T, (I_i, I_j)) \wedge execute(BO, (I_k, I_l)) \rightarrow I_k >_s I_j)) \quad (128)$$

$$\forall BO \forall T(d2\text{-}f2.d1\text{-}f1(BO, T) \wedge blocoptionnel(BO) \wedge tache(T) \Rightarrow$$

$$\forall I_i \forall I_j \forall I_k \forall I_l(execute(T, (I_i, I_j)) \wedge execute(BO, (I_k, I_l)) \rightarrow I_k >_i I_j)) \quad (129)$$

Déroulement d'un bloc optionnel

- L'exécution d'un bloc disjonctif est équivalente à l'exécution du déroulement du bloc ou à l'exécution d'un déroulement vide.

$$\forall BD \forall D1(\text{deroulement}(BO, D1) \wedge \text{blocoptionnel}(BO) \wedge \text{deroulement}(D1) \Rightarrow$$

$$\forall \sigma \forall I_i \forall I_j(\text{execute}(\sigma BO, (I_i, I_j)) \leftrightarrow (\text{execute}((\sigma D1, (I_i, I_j)) \vee$$

$$\exists D2(\text{deroulement}(D2) \wedge \text{execute}(D2, (I_i, I_j)) \wedge \forall A(\neg \text{emboite}(D2, A))))))) \quad (130)$$

8.3. Conclusion

Chaque séquence résultat de simulation doit respecter les contraintes que nous venons de définir. L'algorithme de simulation doit en fait être capable de générer toutes les séquences respectant ces contraintes. Le chapitre suivant est consacré à la présentation d'un tel algorithme.

Chapitre 9

Algorithme de simulation

L'intérêt de la simulation est de permettre à l'expert et au cogniticien d'évaluer les comportements. Cette évaluation des comportements les amènera à corriger et enrichir les modèles et une nouvelle simulation pourra être lancée (figure 103). Premièrement, la simulation montre à l'expert les modèles "au travail" sur des cas concrets en générant toutes les séquences de point-situations. Deuxièmement, elle détecte automatiquement certains cas d'incohérences en contrôlant l'exécution. Enfin, elle permet à l'expert et au cogniticien de perturber l'exécution afin qu'ils puissent juger la robustesse des modèles. Le principe d'évaluation est dynamique, car basé sur une simulation, et coopératif, car l'expert peut interagir avec l'algorithme. L'expert peut demander une séquence résultat, il peut interroger cette séquence, il peut consulter le compte-rendu de contrôle, perturber ou stopper l'exécution. Nous préférons une approche coopérative d'évaluation plutôt qu'une approche automatique car seul l'expert est capable de juger si la connaissance contenue dans la modélisation est réellement sa connaissance.

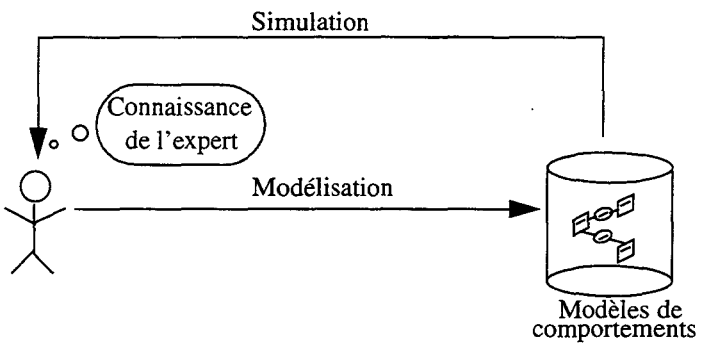


Figure 103. Evaluation des modèles par simulation

Il y a ainsi deux modes de fonctionnement de la simulation. Un premier mode de fonctionnement nécessite uniquement une connaissance supplémentaire par rapport à la modélisation: un scénario initial correspondant à un cas concret. Les résultats de simulation, qui révèlent des défauts ou des manques dans les modèles de comportement, sont alors:

- des séquences générées une à une et issues du scénario initial (plusieurs séquences différentes peuvent être générées à cause de l'indétermination dans les comportements).
- des comptes rendus d'exécution mettant en évidence des inconsistances détectées

automatiquement.

Un second mode interactif de fonctionnement de la simulation permet à l'expert de la perturber par l'introduction de connaissances anormales (une action est stoppée au début de ou pendant son exécution, un événement inattendu se produit...). De cette façon, l'expert peut juger la robustesse des comportements

Dans ce chapitre, nous présentons un algorithme permettant de générer une séquence résultat de simulation. La séquence peut être générée en une seule fois (afin d'obtenir directement un résultat observable) ou pas à pas (afin d'autoriser l'interaction avec l'expert). Les autres séquences peuvent ensuite être calculées en revenant en arrière sur les points de choix.

L'algorithme de simulation doit pouvoir:

- démontrer des états et des événements à partir de point-situations afin de prouver les conditions des tâches (cf partie II).
- mettre à jour un point-situation courant avec les effets d'une action (cf partie II).
- interpréter les relations de condition, les relations temporelles et la relation déroulement afin de savoir quand effectuer les démonstrations et mise à jour.

En parallèle de toutes ces tâches, l'algorithme gère l'indétermination. Si la connaissance contenue dans les modèles de comportement était complète, la simulation appliquée sur une description complète du monde ne générerait qu'une seule séquence. Cependant, les cas d'indéterminations sont nombreux. Chaque fois que l'algorithme rencontre un cas d'indétermination, il est susceptible de générer toutes les séquences correspondant à tous les choix possibles. Par exemple, si un pompier doit stopper le feu avant ou après avoir aidé les blessés, dans une séquence, il stoppera le feu avant d'aider les blessés, dans l'autre séquence, il sauvera les blessés avant de stopper le feu.

L'algorithme contrôle enfin l'exécution. Il détecte par exemple si des actions obligatoires n'ont pas pu être exécutées car leurs préconditions n'ont jamais pu être vérifiées ou car un événement ne s'est jamais produit.

9.1. Présentation de l'algorithme

Nous définissons tout d'abord les variables utilisées pour la présentation de l'algorithme. Nous décrivons alors l'algorithme de façon synthétique puis de façon détaillée. Nous précisons enfin quelques points délicats de l'algorithme tels que le choix des tâches à exécuter, l'instanciation des comportements, la démonstration, la mise à jour, la gestion des points de choix et le contrôle de l'exécution.

9.1.1. Définition des variables

Les objets manipulés sont des comportements ***C*** appartenant à la liste des comportements ***LC*** et des tâches instantanées ***T*** de types différents. Le type d'une tâche T est noté ***T.type***, c'est soit un début d'action, une fin d'action, un début de bloc conditionnel, une fin de bloc conditionnel, un début de bloc disjonctif, une fin de bloc

disjonctif, un début de bloc optionnel ou une fin de bloc optionnel. Le comportement d'une tâche T est noté ***T.comp*** (ie, la tâche T appartient au déroulement du comportement T.comp).

Les conditions et/ou synchronisations d'un comportement sont notées:

- ***C.si***: condition de contrôle du comportement exprimée par un état.
- ***C.sipremièrefois***: condition de contrôle du comportement exprimée par un événement qui se produit pour la première fois, le comportement n'est exécuté qu'une seule fois par un seul intervenant.
- ***C.sichaquefois***: condition de contrôle du comportement exprimé par un événement, le comportement peut être exécuté plusieurs fois par différents intervenants.
- ***C.prec***: précondition du comportement

Les conditions et/ou synchronisations d'une tâche sont notées:

- ***T.desque***: synchronisation d'un début d'action exprimée par un événement
- ***T.jusque***: synchronisation d'une fin d'action exprimée par un événement
- ***T.prec***: précondition d'une action
- ***T.cbd***: condition de bon déroulement d'une action (***LCBD*** est la liste des conditions de bon déroulement qui doivent être testées à chaque instant de l'exécution de l'action)
- ***T.sietat***: condition d'un bloc conditionnel exprimée par un état
- ***T.siévénement***: condition d'un bloc conditionnel exprimée par un événement

Les effets d'une tâche T (début ou fin d'action) sont notés ***T.effet***.

Le déroulement d'un comportement (respectivement d'une tâche) est noté ***C.deroulement*** (respectivement ***T.deroulement***). Un bloc conditionnel est lié à deux déroulements: ***T.deroulementalors*** et ***T.deroulementsinon***. Un bloc disjonctif est également lié à deux déroulements: ***T.déroulement1*** et ***T.deroulement2***. Les tâches du déroulement sont alors divisées en tâches initiales (sans dépendances temporelles, notées par exemple dans le cas du comportement ***C.deroulement.init***) et en tâches suivantes (avec des dépendances temporelles, notées par exemple dans le cas du comportement ***C.deroulement.suiv***).

A chaque tâche T, sont associées ses dépendances temporelles ***T.dep***. Une dépendance temporelle d'une tâche T est une autre tâche T' qui doit avoir été exécutée pour que T puisse être prise en compte dans la simulation. L'ensemble des dépendances d'une tâche est minimal, c'est-à-dire qu'il ne contient pas de dépendance qui se trouve dans l'ensemble des dépendances d'une de ses dépendances.

A chaque tâche T, sont également associées les tâches simultanées à T au niveau grossier de représentation temporelle (ie, les tâches qui doivent être exécutées juste avant ou juste après T au niveau précis de représentation). Ces tâches sont notées

T.egal.

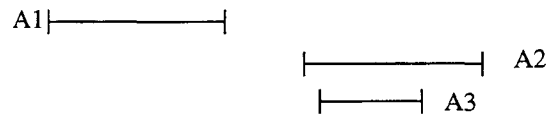


Figure 104. Six tâches ordonnées

Soient les trois actions de la figure 104 liées par des relations temporelles de la façon suivante:

- A2 d2-f2-d1-f1 A1
- A3 d2.d1-f1-f2 A2

Les six tâches équivalentes à ces trois actions sont alors ordonnées de la façon suivante:

- debA1.dep=∅ debA1.egal=∅
- finA1.dep=[debA1] finA1.egal=∅
- debA2.dep=[finA1] debA2.egal=[debA3]
- finA2.dep=[finA3] finA2.egal=∅
- debA3.dep=[debA2] debA3.egal=[debA2]
- finA3.dep=[debA3] finA3.egal=∅

Si des relations temporelles imprécises sont utilisées dans la modélisation, des disjonctions d'ordonnancement entre les tâches sont calculées. L'algorithme de simulation choisit un des ces ordonnancements afin de générer un premier ensemble de séquences, il revient ensuite en arrière et choisit d'autres ordonnancements afin de générer d'autres ensembles de séquences.

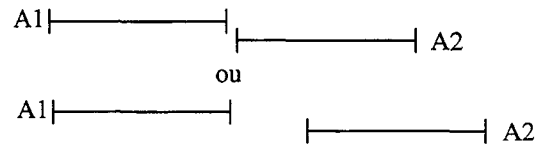


Figure 105. Relation temporelle imprécise

Soient les deux actions de la figure 105 liées par une relation temporelle imprécise:

- A2 d2-f2.d1-f1 ou d2-f2-d1-f1 A1

Les quatre tâches équivalentes à ces trois actions sont alors ordonnées de la

façon suivante:

- debA1.dep=∅ debA1.egal=∅
- finA1.dep=[debA1] finA1.egal=[debA2]
- debA2.dep=[finA1] debA2.egal=[finA1]
- finA2.dep=[debA2] finA2.egal=∅

ou de la façon suivante:

- debA1.dep=∅ debA1.egal=∅
- finA1.dep=[debA1] finA1.egal=∅
- debA2.dep=[finA1] debA2.egal=∅
- finA2.dep=[debA2] finA2.egal=∅

Les objets temporels manipulés par l'algorithme sont des instants du niveau précis de représentation temporelle. ***I*** est l'instant courant. Si $I \in M_k$, alors ***Iprec***=supM_{k-1} et ***Iprecprec***=supM_{k-2}. Le point-situation décrivant le monde à un instant I est noté ***I.PS***.

Les tâches sont rangées dans des listes différentes au fur et à mesure de la simulation:

- ***Tag*** est l'agenda des tâches qui devront être exécutées avant la fin de la simulation mais qui ont encore des dépendances temporelles.
- ***Tcand*** contient les tâches candidates, c'est à dire les tâches qui n'ont plus de dépendances temporelles.
- ***Tpot*** contient les tâches potentielles, c'est à dire les tâches qui peuvent être exécutées au moment courant (ce n'est cependant pas obligatoire).
- ***Tch*** contient les tâches choisies, c'est à dire les tâches qui doivent être exécutées au moment courant.
- ***Tegal*** contient les tâches simultanées aux tâches choisies, elles doivent aussi être exécutées au moment courant.

Les paramètres nécessaires pour démarrer une simulation sont les suivants:

- ***PS1*** et ***PS2*** sont les point-situations décrivant le monde aux deux instants initiaux.
- ***LC*** est la liste des comportements
- ***LLD*** est la liste des lois dynamiques
- ***STRAT*** est une stratégie de simulation (STRAT=tot, STRAT=tard ou STRAT=tout)

9.1.2. Algorithme synthétique

Déterminer quels comportements de LC sont candidats (conditions de contrôle vérifiées) et exécutables (préconditions vérifiées).

Ranger les tâches des déroulements des comportements candidats et exécutables dans les tâches candidates Tcand (tâches initiales du déroulement) et les tâches de l'agenda Tag (tâches suivantes du déroulement).

Trier les tâches candidates en tâches choisies Tch (l'événement qui les synchronise est vérifiée au moment précédant le moment courant; elles doivent être exécutées) et en tâches potentielles Tpot.

Choisir un sous ensemble des tâches potentielles Tpot pour les ajouter aux tâches choisies Tch.

Créer de nouveaux point-situations en mettant à jour les point-situations courants avec les effets des actions de Tch et Tegal.

Ranger les tâches des déroulements des tâches de Tch et Tegal dans les tâches candidates Tcand et les tâches de l'agenda Tag.

Ranger les tâches de l'agenda Tag qui n'ont plus de dépendances temporelles dans les tâches candidates Tcand.

9.1.3. Algorithme détaillé

Algorithme 8: simuler(PS1,PS2,LC,LLD,STRAT)
Début
Tag ← ∅
Tcand ← ∅
Tch ← ∅
LCBD ← ∅
Iprec ← 0
I ← 1
Iprec · PS ← PS1
I · PS ← PS2
%Rangement des tâches des déroulements des comportements candidats et exécutables dans Tcand et Tag
Itérer
Pour chaque C de LC
(V1, S1) ← demontreretat(C · si, I)
Si C · sipremierefois ≠ ∅
Si V1=vrai
(V2, S2) ← demontrerevt(C · sipremierefois, Iprec, I, S1)
Si V2=vrai
(V3, S3) ← demontreretat(C · prec, I, S2)
Si V3=vrai
Choisir s ∈ S3
LC ← LC – C
instancier(C,s)
Tcand ← Tcand + C · deroulement · init

```
                                    Tag ← Tag + C · deroulement · suiv
                                    Pour chaque T de C.deroulement.init
                                          ou de C.deroulement.suiv
                                            T · comp ← C
                                    Fpour
                              sinon
                                    PB1
                              Fsi
                        Fsi
                  sinon si C · sichaquefois ≠ ∅
                        Si V1=vrai
                              (V2, S2) ← demontrerevt(C · sichaquefois, Iprec, I, S1)
                              Si V2=vrai
                                    (V3, S3) ← demontreretat(C · prec, I, S2)
                                    Si V3=vrai
                                          Pour chaque s ∈ S3
                                                C1 ← C
                                                instancier(C1,s)
                                                Tcand ← Tcand + C1 · deroulement · init
                                                Tag ← Tag + C1 · deroulement · suiv
                                                Pour chaque T de C1.deroulement.init
                                                      ou de C1.deroulement.suiv
                                                        T · comp ← C1
                                                Fpour
                                          Fpour
                                    sinon
                                          PB2
                                    Fsi
                              Fsi
                        Fsi
                  Fsi
            Fpour
            Si Tcand = ∅
                  Si Tag ≠ ∅
                        PB3
                  sinon
                        fin simulation
                  Fsi
            Fsi
            Tpot ← ∅
%Tri des tâches candidates Tcand en tâches choisies Tch et tâches potentielles Tpot
            Pour chaque T de Tcand
                  Si T.type = debaction
                        Si T · desque ≠ ∅
                              (V1, S1) ← demontrerevt(T · desque, Iprec, I)
                              Si V1=vrai
                                    (V2, S2) ← demontreretat(T · prec, I, S1)
                                    (V3, S3) ← demontreretat(T · cbd, I, S2)
                                    Si V2=vrai et V3=vrai
                                          Choisir s ∈ S3
                                          instancier(T.comp,s)
                                          Tch ← Tch + T
                                          Tcand ← Tcand – T
                                    sinon
                                          PB4
```

```
                        Fsi
                    Fsi
                sinon
                    (V1, S1) ← demontreretat(T · prec, I)
                    (V2, S2) ← demontreretat(T · cbd, I, S1)
                    Si V1=vrai et V2=vrai
                        Choisir s ∈ S2
                        instancier(T.comp,s)
                        Tpot ← Tpot + T
                    Fsi
                Fsi
            sinon si T.type=debblocconditionnel
                Si T · sietat ≠ ∅
                    Tpot ← Tpot + T
                sinon si T · sievenement ≠ ∅
                    (V1, S1) ← demontrerevt(T · sievenement, Iprec, I)
                    Si V1=vrai
                        Choisir s ∈ S1
                        instancier(T.comp,s)
                        Tch ← Tch + T
                        Tcand ← Tcand – T
                    sinon
                        Tpot ← Tpot + T
                    Fsi
                Fsi
            sinon si T.type=finaction
                Si T · jusque ≠ ∅
                    (V1, S1) ← demontrerevt(T · jusque, Iprec, I)
                    Si V1=vrai
                        Choisir s ∈ S1
                        instancier(T.comp,s)
                        Tch ← Tch + T
                        Tcand ← Tcand – T
                    Fsi
                sinon
                    Tpot ← Tpot + T
                Fsi
            sinon si T.type=finblocconditionnel ou debblocdisjonctif
                ou finblocdisjonctif ou debblocoptionnel ou finblocoptionnel
                Tpot ← Tpot + T
            Fsi
        Fpour
        Si Tch = ∅ et Tpot = ∅
            Si Tcand ≠ ∅
                PB5
            sinon
                finsimulation
            Fsi
        Fsi
        choisir(Tpot,Tch,Tcand,STRAT)
        Tegal ← ∅
%Exécution des tâches choisies Tch et des tâches simultanées aux tâches choisies Tegal
        Pour chaque T de Tch
            Tegal ← Tegal + T · egal
        Fpour
        Iprec ← I
```

```
Iprecprec ← Iprec
Tant que Tch ≠ ∅
    Choisir T ∈ Tch
    Si T.type=debaction
        (V1, S1) ← demontreretat(T · prec, Iprec, I)
        (V2, S2) ← demontreretat(T · cbd, Iprec, I, S1)
        Si V1=vrai et V2=vrai
            Choisir s ∈ S2
            instancier(T.comp,s)
            majour(T.effet,LLD,I)
             LCBD ← LCBD + T · cbd
             Tcand ← Tcand + T · deroulement · init
             Tag ← Tag + T · deroulement · suiv
        sinon
            PB6
        Fsi
    sinon si T.type=finaction
        majour(T.effet,LLD,I)
          LCBD ← LCBD–T · cbd
    sinon si T.type=debblocconditionnel
        Si T · sietat ≠ ∅
            (V1, S1) ← demontreretat(T · sietat, Iprec, I)
            Si V1=vrai
                Choisir s ∈ S1
                instancier(T.comp,s)
                Tcand ← Tcand + T · deroulementalors · init
                Tag ← Tag + T · deroulementalors · suiv
            sinon
                Si T · deroulementsinon ≠ ∅
                    Tcand ← Tcand + T · deroulementsinon · init
                    Tag ← Tag + T · deroulementsinon · suiv
                Fsi
            Fsi
        sinon si T · sievenement ≠ ∅
            (V1, S1) ← demontrerevt(T · sievenement, Iprecprec, Iprec)
            Si V1=vrai
                Choisir s ∈ S1
                instancier(T.comp,s)
                Tcand ← Tcand + T · deroulementalors · init
                Tag ← Tag + T · deroulementalors · suiv
            sinon
                Si T · deroulementsinon ≠ ∅
                    Tcand ← Tcand + T · deroulementsinon · init
                    Tag ← Tag + T · deroulementsinon · suiv
                Fsi
            Fsi
        Fsi
    sinon si T.type=debblocdisjonctif
        Choisir deroulement1 ou deroulement2
        Si deroulement1
            Tcand ← Tcand + T · deroulement1 · init
            Tag ← Tag + T · deroulement1 · suiv
        sinon
            Tcand ← Tcand + T · deroulement2 · init
            Tag ← Tag + T · deroulement2 · suiv
        Fsi
```

```
                sinon si T.type=debblocoptionnel
                        Choisir deroulement ou non
                        Si deroulement
                                Tcand ← Tcand + T · deroulement1 · init
                                Tag ← Tag + T · deroulement1 · suiv
                        Fsi
                Fsi
                Pour chaque T' de Tag
                        Si T ∈ T' · dep
                                T' · dep ← T' · dep – T
                        Fsi
                        Si T' · dep = ∅
                                Si T' ∈ Tegal
                                        Tch ← Tch + T'
                                sinon
                                        Tcand ← Tcand + T'
                                Fsi
                                Tag ← Tag – T'
                        Fsi
                Fpour
                Tch ← Tch – T
                Pour chaque C ∈ LCBD
                        (V1, S1) ← demontreretat(C, I)
                        Si V1=faux
                                PB7
                        Fsi
                Fpour
                I ← I + 1
        Ftantque
    Finitérer
Fin
```

9.1.4. Choix des tâches dans Tpot

Plusieurs stratégies STRAT peuvent être adoptées pour choisir un sous-ensemble de Tpot qui sera ajouté aux tâches choisies de Tch.

Stratégie au plus tôt

Dès qu'une tâche peut être rangée dans Tpot, elle est transférée dans Tch.

Algorithme 9: choisir(Tpot,Tch,Tcand,tot)

```
Début
    Tch ← Tch + Tpot
    Tcand ← Tcand – Tpot
Fin
```

Stratégie au plus tard

Tch doit contenir le moins d'élément possible.

Algorithme 10: choisir(Tpot,Tch,Tcand,tard)
Début
Si Tch = ∅
E ← Choisir un élément de Tpot
Tch ← Tch + E
Tcand ← Tcand – E
Fsi
Fin

Stratégie donnant toutes les solutions

Les points de choix sont plus nombreux afin de pouvoir obtenir toutes les solutions.

Algorithme 11: choisir(Tpot,Tch,Tcand,tout)
Début
Tpot.cardinalité=n
Si Tch = ∅
E ← Choisir entre 1 et n éléments de Tpot
sinon
E ← Choisir entre 0 et n éléments de Tpot
Fsi
Tch ← Tch + E
Tcand ← Tcand – E
Fin

Les séquences obtenues par une stratégie au plus tôt ou au plus tard sont un sous-ensemble des séquences obtenues par la stratégie donnant toutes les solutions. Ces stratégies permettent de limiter la complexité si on veut générer plusieurs séquences résultats de simulation en une seule fois.

9.1.5. Instanciation d'un comportement

Les sommets d'un comportement sont les sommets des graphes emboîtés dans le déroulement et les états et événements liés au comportement. instancier(C,s) avec s une application résultat de la démonstration d'un état ou d'un événement dans un point-situation, signifie que s est appliquée aux sommets du comportement C. Nous parlons d'instanciation plutôt que de spécialisation car les sommets concepts génériques de C sur lesquels s est appliquée sont transformés en sommets concepts individuels.

9.1.6. Démonstration

Les principes de démonstration utilisés sont ceux des algorithmes 3 et 4 de la partie II. Les arguments des fonctions demontreetat et demontrerevt ne sont cependant pas des point-situations mais les instants auxquels ces point-situations sont associés. Un argument supplémentaire de ces fonctions peut également être une application s, résultat d'une démonstration, ceci, afin de pouvoir démontrer la spécialisation (instanciation) d'un état ou événement par s.

L'algorithme 12 démontre la valeur de vérité V d'un état E en un instant I. Si

E est vrai, S est l'ensemble des instanciations de E pour lesquelles E est vrai.

Algorithme 12: demontreretat(E,I)
Début
(V, S) ← demontrer(E, I · PS)
Retourner (V,S)
Fin

L'algorithme 13 démontre la valeur de vérité V d'un état E entre deux instants I1 et I2 (E doit être vrai à chaque instant entre I1 et I2). Si E est vrai, S est l'ensemble des instanciations de E pour lesquelles E est vrai.

Algorithme 13: demontreretat(E,I1,I2)
Début
(V, S) ← demontreretat(E, I1)
Si V=faux
Retourner(V,S)
sinon
I ← I1 + 1
Si S = ∅
Tant que, $I \leq_p I2$ et V ≠ faux
(V, S1) ← demontrer(E, I · PS)
I ← I + 1
Ftantque
Retourner(V,S)
sinon
Tant que, $I \leq_p I2$ et S ≠ ∅
Pour chaque s ∈ S
(V, S1) ← demontrer(sE, I · PS)
Si V=faux
S ← S − s
Fsi
Fpour
I ← I + 1
Ftantque
Si S = ∅
Retourner (faux,S)
sinon
Retourner(vrai,S)
Fsi
Fsi
Fsi
Fin

L'algorithme 14 démontre la valeur de vérité V d'un événement Ev entre deux instants I1 et I2 (Il suffit que Ev se produise en un instant entre I1 et I2). Si Ev est vrai, S est l'ensemble des instanciations de Ev pour lesquelles Ev est vrai.

Algorithme 14: demontrerevt(Ev,I1,I2)
Début
S ← ∅
V ← faux
Pour chaque I, $I1 \leq_p I <_p I2$
(V1, S1) ← demontrer(Ev, I · PS, I + 1 · PS)

```
            Si V1=vrai
                    V ← vrai
            Fsi
             S ← S + S1
        Fpour
        Retourner(V,S)
Fin
```

L'algorithme 15 démontre la valeur de vérité V d'un état E instancié par une application de S en un instant I. Si E est vrai, chaque élément de S1 est la composition d'une application de S et d'une instanciation de E pour laquelle E est vrai.

Algorithme 15: <u>demontreretat(E,I,S)</u>

```
Début
        S1 ← ∅
        V ← faux
        Pour chaque s ∈ S
                (V2, S2) ← demontrer(sE, I · PS)
                Si V2=vrai
                        V ← vrai
                        Pour chaque s2 ∈ S2
                                S1 ← S1 + s2os  % o représente la composition de deux applications
                        Fpour
                Fsi
        Fpour
        Retourner(V,S1)
Fin
```

L'algorithme 16 démontre la valeur de vérité V d'un état E instancié par une application de S entre deux instants I1 et I2. Si E est vrai, chaque élément de S1 est la composition d'une application de S et d'une instanciation de E pour laquelle E est vrai.

Algorithme 16: <u>demontreretat(E,I1,I2,S)</u>

```
Début
        (V, S1) ← demontreretat(E, I1, S)
        Si V=faux
                Retourner(V,S1)
        sinon
                I ← I1 + 1
                Si S1 = ∅
                        Tant que, I ≤p I2 et V ≠ faux
                                (V, S2) ← demontrer(E, I · PS)
                                I ← I + 1
                        Ftantque
                        Retourner(V,S1)
                sinon
                        Tant que I ≤p I2 et  S1 ≠ ∅
                                Pour chaque s ∈ S1
                                        (V, S2) ← demontrer(sE, I · PS)
                                        Si V=faux
                                                S1 ← S1 – s
                                        Fsi
                                Fpour
                                I ← I + 1
```

```
                Ftantque
                Si S1 = ∅
                        Retourner (faux,S1)
                sinon
                        Retourner(vrai,S1)
                Fsi
        Fsi
Fsi
Fin
```

L'algorithme 17 démontre la valeur de vérité V d'un événement Ev instancié par une application de S entre deux instants I1 et I2. Si Ev est vrai, chaque élément de S1 est la composition d'une application de S et d'une instanciation de Ev pour laquelle Ev est vrai.

Algorithme 17: demontrerevt(Ev,I1,I2,S)

```
Début
    S1 ← ∅
    V ← faux
    Pour chaque s ∈ S
        Pour chaque I, I1 ≤ pI < pI2
            (V2, S2) ← demontrer(sEv, I · PS, I + 1 · PS)
            Si V2=vrai
                V ← vrai
                Pour chaque s2 ∈ S2
                    S1 ← S1 + s2os
                Fpour
            Fsi
        Fpour
    Fpour
    Retourner(V,S1)
Fin
```

9.1.7. Mise à jour

Le principe de mise à jour repose sur l'algorithme 5 d'application d'une transition et sur l'algorithme 7 d'application des lois dynamiques.

Algorithme 18: majour(T,LLD,I)

```
Début
    PSn ← appliquertransition(T, I · PS)
    PSn′ ← appliquertouteslois(LLD, I · PS, PSn)
    I + 1 · PS ← PSn′
Fin
```

9.1.8. Points de choix

Seul le retour arrière sur tous les points de choix assure la complétude de la simulation. Les points de choix sur lesquels l'algorithme revient en arrière sont dûs:

- aux relations temporelles imprécises; l'algorithme choisit un des ordonnancements entre les tâches.

- aux paramètres indéterminés dans les conditions et/ou synchronisations exprimés par des états et/ou des événements; l'algorithme choisit une instanciation de l'état ou de l'événement.
- aux blocs disjonctifs; l'algorithme choisit un des deux déroulements du bloc.
- aux blocs optionnels; l'algorithme choisit ou non le déroulement du bloc.

9.1.9. Contrôle

Une séquence générée n'est correcte (ie, elle respecte les contraintes formelles posées au chapitre 8) que si aucun problème n'est détecté au cours de la simulation. Les problèmes qui peuvent être détectés au cours de la simulation sont les suivants:

- PB1 et PB2; le comportement doit être exécuté à un moment donné car il est synchronisé par un événement, et ses préconditions ne sont pas vérifiées (les formules 74 et 76 ou 75 et 76 ne peuvent pas être respectées)
- PB3; des tâches n'ont pas pu être libérées de leurs dépendances temporelles (l'ensemble des contraintes temporelles, exprimées par des formules telles que les formules 80, 81, 82, n'est pas cohérent)
- PB4; une action doit être exécutée à un moment donné car elle est synchronisée par un événement, et ses préconditions ne sont pas vérifiées (les formules 90 et 96 ne peuvent pas être respectées)
- PB5; des actions n'ont pas pu être exécutées car leurs préconditions n'ont jamais été vérifiées ou car l'événement qui les déclenche ne s'est jamais produit (la formule 79 rendait ces actions obligatoires et les formules 90 ou 96 n'ont jamais pu être vérifiées)
- PB6; deux actions sont considérées pouvoir débuter en même temps ou doivent débuter en même temps alors que les effets d'une des actions contredisent les préconditions de l'autre
- PB7; les conditions de bon déroulement d'une action ne sont pas vérifiées pendant l'exécution de l'action (la formule 98 ne peut pas être respectée)

9.2. Implémentation

Nous avons implémenté l'algorithme général de simulation proposé dans la partie III de la thèse, ainsi que les procédures de démonstration et de mise à jour proposées dans la partie II. Les algorithmes implémentés sont ainsi ceux numérotés de 2 à 18. La simulation peut être exécutée en mode pas-à-pas ou de façon continue. Il est également possible de revenir en arrière sur les points de choix afin de générer toutes les solutions possibles.

Une interface graphique permet de saisir les paramètres d'entrée de la simulation (ie, une modélisation) et d'en afficher les résultats (ie, des séquences de point-situations).

Les paramètres d'entrée de la simulation sont:

- une ontologie du domaine composée d'une hiérarchie de types de concepts et d'une hiérarchie de types de relations.
- des modèles génériques d'actions.
- des lois dynamiques.
- des comportements.
- une stratégie de simulation (tôt, tard ou tout).
- un scénario initial composé de deux point-situations (deux point-situations sont nécessaires afin de pouvoir modéliser un événement).

L'ontologie comportementale n'est pas considérée comme un paramètre d'entrée de la simulation. Elle est en effet figée; la simulation repose sur sa sémantique.

Les métaconcepts (signature, acceptions, conditions....) ne sont pas non plus des paramètres d'entrée de la simulation. Leur intérêt est de guider et contrôler la saisie des graphes.

L'annexe E contient des copies d'écran de l'interface graphique. Ces copies illustrent un exemple de modélisation donné en entrée de la simulation et les résultats de simulation correspondants. Le domaine considéré est celui des plans particuliers d'intervention de la sécurité civile. L'exemple de modélisation s'inspire de la fiche réflexe du témoin d'un sinistre dans une usine chimique ([SCI 90]).

De façon plus précise, l'annexe E contient des copies d'écran:

- de l'ontologie comportementale (annexe E.1.).
- d'une ontologie du domaine (annexe E.2.).
- des concepts de simulation - les instances de ces concepts sont des modèles d'actions, des lois dynamiques, des comportements ou des point-situations- (annexe E.3).
- de l'ensemble des signatures de la relation précondition (annexe E.4.).
- de trois modèles génériques d'actions -dans ces modèles, des surbrillances permettent d'exprimer des transitions précises- (annexe E.5.).
- d'une loi dynamique (annexe E.6.).
- d'un comportement -dans ce comportement, un emboîtement gris foncé représente une négation, les tâches de ce comportement sont un bloc-optionnel, deux blocs-conditionnels et trois actions- (annexe E.7.).
- d'un scénario initial -composé de deux point-situations permettant d'exprimer l'événement "dupont entre dans l'entrepot1 et sait maintenant qu'il existe un sinistre grave"- (annexe E.8.).
- des résultats de simulation (annexe E.9.).

Les résultats de simulation sont constitués de six point-situations. Chaque point-situation correspond à un début ou à une fin d'action. L'exécution d'un début ou d'une fin d'action peut provoquer un changement du monde positif ou un changement du monde négatif. Par exemple, entre ps2 et ps3 dupont se met à vouloir le déclenchement d'une alerte et entre ps3 et ps4 dupont quitte l'entrepot 1.

Chapitre 10

Conclusion

Nous concluons cette thèse en résumant nos contributions, en décrivant des travaux futurs qu'il serait intéressant de réaliser et en évoquant des perspectives applicatives relatives à l'implémentation d'un système d'assistance à la modélisation de comportements dans une organisation humaine.

10.1.Contributions

Les résultats de l'acquisition des connaissances comportementales dans une organisation humaine sont généralement incomplets, pauvres et peu robustes car les connaissances à acquérir sont complexes (environnement complexe, tâches de natures diverses, relations temporelles variées, indéterminations...), mal établies dans la tête de l'expert du domaine et difficiles à généraliser.

Pour remédier à ces problèmes, nos objectifs ont été de guider l'expert dans la description des comportements par une approche descendante de modélisation et surtout de le confronter le plus tôt possible aux résultats de modélisation grâce à une simulation symbolique des comportements. La simulation montre à l'expert les modèles de comportement "au travail" sur des cas concrets afin de lui faire prendre conscience des limitations, des performances et de la robustesse de la modélisation. Le cogniticien et l'expert peuvent alors revenir sur la modélisation en corrigeant et en enrichissant les modèles.

Nos contributions sont un langage de modélisation de comportements et un principe de simulation reposant sur ce langage. Le langage de modélisation doit pouvoir être interprété par l'algorithme de simulation: il est ainsi nécessaire de ***modéliser pour simuler***. Et la simulation aide l'expert à modéliser les comportements: il est ainsi avantageux de ***simuler pour modéliser***.

En ce qui concerne le langage de modélisation:

- nous avons proposé une ontologie comportementale adaptée à la modélisation de comportements humains et nous avons défini formellement sa sémantique.
- nous avons défini et utilisé des extensions du formalisme de base des graphes conceptuels (emboîtements typés ou non, métaconcepts, négation sous forme de monde clos ou de contexte ovale, différences de graphes notées par des surbrillances) afin de représenter l'ontologie, les comportements et les descriptions

du monde.

En ce qui concerne le principe de simulation:

- nous avons proposé un algorithme général de simulation qui interprète les types de l'ontologie comportementale.
- nous avons détaillé un principe de démonstration d'un état ou d'un événement à partir d'un point-situation basé sur l'opération de projection et sur un principe de négation par échec.
- nous avons enfin détaillé un principe de mise à jour utilisant un modèle explicite d'évolution du monde (les lois dynamiques).

10.2.Travaux futurs

L'ontologie comportementale pourrait être enrichie ou modifiée en posant des hypothèses différentes de celles que nous avons faites. Ces modifications devraient cependant être prises en compte dans la définition de la sémantique formelle et de l'algorithme de simulation.

Enrichir l'ontologie du domaine nécessiterait de réaliser des travaux de recherche concernant les ontologies spatiales, les ontologies sur les collections...

Le pouvoir d'expression dans les états, événements et transitions pourrait être augmenté en prenant en compte des extensions du formalisme de base des graphes conceptuels plus riches que celles que nous avons définies. Ces extensions concerneraient essentiellement la représentation de la négation et la représentation de la différence entre deux point-situations (ie, un changement d'état). Il serait alors nécessaire de redéfinir les principes de démonstration et mise à jour en les basant sur ces représentations plus expressives. Il faudrait également faire évaluer par l'expert la compréhension et la lisibilité de ces nouvelles extensions.

Un principe de démonstration reposant sur un monde ouvert dans les point-situations (représentation explicite de la négation) et prenant directement en compte les lois du domaine (chaînage arrière) serait également plus avantageux car les point-situations pourraient contenir des indéterminations et ils n'auraient pas besoin d'être complétés vis à vis des lois du domaine (ils devraient cependant être consistants vis à vis de celles-ci).

Il serait enfin intéressant d'explorer l'utilisation d'autres types de connaissances que les lois dynamiques pour l'opération de mise à jour. Il serait par exemple possible de distinguer des propriétés du monde de base et des propriétés déduites. Les effets des actions ne serait alors que des propriétés de base et le principe de démonstration permettrait aussi bien de déterminer la valeur de vérité des propriétés de base que des propriétés déduites.

10.3.Vers un système d'assistance à la modélisation de comportements

Le but ultime de ce travail est un système d'assistance à la modélisation de comportements dans une organisation humaine (figure 106). La modélisation pourra ensuite être exploitée pour construire diverses applications informatiques.

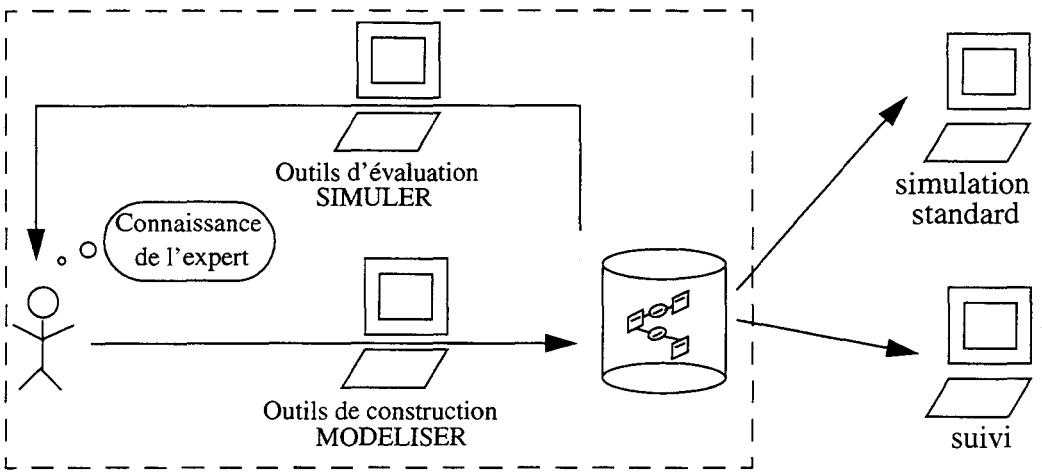


Figure 106. Un système d'assistance à la modélisation de comportements

Dans ce système, un premier ensemble d'outils aide le cogniticien à compléter l'ontologie, construire les modèles d'actions, les lois dynamiques et les comportements. Ces outils exploitent la notation graphique des graphes conceptuels par une interface graphique. Ils exploitent également les aspects terminologiques des graphes conceptuels (métaconcepts, héritage...) afin de guider la saisie et d'effectuer des contrôles sur les connaissances déjà saisies.

Un deuxième ensemble d'outils aide l'expert à évaluer la modélisation. Ces outils implémentent la simulation et gèrent l'interaction de l'expert et du cogniticien avec la simulation.

Nous avons développé une première maquette de ce système pendant cette thèse. Celle-ci implémente la simulation et comporte une interface graphique permettant de saisir la modélisation et de montrer les résultats de simulation. Dans cette maquette, les principaux aspects manquants sont les suivants:

- en ce qui concerne les outils de construction, il manque une méthodologie indiquant comment exploiter avantageusement les métaconcepts pour faciliter la modélisation.
- en ce qui concerne les outils d'évaluation, il faudrait une interaction plus riche de l'expert et du cogniticien avec la simulation (notamment afin de perturber la simulation)

Références

[ALL 83] J.F. Allen. "Maintaining knowledge about Temporal Intervals". *Communications of the ACM*, 26(11), pp 832-843, 1983

[ALL 84] J.F.Allen. "Towards a general theory of actions and time". *Artificial Intelligence* 23, pp 123-154, 1984

[ALL 94] J.M. Alliot, T. Schiex. *Intelligence Artificielle et Informatique Théorique*, eds Cepaduès, 1994

[BOS 96] C. Bos. "A Graph-based Approach of Knowledge Modelling for Simulation Systems". *Proceedings of CESA'96, Symposium on Modelling, Analysis and Simulation*, Lille, France, pp 688-693, 1996

[BOS 97a] C. Bos, B. Botella. "Modelling Stereotyped Behaviours in Human Organizations". *Proceedings of 7th Workshop on Knowledge Engineering: Methods and Languages*, Milton Keynes, United Kingdom, 1997

[BOS 97b] C. Bos, B. Botella. "Graphes Conceptuels Deuxième Partie: Application pour la Modélisation de Comportements Stéréotypés dans des Organisations Humaines". *Actes des 6è journées nationales du PRC-GDR Intelligence Artificielle*, Grenoble, France, eds Hermes, pp55-75, 1997

[BOS 97c] C. Bos, B. Botella. "Acquisition et Validation de Comportements Stéréotypés dans des Organisations Humaines". *Actes des Journées d'Ingénierie des Connaissances et Apprentissage Automatique*, Roscoff, France, JICAA'97, pp 305-316, 1997

[BOS 97d] C. Bos, B. Botella, P. Vanheeghe. "Modelling and Simulating Human Behaviours with Conceptual Graphs". *Proceedings of 5th International Conference on Conceptual Structures*, Seattle, USA, eds Springer Verlag, Lecture Notes in Artificial Intelligence, pp 275-289, 1997

[BRA 85] R.J. Brachman, J.G. Schmolze. "An overview of the KL-ONE knowledge representation system". *Cognitive Science*, 9(2), pp171-216, 1985

[BRE 94] J. Breuker, W. Van de Velde. *CommonKADS Library for Expertise Modelling, Reusable Problem Solving Components*, IOS Press, Amsterdam, 1994

[CAR 94] B.Carboneill, O.Haemmerle. "ROCK: un système de question/réponse fondé sur le formalisme des graphes conceptuels". *Actes du 9ème congrès Reconnaissance des Formes et Intelligence Artificielle*, RFIA'94, Paris, France, pp 159-169, 1994

[CAR 96] B. Carboneill. "Vers un système de représentation de connaissances et de raisonnement fondé sur les graphes conceptuels". Thèse de Doctorat en Informatique, Université Montpellier II, France, Janvier 1996

[CHE 92] M. Chein, M.L. Mugnier. "Conceptual Graphs: Fundamental Notions", *Revue d'Intelligence Artificielle,* 6(4), pp 365-406, 1992

[CHE 97a] M. Chein, M.L. Mugnier. "Positive Nested Conceptual Graphs", *Proceedings of 5th International Conference on Conceptual Structures*, Seattle, USA, eds Springer Verlag, Lecture Notes in Artificial Intelligence, pp 95-109, 1997

[CHE 97b] M.Chein, M.L.Mugnier, G.Simonet. "Nested Conceptual Graphs: Projection and FOL semantics", Rapport de recherche, LIRMM, #97003, 1997

[COR 94]] M.O. Cordier, P. Siegel. "A Temporal Revision Model for Reasoning about World Change", In B.Nebel, C.Rich, W.Swartout, eds, *Principles of Knowledge Representation and Reasoning* (KR'92), pp 732-739, Morgan Kaufmann, 1992

[COR 96] O. Corby, R. Dieng. "Cokace: A Centaur-based environment for CommonKADS Conceptual Modelling Language". *Proceedings of 12th European Conference on Artificial Intelligence*, ed. W. Wahlster, pp 418-422, 1996

[DEL 86] J.P. Delahaye. *Outils logiques pour l'intelligence artificielle*, eds Eyrolles, 1986

[EUZ 93] J. Euzenat. "Représentation granulaire du temps". *Revue d'Intelligence Artificielle*, 7(3), pp 329-361, 1993

[FEN 91] D. Fensel, J. Angele, D. Landes. "KARL: A Knowledge Acquisition and Representation Language". *Proceedings of Experts Systems and their Applications, 11th International Workshop Conference Tools, Techniques and Methods*, Avignon, France,1991

[FEN 94] D. Fensel, F. van Harmelen, "A Comparison of Languages which Operationalise and Formalise KADS Models of Expertise", *The Knowledge Engineering Review* 9, pp 105-146, 1994

[FIK 88] R.E. Fikes, N.J. Nilsson. "STRIPS: a new approach to the application of theorem proving to problem solving". *Artificial Intelligence* 35, pp 165-195, 1988.

[FIL 68] C.J.Fillmore. "The case for case". *Universals in Linguistic Theory,* eds E.Bach R.Harms, New York: Holt, Reinhart and Winston, 1968

[FRE 92] C. Freksa. "Temporal reasoning based on semi-intervals". *Artificial Intelligence* 54, pp 199-227, 1992

[FRI 97] D.B. Fridsma, J. Gennari, M. Musen. "Acquiring, Maintaining and Customizing Organizational Work Process Descriptions". *AAAI Spring Symposium, Proceedings of AIKM'97, Artificial Intelligence in Knowledge Management*, Stanford University, USA, 1997

[GAM 84] J.G. Gammack, R.M. Young. "Psychological Techniques for Eliciting Expert Knowledge". *Research and Development in Expert Systems, 4th Expert System Conference*, ed. M. Bramer, London, UK, 1984

[GAR 93] F. Garcia. "Révision des croyances et révision du raisonnement pour la planification". Thèse de Doctorat en Intelligence Artificielle, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, France, Février 1993

[GHA 89] M. Ghallab, A.M. Alaoui. "Relations temporelles symboliques: représentations et algorithmes". *Revue d'Intelligence Artificielle*, 3(3), pp 67-115, 1989

[GIN 88a] M.L. Ginsberg, D.E. Smith, "Reasoning about Actions I: A Possible Worlds Approach", *Artificial Intelligence* 35, pp 165-195, 1988

[GIN 88b] M.L. Ginsberg, D.E. Smith, "Reasoning about Actions II: The Qualification Problem", *Artificial Intelligence* 35, pp 311-342, 1988

[HAE 95] O. Hammerlé. "CoGITo: Une plate-forme de développement de logiciels sur les graphes conceptuels". Thèse de Doctorat en Informatique, Université Montpellier II, France, Février 1995

[HAN 87] S. Hanks, D. McDermott. "Nonmonotonic Logic and Temporal Projection". *Artificial Intelligence* 33, pp 379-412, 1987

[HAR 87] D. Harel, "Statecharts: a Visual Formalism for Complex Systems". *Science of Computer Programming* 8, pp 231-274, 1987

[HAR 92] F. van Harmelen, J. Balder. "$(ML)^2$: A Formal Language for KADS Conceptual Models". *Knowledge Acquisition*, 4(1),1992

[HEI 96] G. Heijst, A.T. Schreiber, B.J. Wielinga. "Using Explicit Ontologies in Knowledge Base System Development". *International Journal of Human-Computer Studies/ Knowledge Acquisition*, 1996

[JAC 90] R. Jackendoff. *Semantic Structures*, MIT Press, 1990

[KAT 91] H. Katsuno, A.O. Mendelzon. "On the Difference between Updating a Knowledge Base and Revising it". *Proceedings of KR'91, Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, pp 387-394, 1991

[KER 97] G. Kerdiles, E. Salvat. "A Sound and Complete Conceptual Graph Proof Procedure Combining Projections with Analytic Tableaux". *Proceedings of 5th International Conference on Conceptual Structures*, Seattle, USA, eds Springer Verlag, Lecture Notes in Artificial Intelligence, pp 248-262, 1997

[KOW 86] R. Kowalski, M. Sergot. "A logic-based calculus of events". *New Generation Computing*, vol 4, pp 67-95, 1986

[LAP 97] S. Lapalut. "Sémantique formelle et spécifications algébriques du raisonnement sur les graphes conceptuels simples et étendus". Thèse de Doctorat en Informatique, Université de Nice-Sophia Antipolis, France, Juin 1997

[LEC 95] M. Leclère. "Les connaissances du niveau terminologique du modèle des graphes conceptuels: construction et exploitation". Thèse de Doctorat en Informatique, Université Montpellier II, France, Décembre 1995

[LEC 96] M. Leclère. "C-CHiC: construction coopérative de hiérarchies de catégories". *Revue d'Intelligence Artificielle*, 10(1), 1996

[LEN 95] D.B. Lenat. "CYC: A Large-Scale Investment in Knowledge Infrastructure". *Communications of the ACM*, 38(11), 1995 voir aussi http:// www.cyc.com

[MAR 95] P. Martin. "Knowledge Acquisition using Documents, Conceptual Graphs and a Semantically Structured Dictionary", *Proceedings of 9th International Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, KAW'95, 1995

[MAR 96] P. Martin. "Exploitation de graphes conceptuels et de documents structurés et hypertextes pour l'acquisition de connaissances et la recherche d'informations". Thèse de Doctorat en Informatique, Université de Nice-Sophia Antipolis, France, Octobre 1996

[MCC 69] J. Mc Carthy, P.J. Hayes, "Some Philosophical Problems From the Standpoint of Artificial Intelligence", *Machine Intelligence* 4, eds B.Meltzer, D.Michie, pp 463-502, 1969

[MCD 82] D.V. Mc Dermott. "A Temporal Logic for Reasoning about Processes and Plans". *Cognitive Science*, 6, pp 65-74, 1982

[MIK 97] S. Miksch, Y. Shahar, P. Johnson. "Asbru: a Task-Specific, Intention-Based, and Time-Oriented Language for Representing Skeletal Plans". *Proceedings of 7th Workshop on Knowledge Engineering: Methods and Languages*, The Open University, Milton Keynes, UK, 1997

[MIL 93] G.A.Miller, R.Beckwith, C.Fellbaum, D.Gross, K.Miller. "Five Papers on Wordnet". *CSL Report,* 43, Cognitive Science Laboratory, Princetown University, 1993

[MUG 93] M.L. Mugnier, M. Chein. "Characterization and algorithmic recognition of canonical conceptual graphs". *Proceedings of 1st International Conference on Conceptual Structures*, Quebec City, Canada, eds Springer Verlag, Lecture Notes in Artificial Intelligence, pp 294-311, 1993

[MUG 95] M.L. Mugnier, M. Chein. "Représenter des connaissances et raisonner avec des graphes". *Revue d'Intelligence Artificielle,* 10(1), pp 365-406, 1995

[NEB 90] B. Nebel. *Reasoning and revision in hybrid representation systems*, eds Springer Verlag, Lecture Notes in Artificial Intelligence, 1990

[PET 81] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, 1981

[REI 77] R. Reiter. "On Closed World Data Bases". *Proceedings of the Symposium on Logic and Data Bases*, Toulouse, France, 1977

[RUM 96] J.Rumbaugh, M. Blaha, F. Eddy, W. Premerlani, W. Lorensen. *OMT 1. Modélisation et conception orientés objet*, eds Masson, Prentice Hall, 1996

[SAL 96] E.Salvat, M.L.Mugnier. "Sound and Complete Forward and Backward Chainings of Graph Rules", *Proceedings of 4th International Conference on Conceptual Structures*, Melbourne, Australia, eds Springer Verlag, Lecture Notes in Artificial Intelligence, 1996

[SAL 97] E. Salvat. "Graphes Conceptuels Première Partie: Graphes Conceptuels Emboîtés et Règles d'Inférence", *Actes des 6è journées nationales du PRC-GDR Intelligence Artificielle*, Grenoble, France, eds Hermes, pp 31-54, 1997

[SCH 90] P.H. Schmitt, W. Wernecke. "Tableau Calculus for Order-Sorted Logic with Term Declarations". *Sorts and Types in Artificial Intelligence*, eds Springer Verlag, Lecture Notes in Artificial Intelligence 418, pp 49-60, 1990

[SCH 88] G.Schreiber, J.Breuker, B.Bredeweg, B. Wielinga. "Modelling in KBS Development". *Proceedings of European Knowledge Acquisition Workshop*, EKAW'88, 1988

[SCI 90] Sécurité Civile. Plan Particulier d'Intervention de l'usine chimique de Chocques dans le Pas de Calais, 1990

[SOW 84] J.Sowa. *Conceptual Structures: Information processing in mind and machine*, Addison Wesley, Reading Mass, 1984

[SOW 93] J. Sowa. "Relating Diagrams to Logic". *Proceedings of 1st International Conference on Conceptual Structures*, Quebec City, Canada, eds Springer Verlag, Lecture Notes in Artificial Intelligence, pp 1-35, 1993

[VIL 86] M. Vilain, H. Kautz. "Constraint Propagation Algorithms for Temporal Reasoning". *Proceedings of AAAI*, pp 377-382, 1986

[VOG 88] C. Vogel. *Génie Cognitif*, ed. Masson, 1988

[WIE 92] B. Wielinga, G. Schreiber, J. Breuker. "KADS: a modelling approach to knowledge engineering". *Knowledge Acquisition*, 4(1), 1992

[WIN 88] M. Winslett, "Reasoning about Actions using a Possible Models Approach", *Proceedings of AAAI*, pp 89-93, 1988

Annexe A

Exemple de fiche réflexe

Dans l'usine chimique de Chocques dans le Pas-de-Calais, le témoin d'un sinistre doit:

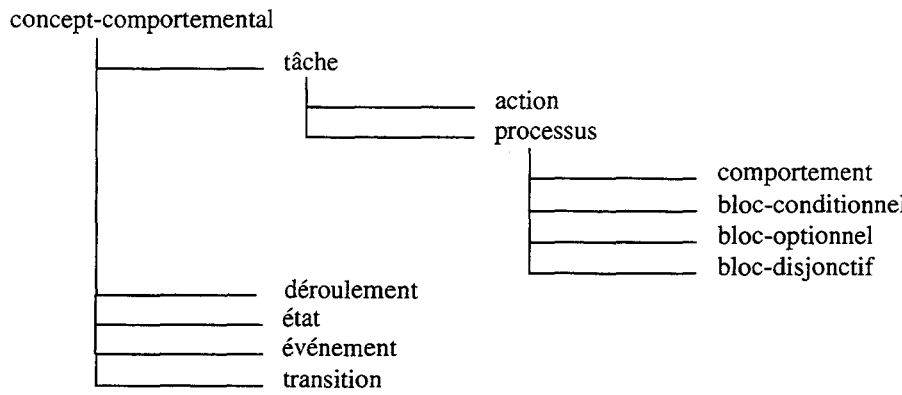
- Identifier le sinistre
- Décider de déclencher l'alerte
- Appuyer sur le bouton d'alarme le plus proche
- Téléphoner à la salle de contrôle poste 373
- Transmettre le message suivant:

  lieu exact du sinistre
  nature du sinistre (feu, fuite toxique ou présentant un risque d'inflammation ou d'explosion, accident corporel, pollution grave)
  importance du sinistre

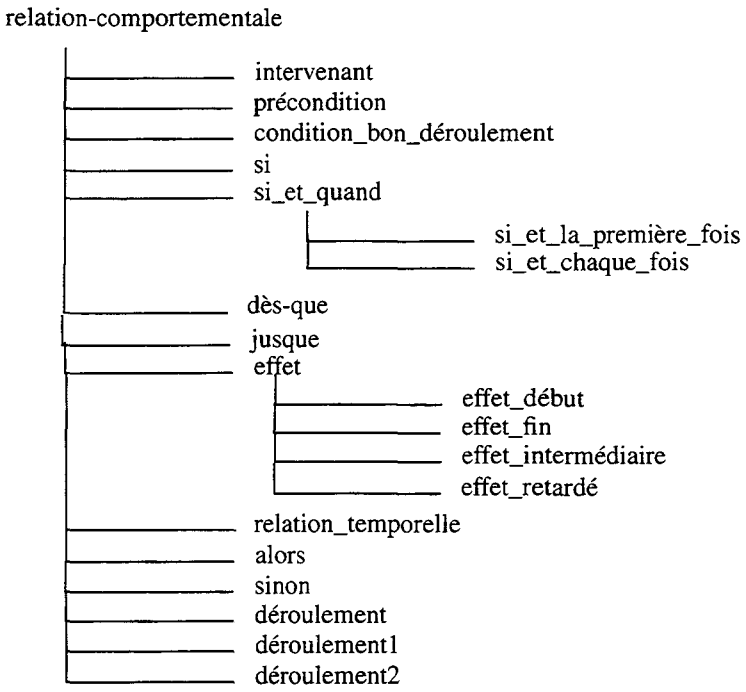
Annexe B

L'ontologie comportementale

B.1. Hiérarchie des types de concepts comportementaux

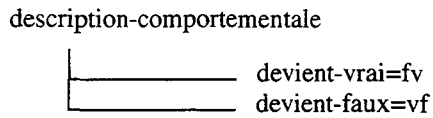


B.2. Hiérarchie des types de relations comportementales



(cf annexe C pour la hiérarchie des relations temporelles)

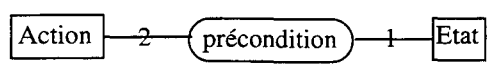
B.3. Hiérarchie des types d'emboîtements comportementaux

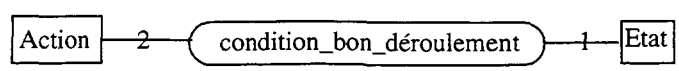


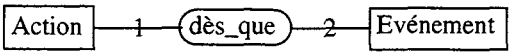
B.4. Métaconcepts associés aux types de concepts

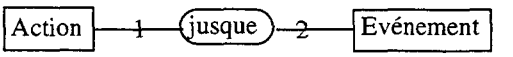
B.4.1. Action

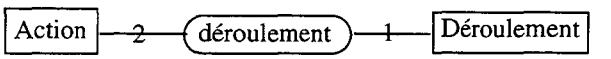
Acceptions

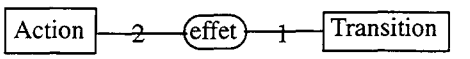


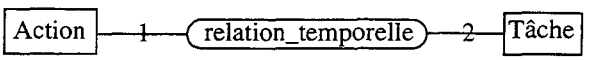


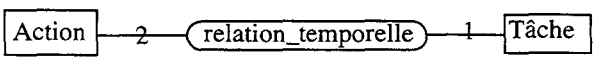




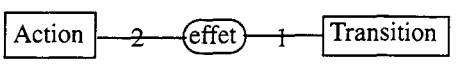


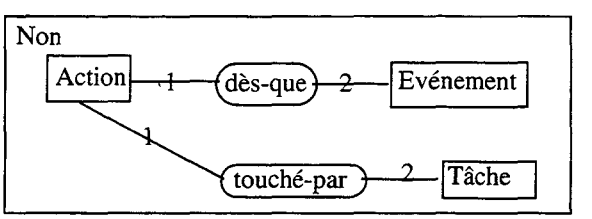


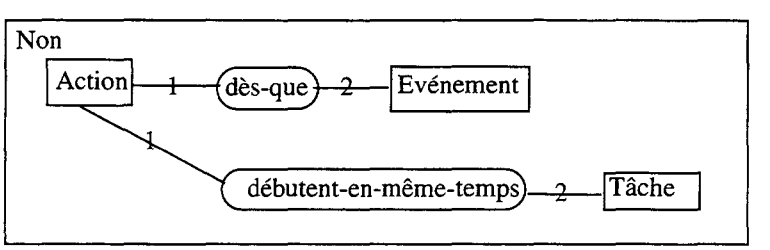


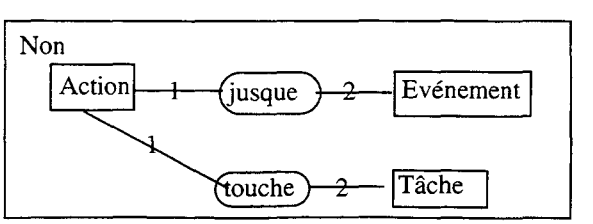


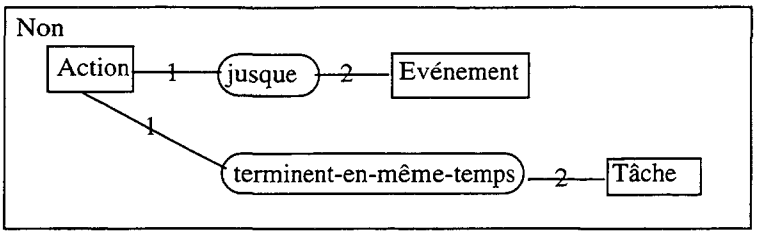
Conditions nécessaires





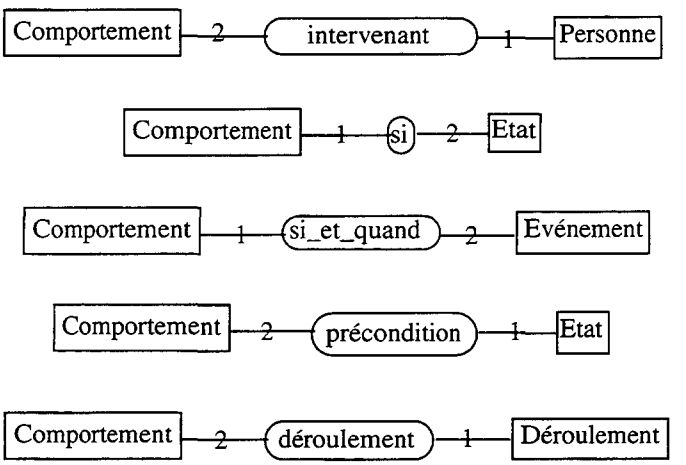




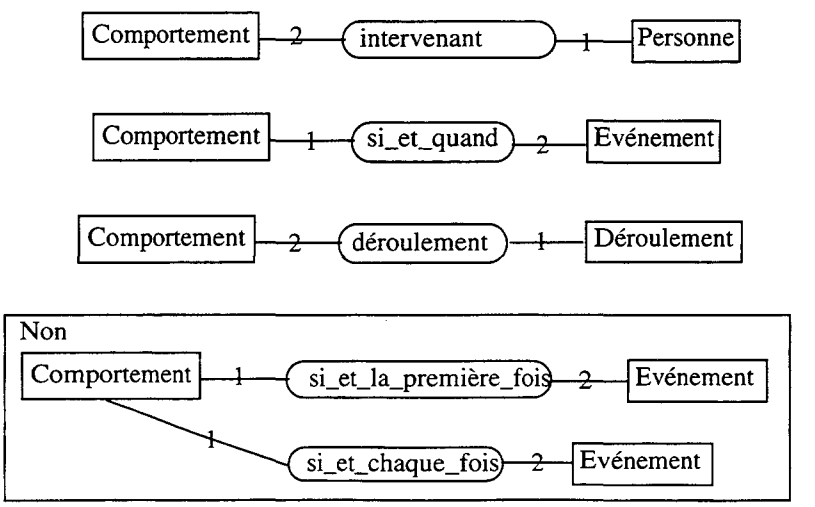


B.4.2.Comportement

Acceptions

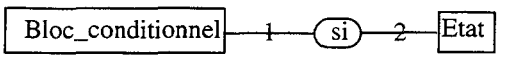


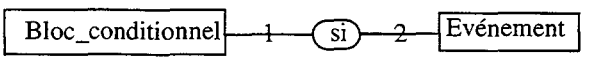
Conditions nécessaires

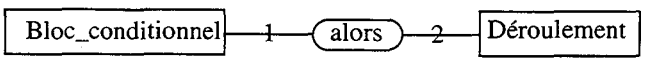


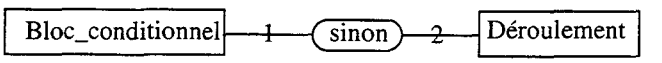
B.4.3.Bloc_conditionnel

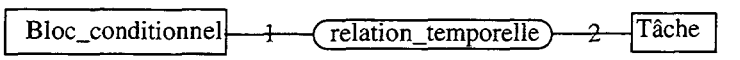
Acceptions

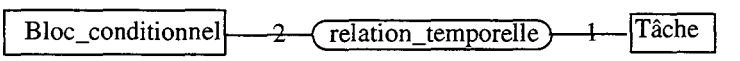




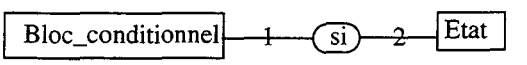


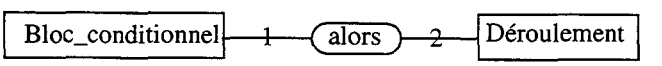


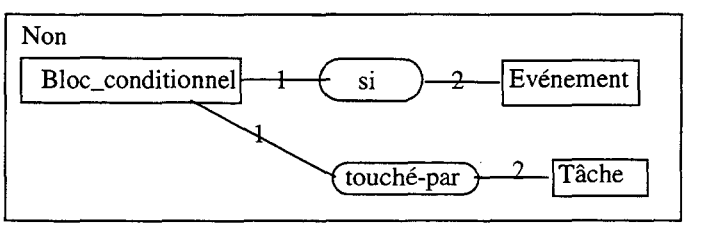


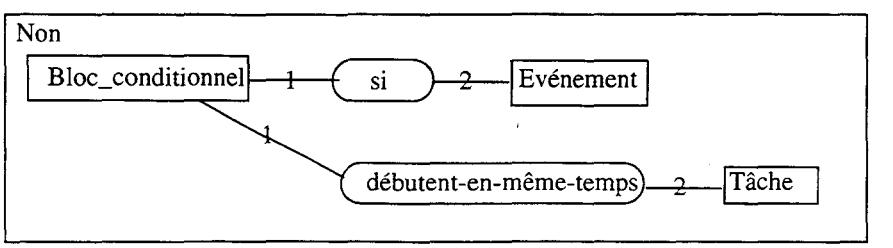


Conditions nécessaires



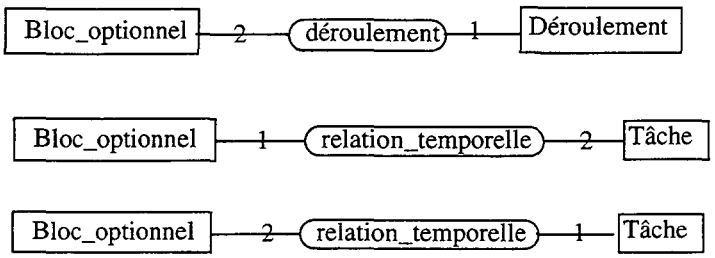




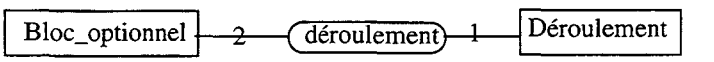


B.4.4.Bloc_optionnel

Acceptions

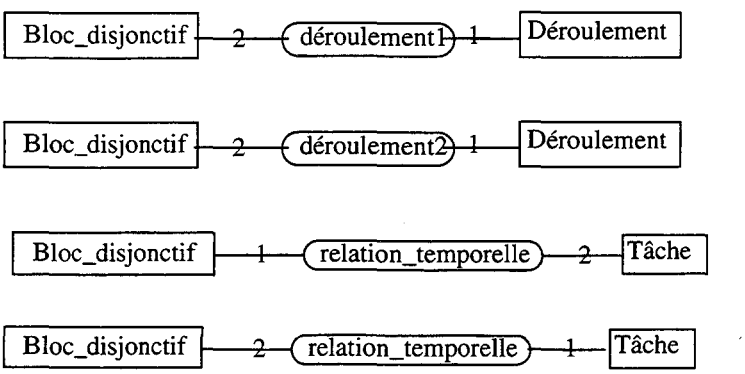


Conditions nécessaires

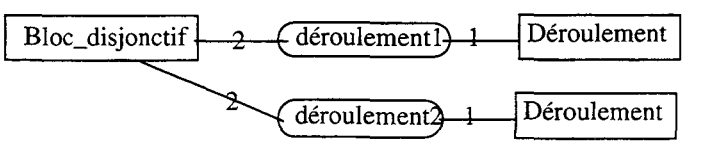


B.4.5.Bloc_disjonctif

Acceptions

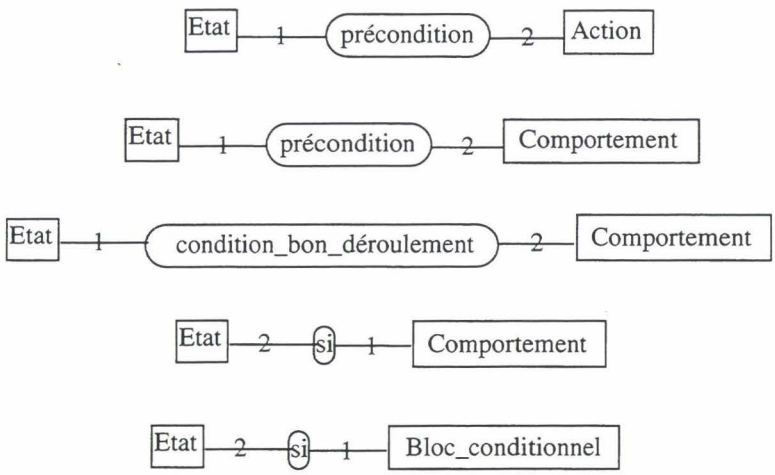


Conditions nécessaires



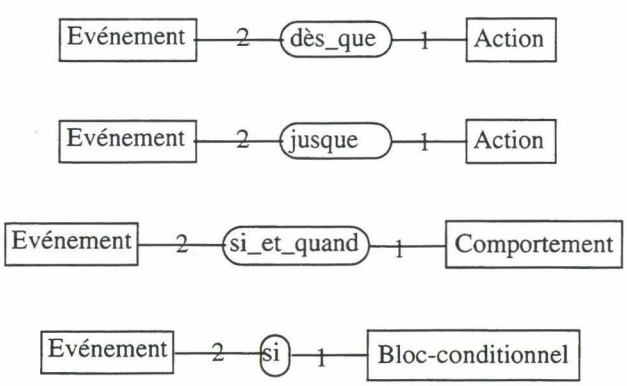
B.4.6.Etat

Acceptions



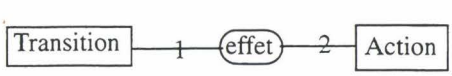
B.4.7.Evénement

Acceptions



B.4.8.Transition

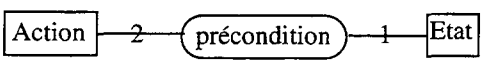
Acceptions

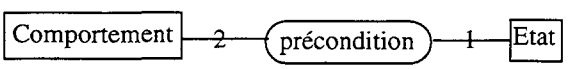


B.5. Signatures des types de relations

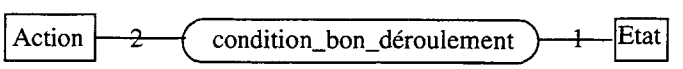
B.5.1.intervenant

B.5.2.précondition

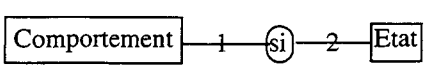


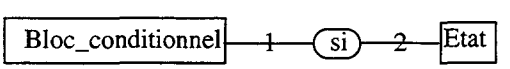


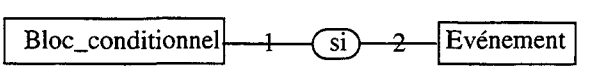
B.5.3.condition_bon_déroulement



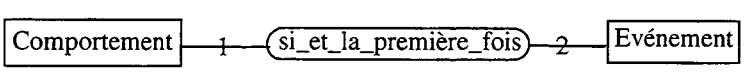
B.5.4.si



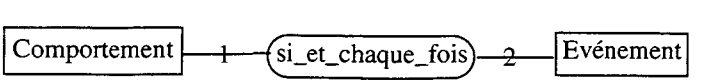




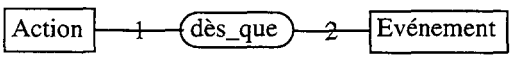
B.5.5.si_et_la_première_fois



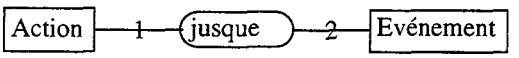
B.5.6.si_et-chaque_fois



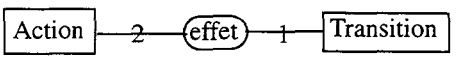
B.5.7.dès_que



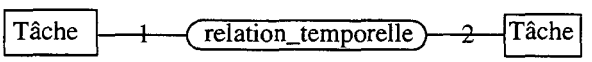
B.5.8.jusque



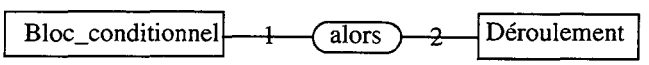
B.5.9.effet (début, fin, intermédiaire ou retardé)



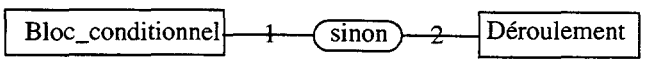
B.5.10.relation_temporelle



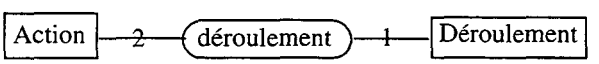
B.5.11.alors

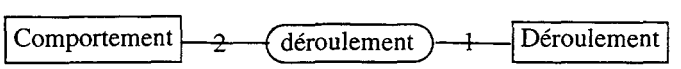


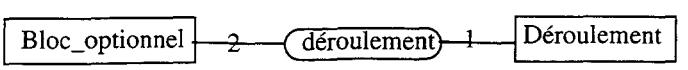
B.5.12.sinon



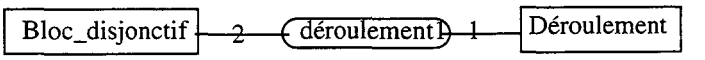
B.5.13.déroulement



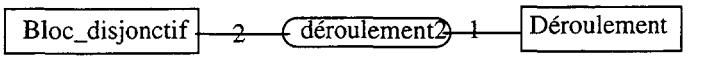




B.5.14.Déroulement1

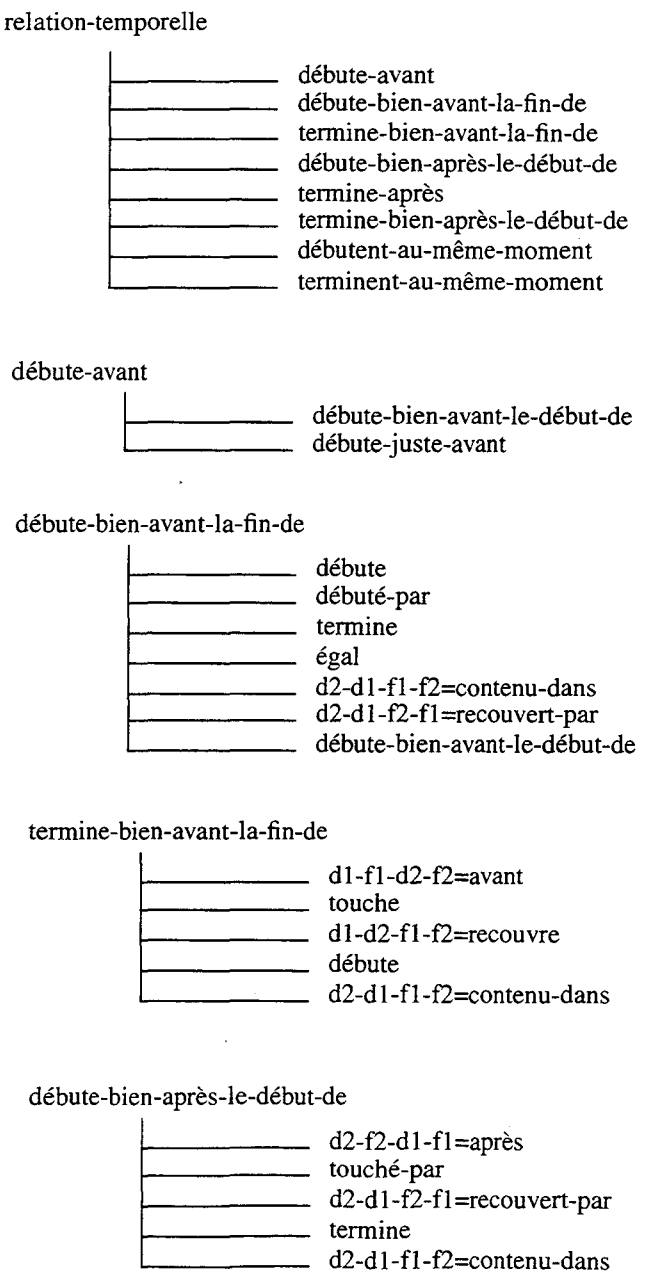


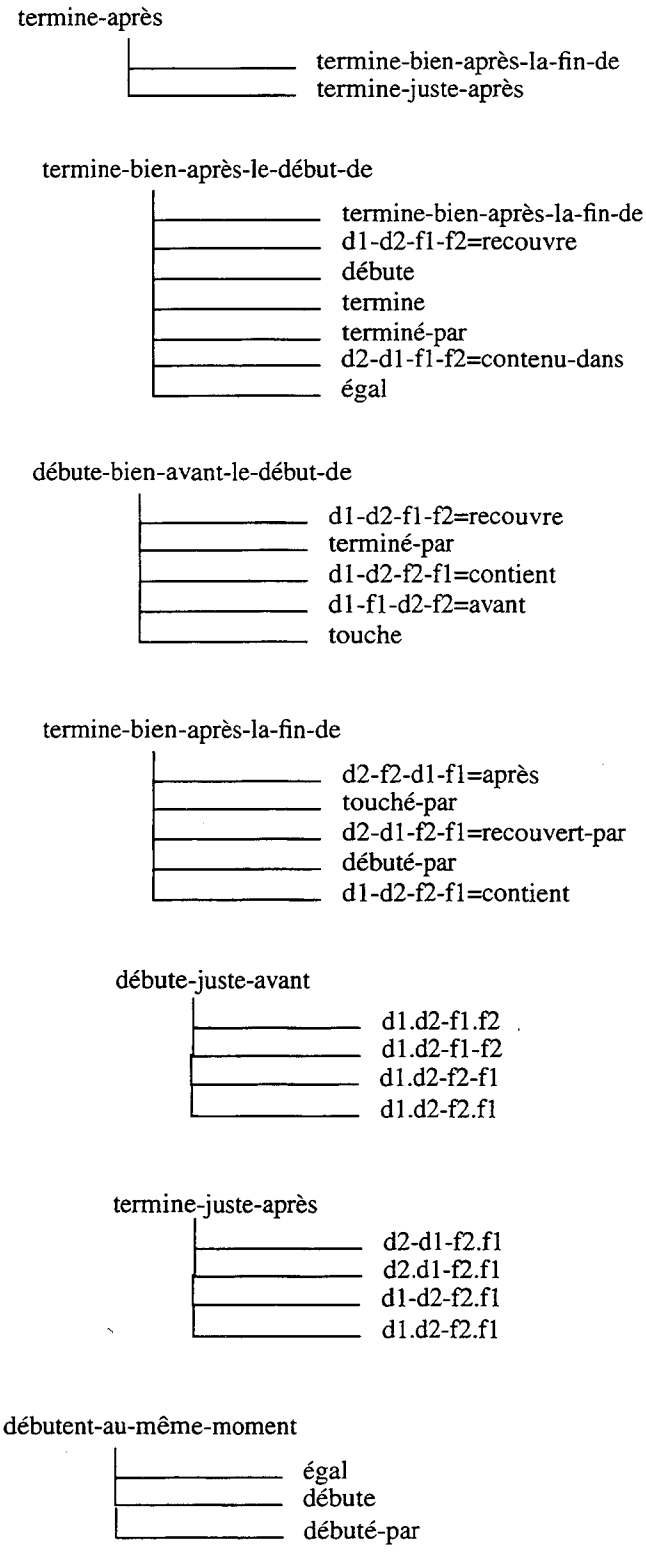
B.5.15.Déroulement2

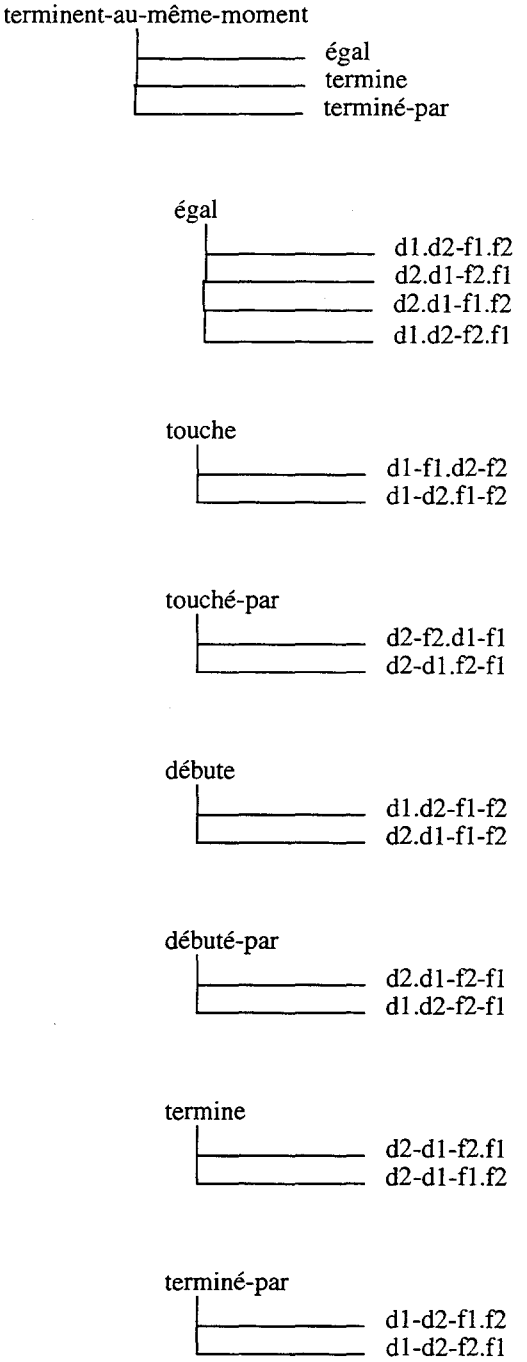


Annexe C

Les relations temporelles



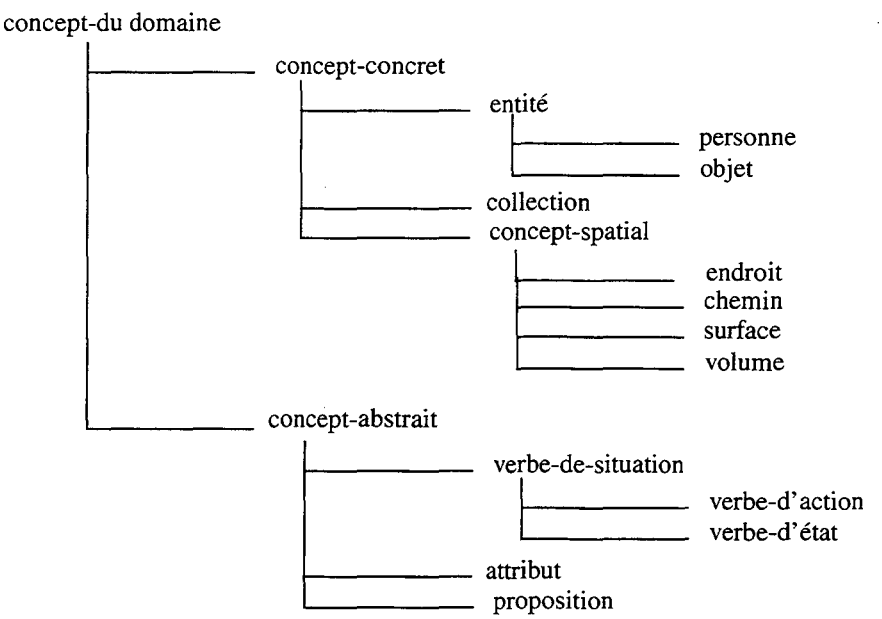

termine-après
termine-bien-après-la-fin-de
termine-juste-après
termine-bien-après-le-début-de
termine-bien-après-la-fin-de
d1-d2-f1-f2=recouvre
débute
termine
terminé-par
d2-d1-f1-f2=contenu-dans
égal
débute-bien-avant-le-début-de
d1-d2-f1-f2=recouvre
terminé-par
d1-d2-f2-f1=contient
d1-f1-d2-f2=avant
touche
termine-bien-après-la-fin-de
d2-f2-d1-f1=après
touché-par
d2-d1-f2-f1=recouvert-par
débuté-par
d1-d2-f2-f1=contient
débute-juste-avant
d1.d2-f1.f2
d1.d2-f1-f2
d1.d2-f2-f1
d1.d2-f2.f1
termine-juste-après
d2-d1-f2.f1
d2.d1-f2.f1
d1-d2-f2.f1
d1.d2-f2.f1
débutent-au-même-moment
égal
débute
débuté-par


terminent-au-même-moment
égal
termine
terminé-par
égal
d1.d2-f1.f2
d2.d1-f2.f1
d2.d1-f1.f2
d1.d2-f2.f1
touche
d1-f1.d2-f2
d1-d2.f1-f2
touché-par
d2-f2.d1-f1
d2-d1.f2-f1
débute
d1.d2-f1-f2
d2.d1-f1-f2
débuté-par
d2.d1-f2-f1
d1.d2-f2-f1
termine
d2-d1-f2.f1
d2-d1-f1.f2
terminé-par
d1-d2-f1.f2
d1-d2-f2.f1

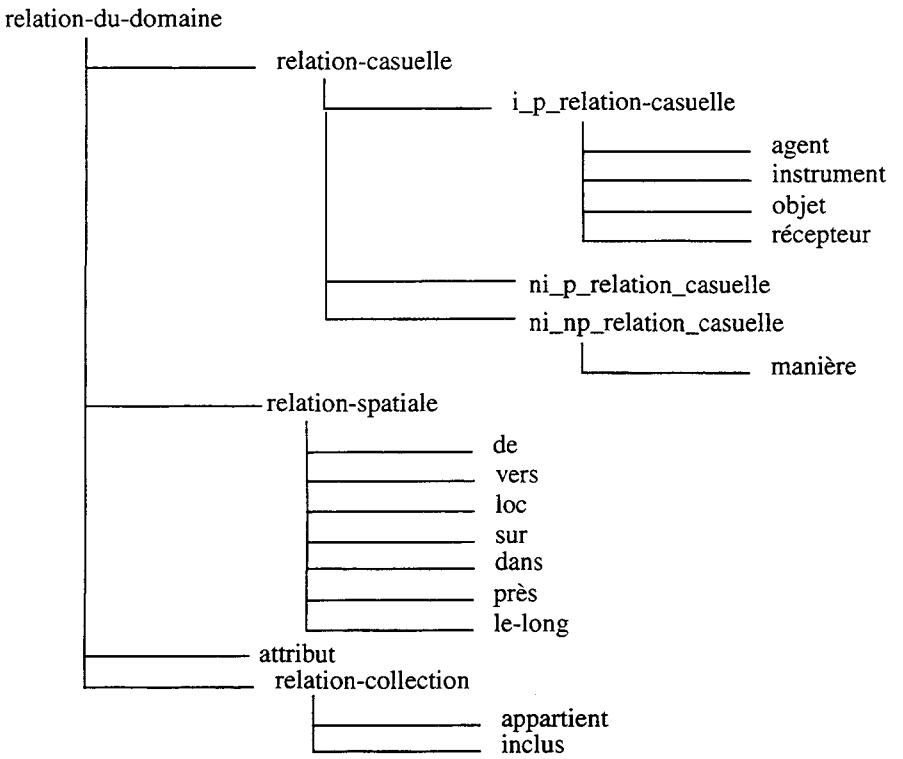
Annexe D

L'ontologie générique du domaine

D.1. Hiérarchie des types de concepts



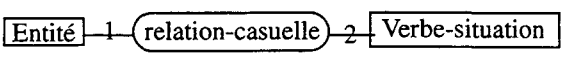
D.2. Hiérarchie des types de relations

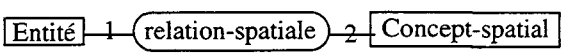


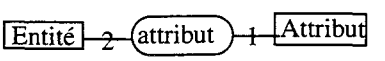
D.3. Métaconcepts associés aux types de concept

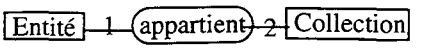
D.3.1. Entité

Acceptions



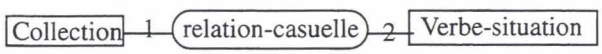


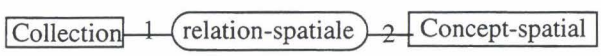


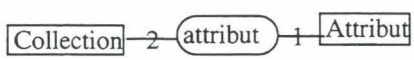


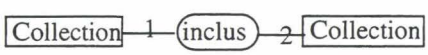
Conditions nécessaires

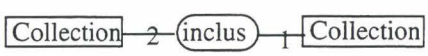
D.3.2.Collection



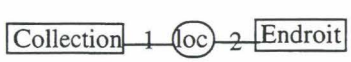






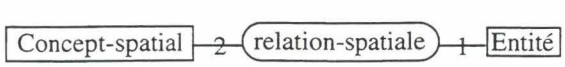


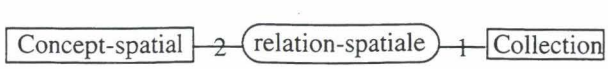
Conditions nécessaires

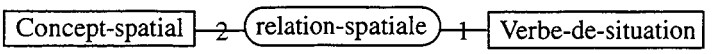


D.3.3.Concept-spatial

Acceptions

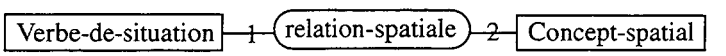


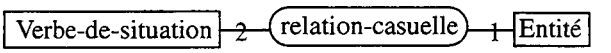


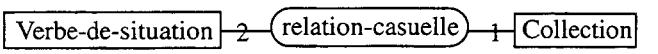


D.3.4.Verbe-de-situation

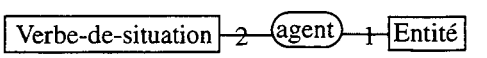
Acceptions



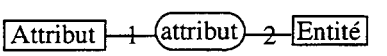


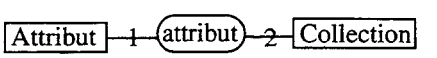


Conditions nécessaires



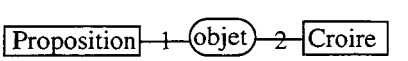
D.3.5.Attribut

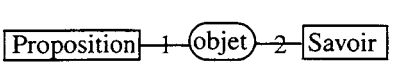


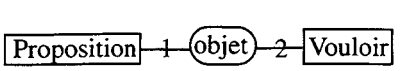


D.3.6.Proposition

Acceptions

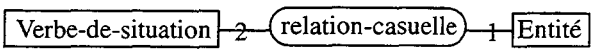


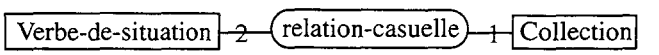




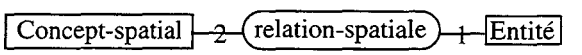
D.4.Signatures associées aux types de relations

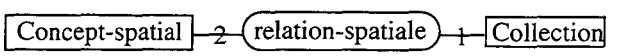
D.4.1.relation-casuelle

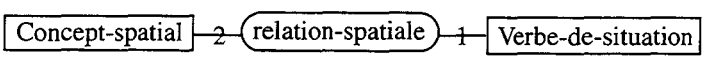




D.4.2.relation-spatiale

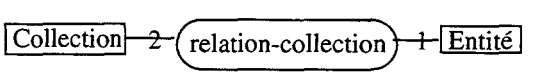


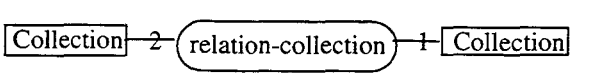




D.4.3.attribut

D.4.4.relation-collection

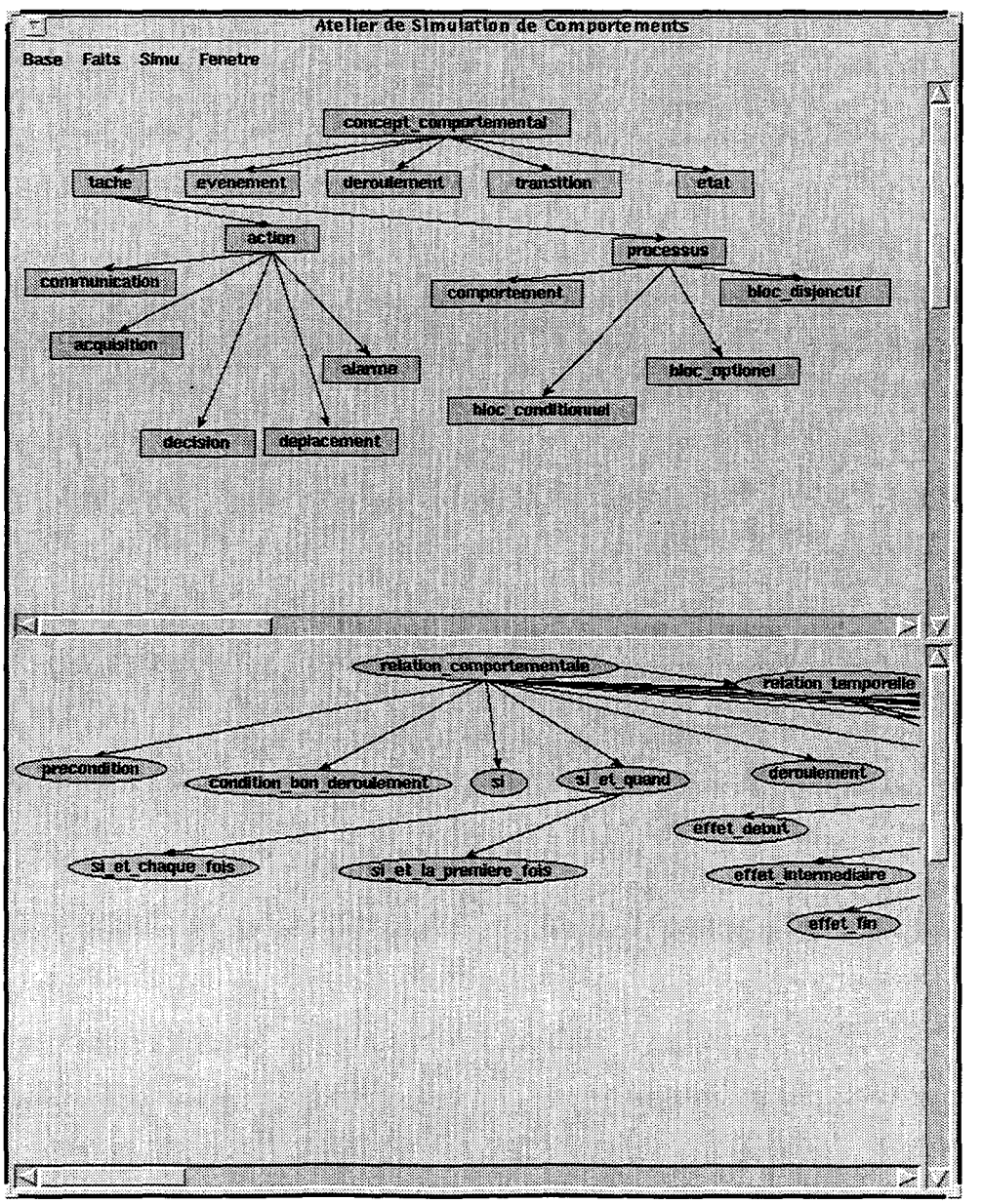




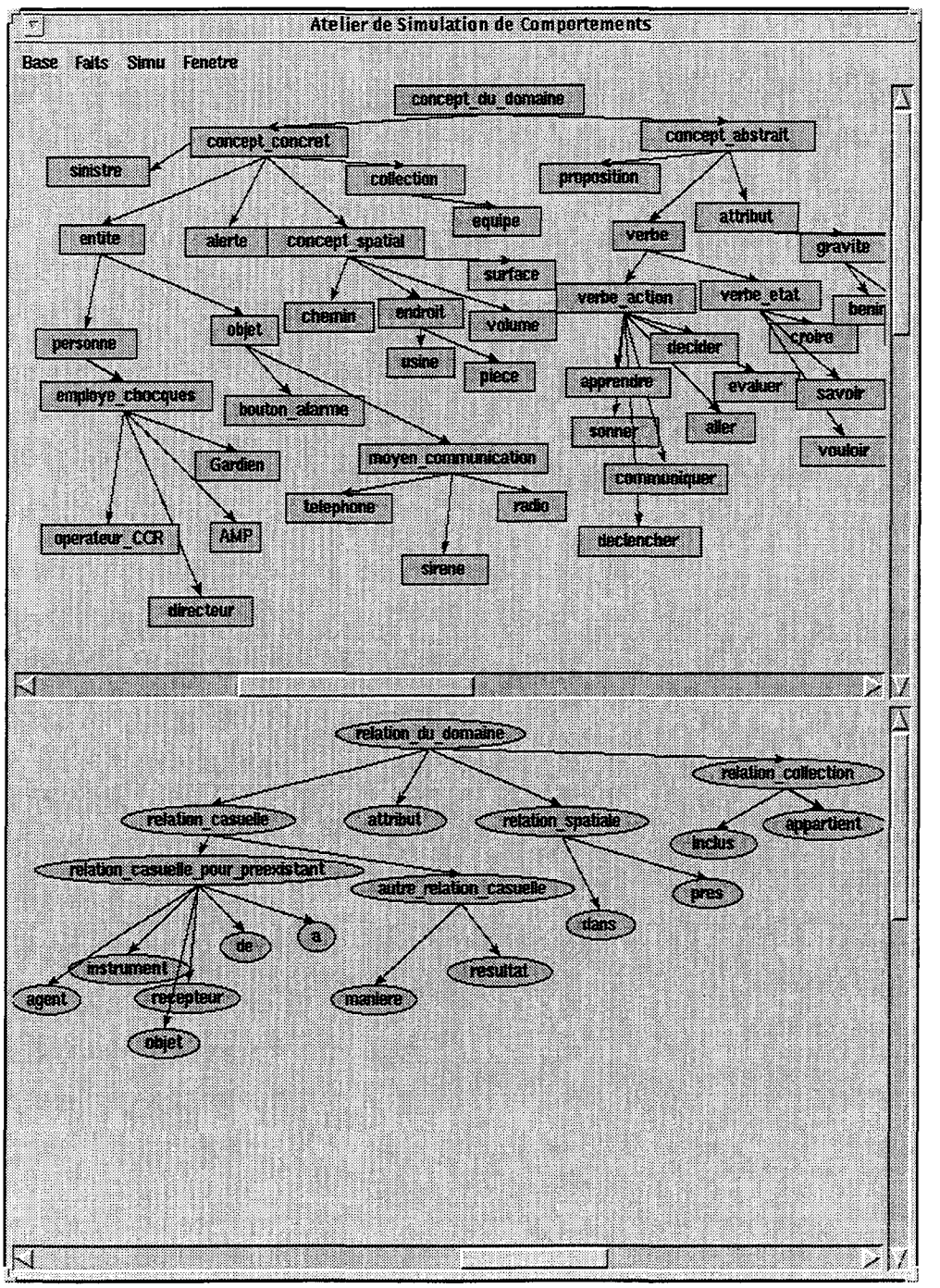
Annexe E

Exemple de modélisation et résultats de simulation

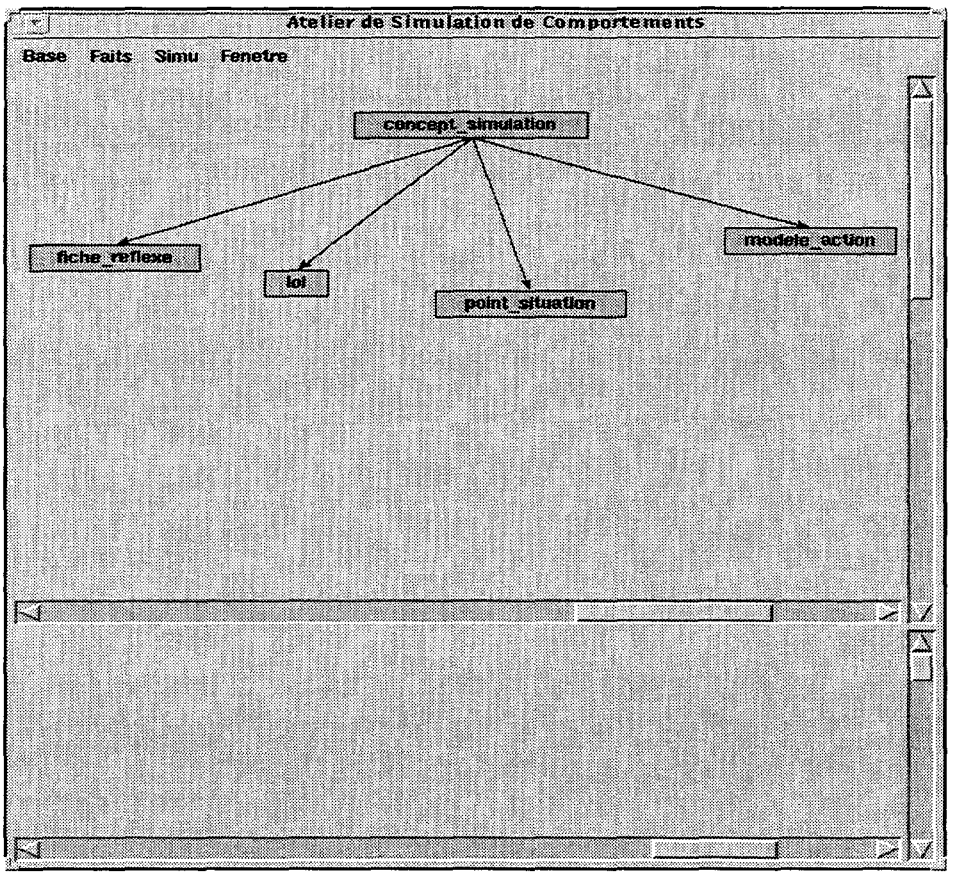
E.1. Ontologie comportementale



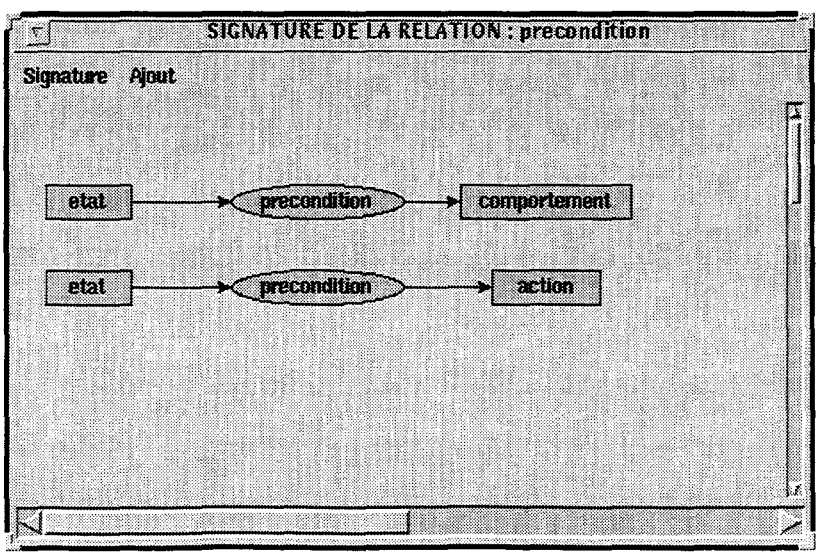
E.2. Ontologie du domaine



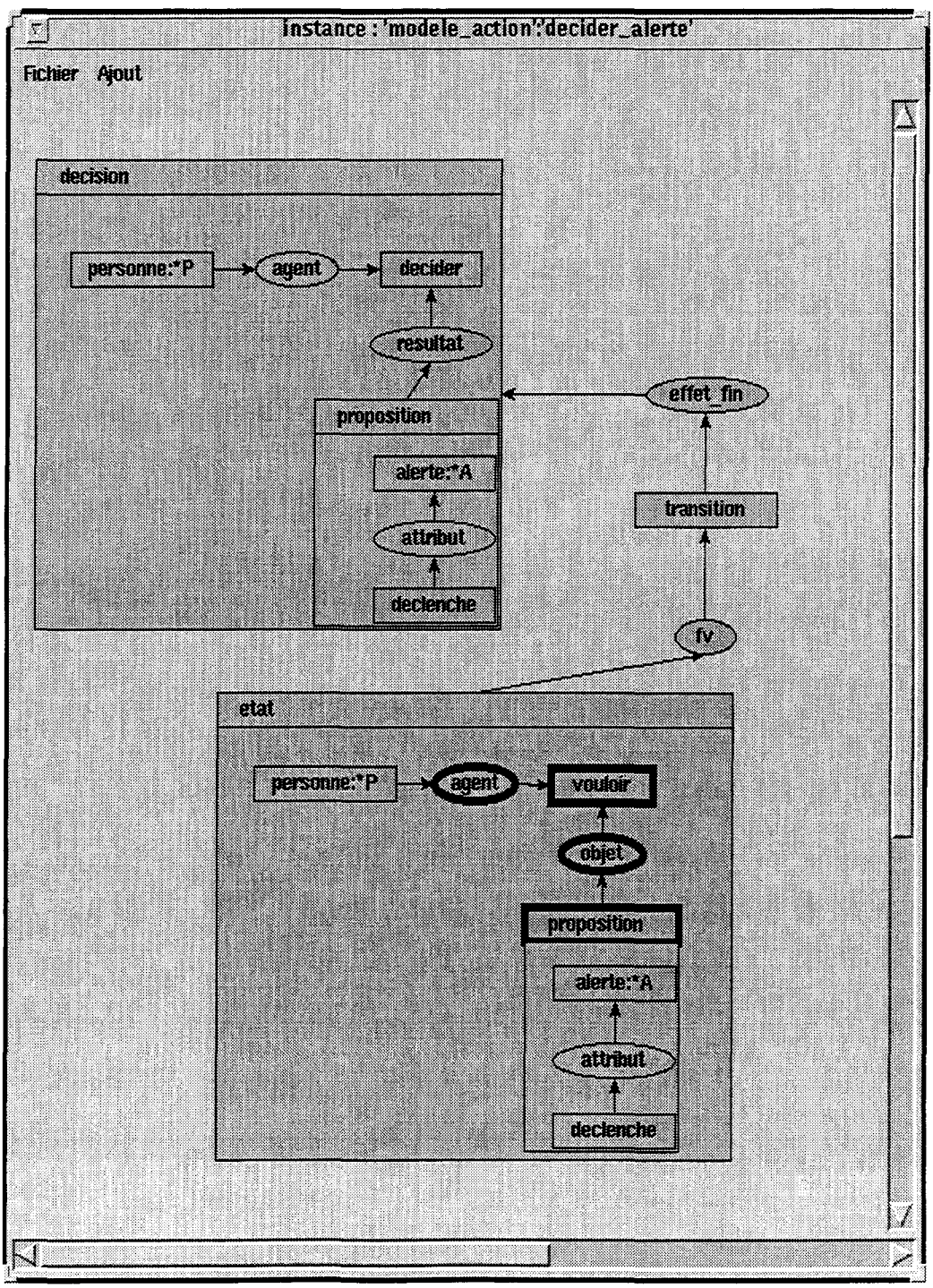
E.3. Ontologie pour la simulation

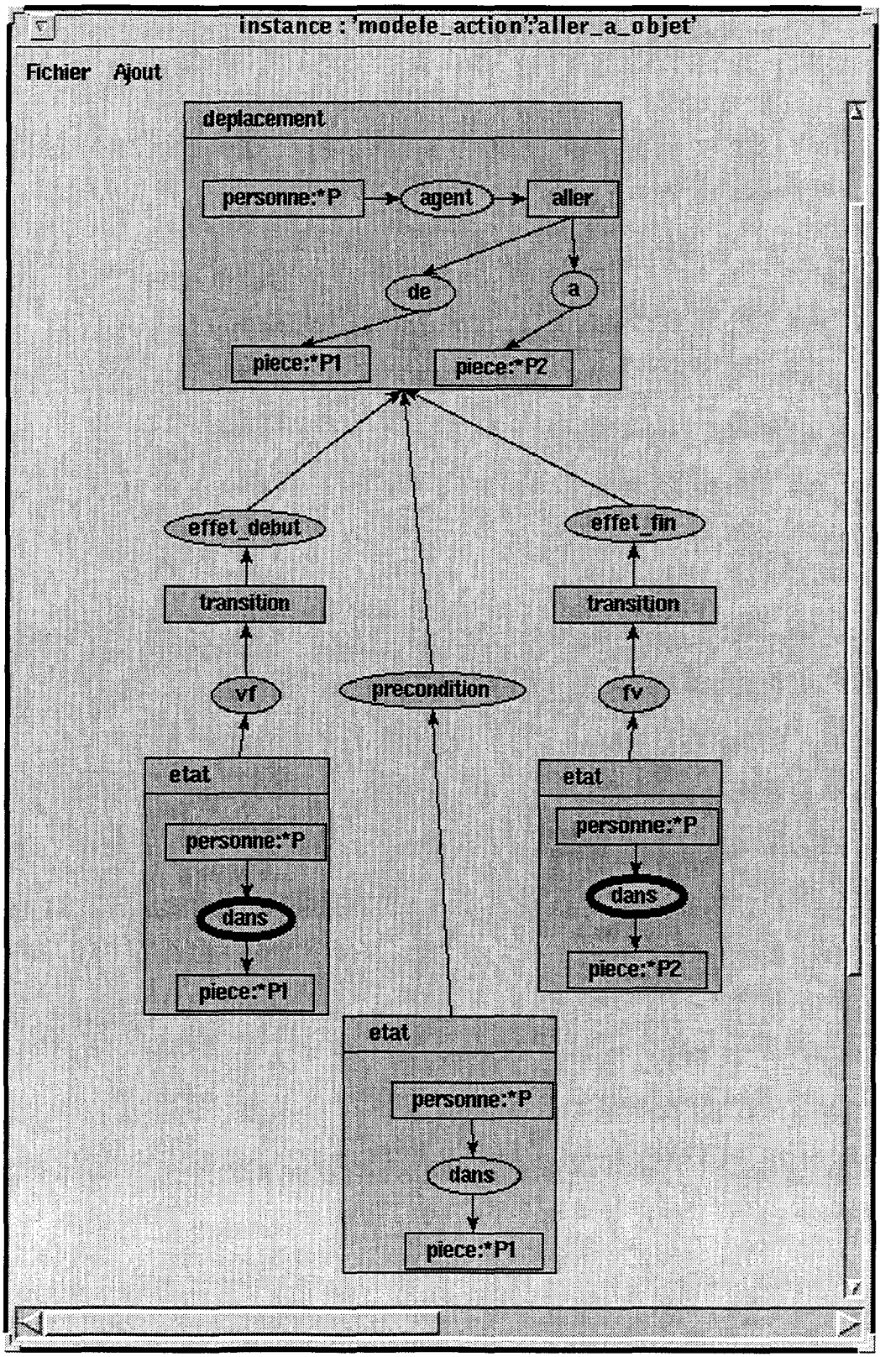


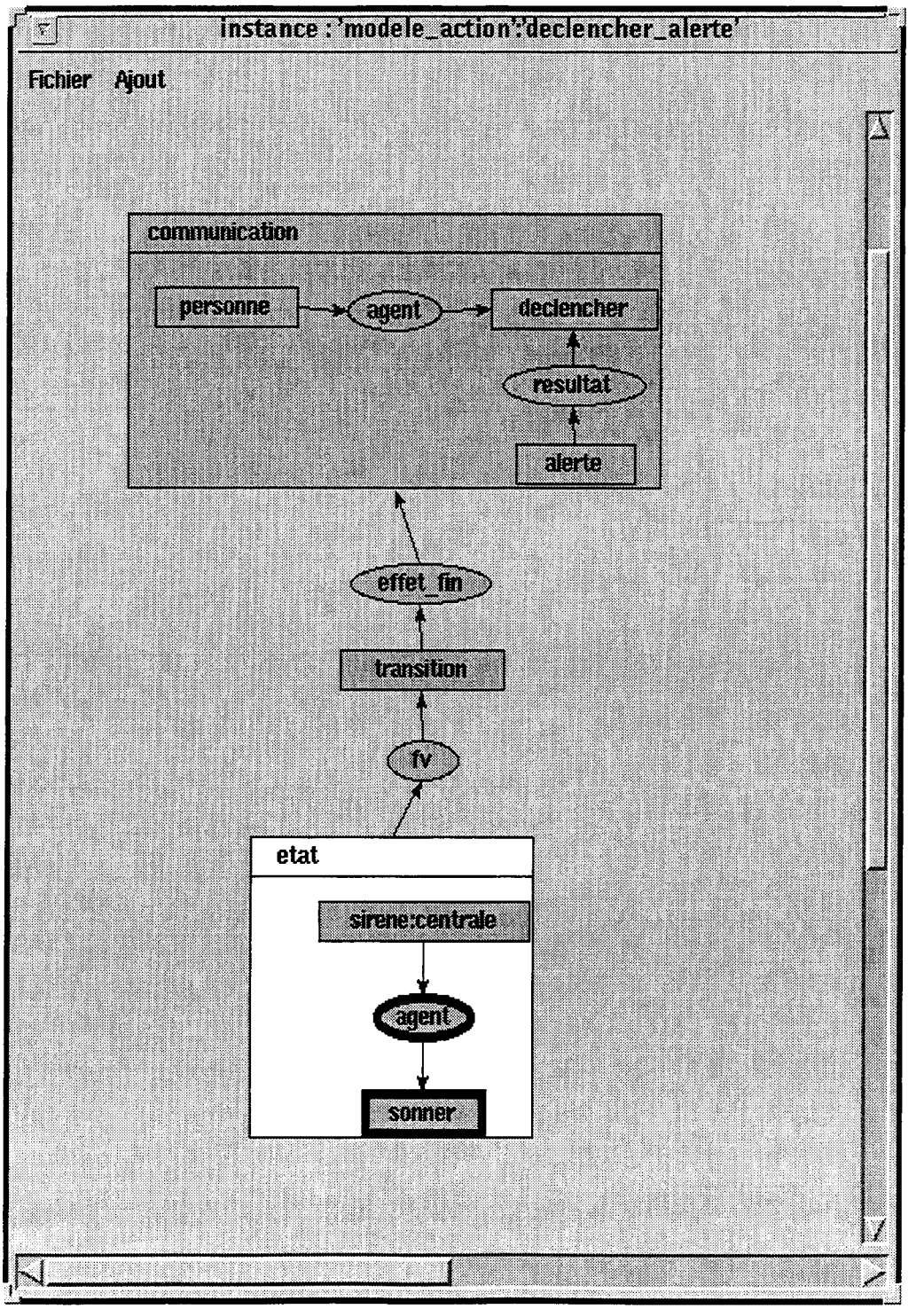
E.4. Métaconcepts



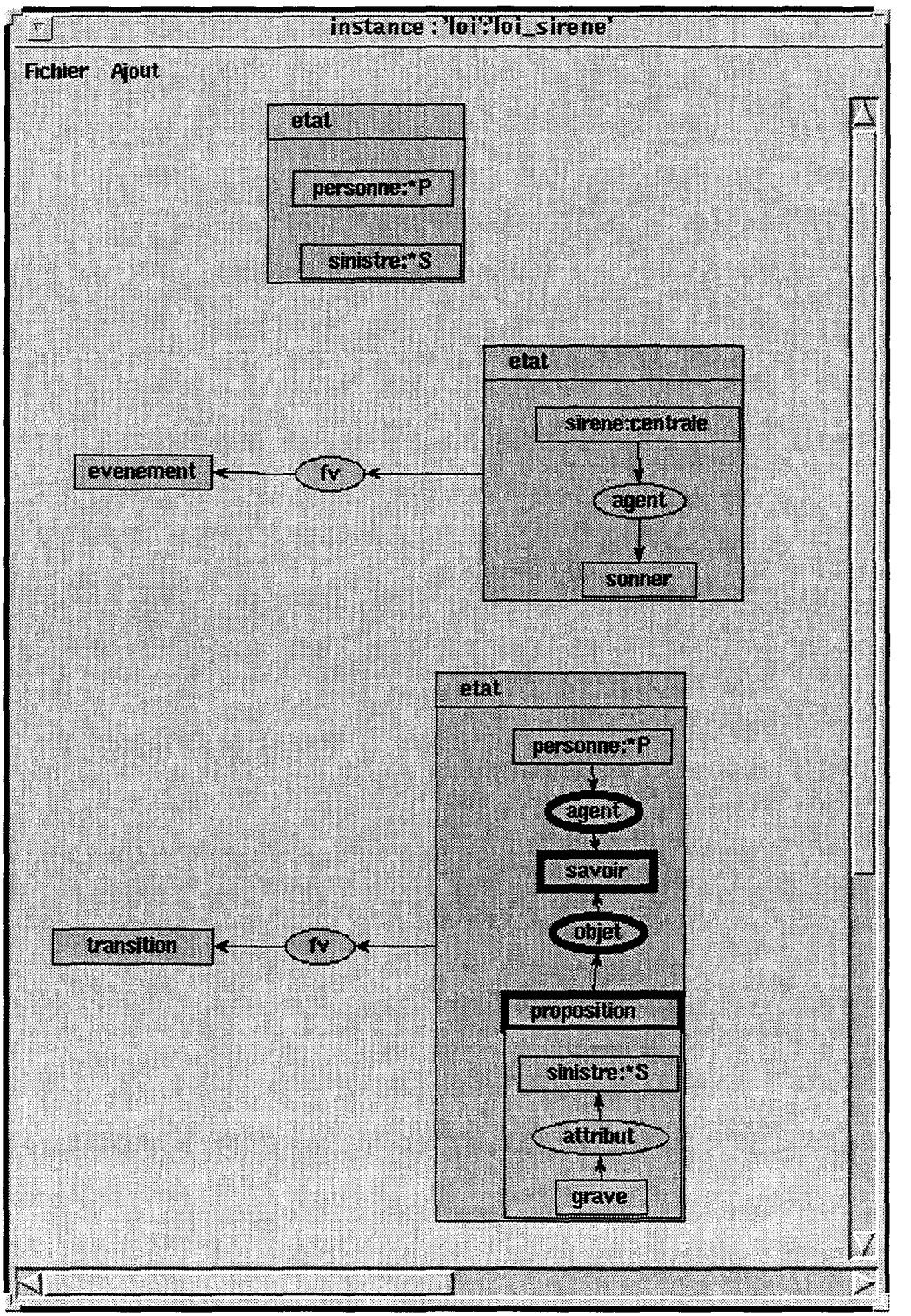
E.5. Modèles génériques d'actions



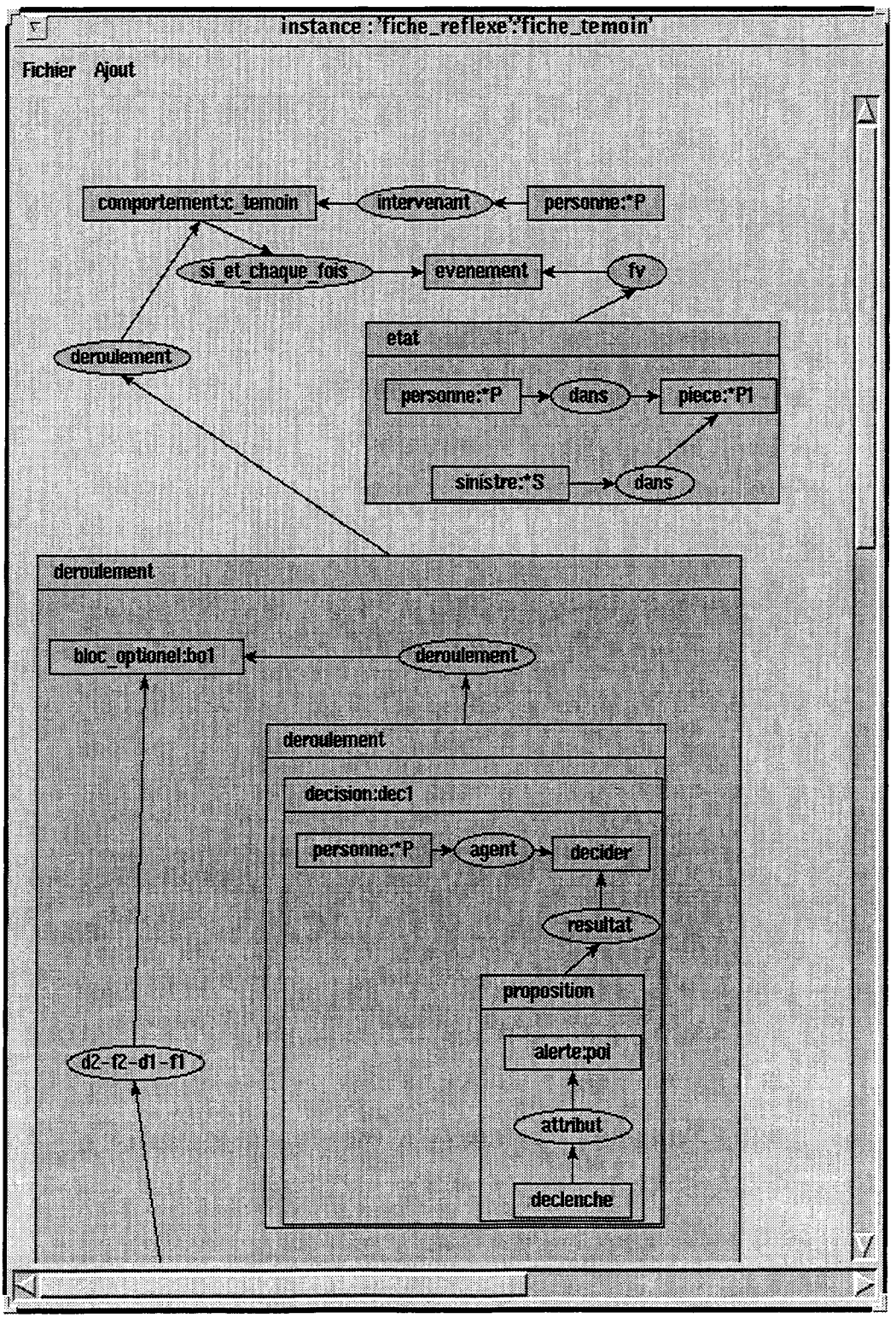

instance : 'modele_action':'aller_a_objet'
Fichier Ajout
deplacement
personne:*P
agent
aller
de
a
piece:*P1
piece:*P2
effet_debut
effet_fin
transition
transition
vf
precondition
fv
etat
personne:*P
dans
piece:*P1
etat
personne:*P
dans
piece:*P2
etat
personne:*P
dans
piece:*P1

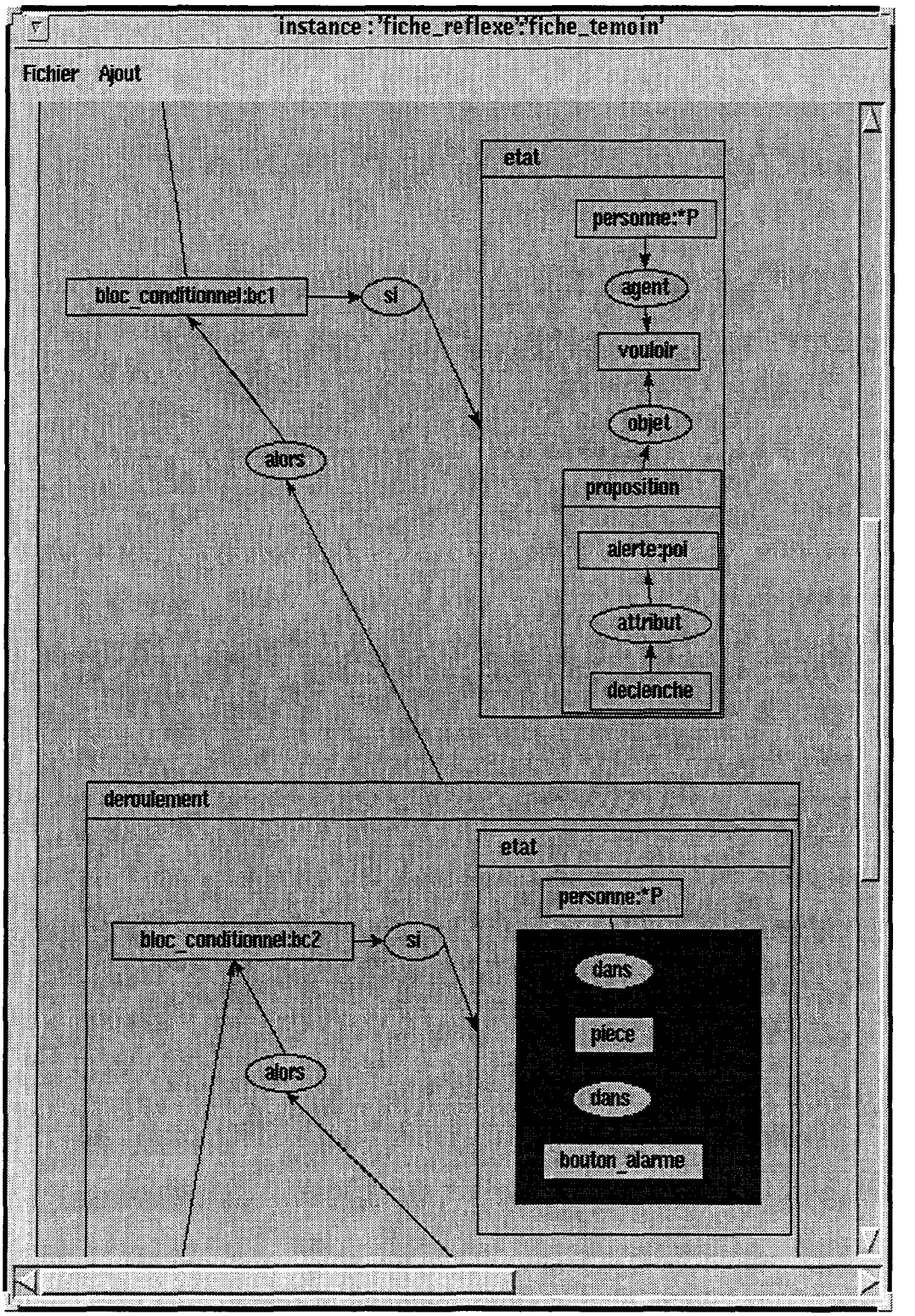

instance : 'modele_action':'declencher_alerte'
Fichier Ajout
communication
personne
agent
declencher
resultat
alerte
effet_fin
transition
fv
etat
sirene:centrale
agent
sonner

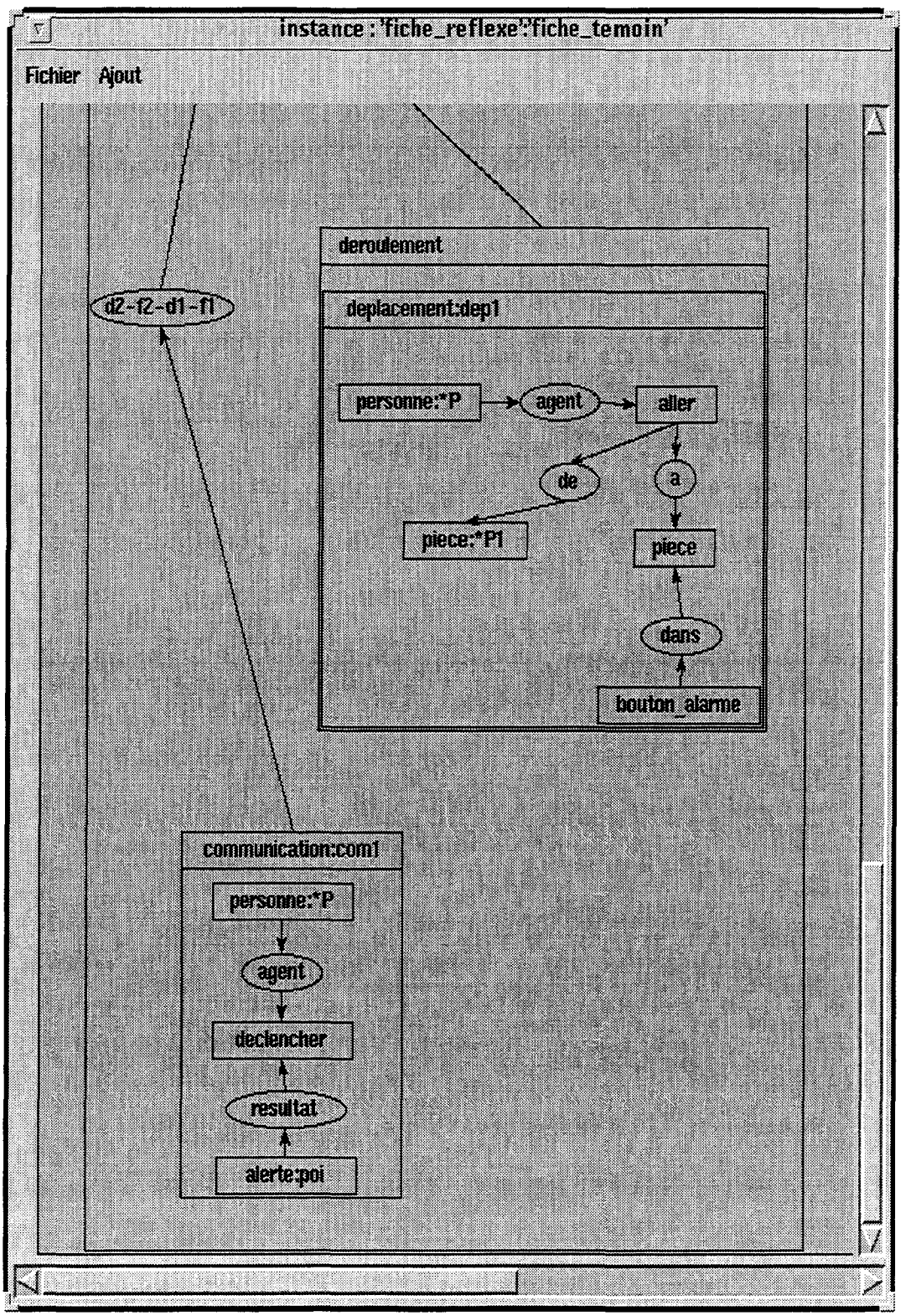
E.6. Lois dynamiques



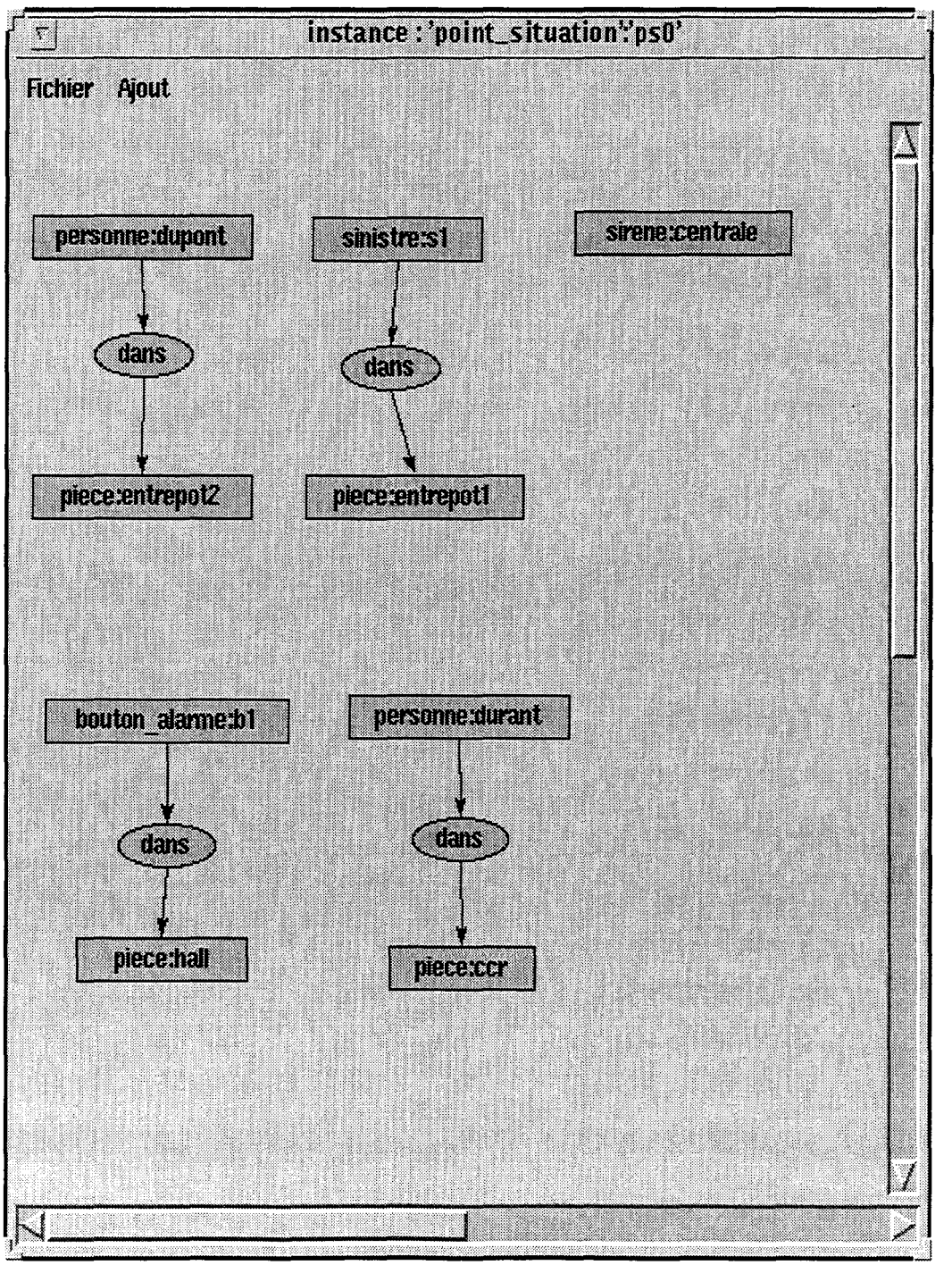
E.7. Comportement

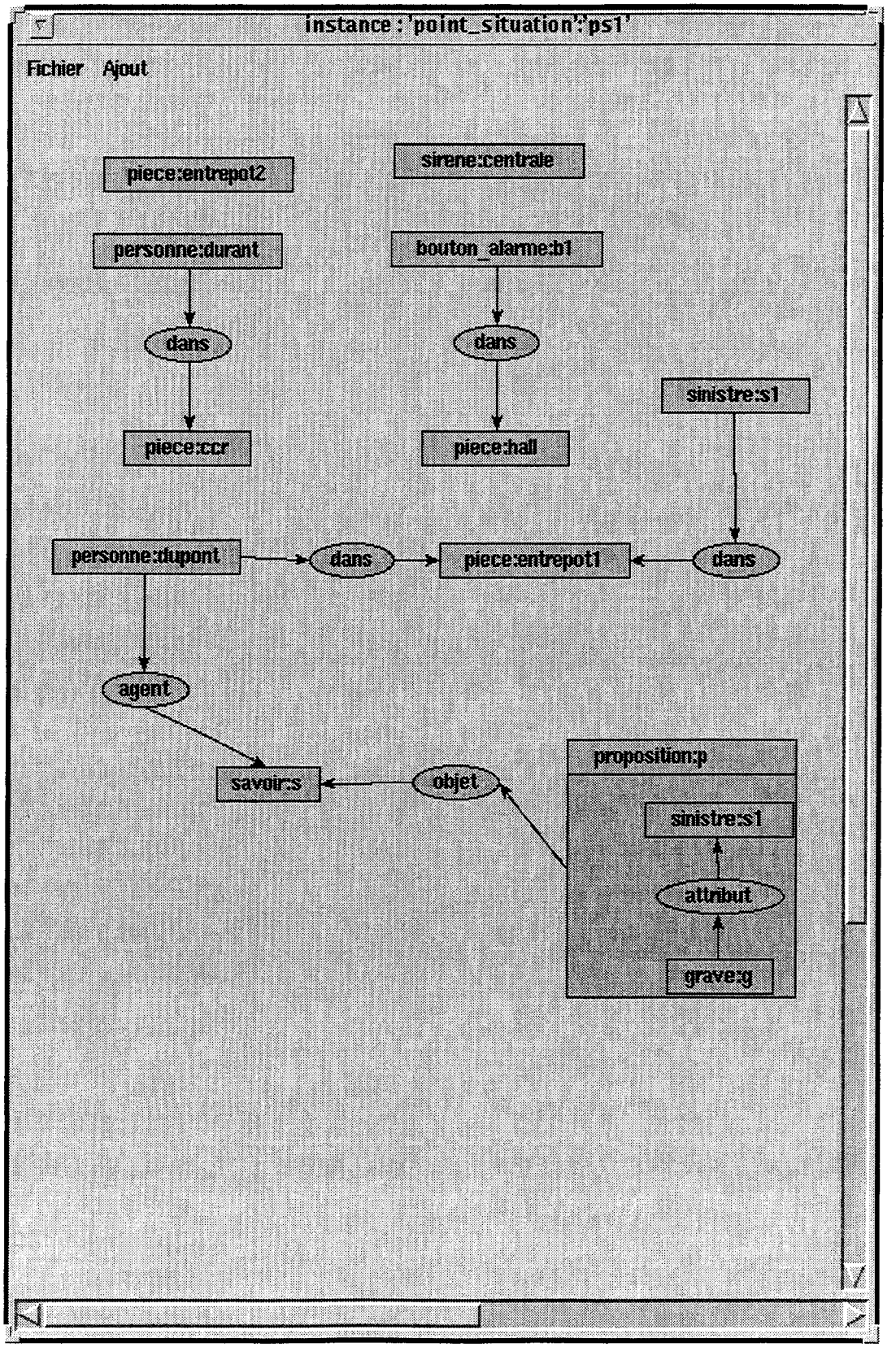



instance : 'fiche_reflexe':'fiche_temoin'
Fichier Ajout
etat
personne:*P
agent
vouloir
objet
proposition
alerte:poi
attribut
declenche
bloc_conditionnel:bc1
si
alors
deroulement
etat
personne:*P
bloc_conditionnel:bc2
si
alors
dans
piece
dans
bouton_alarme

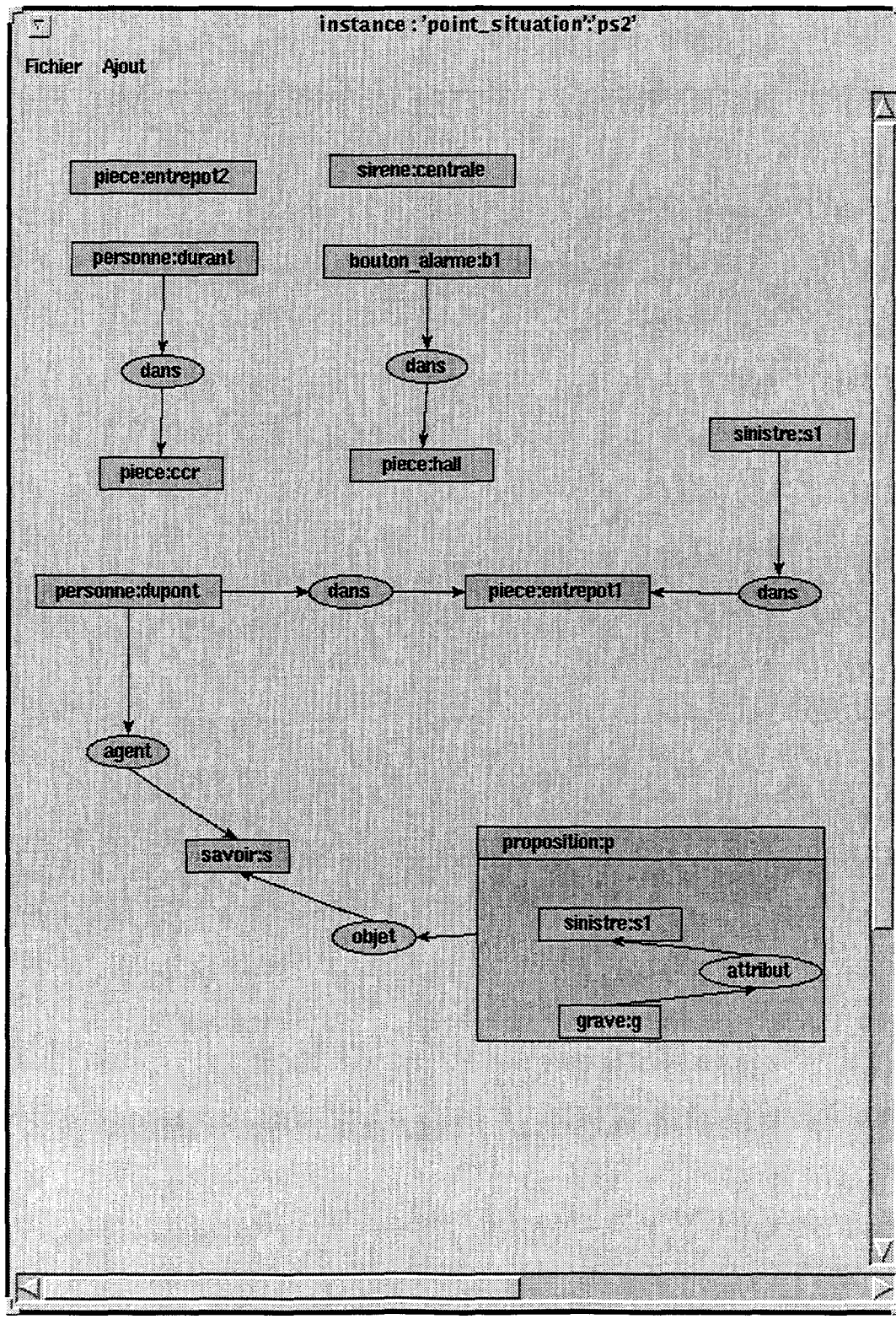

instance : 'fiche_reflexe':'fiche_temoin'
Fichier Ajout
d2-f2-d1-f1
deroulement
deplacement:dep1
personne:*P
agent
aller
de
a
piece:*P1
piece
dans
bouton_alarme
communication:com1
personne:*P
agent
declencher
resultat
alerte:poi

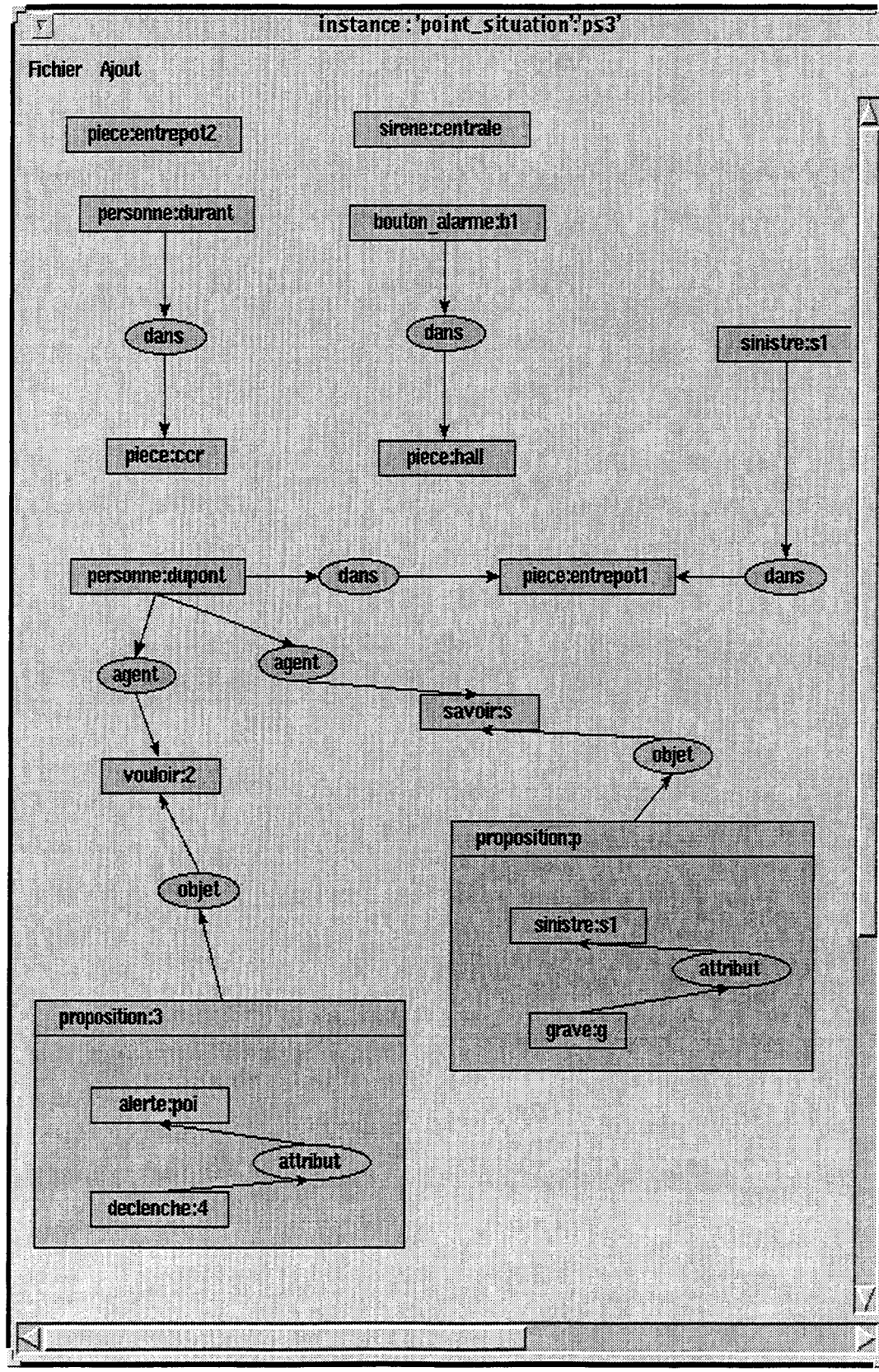
E.8. Scénario initial

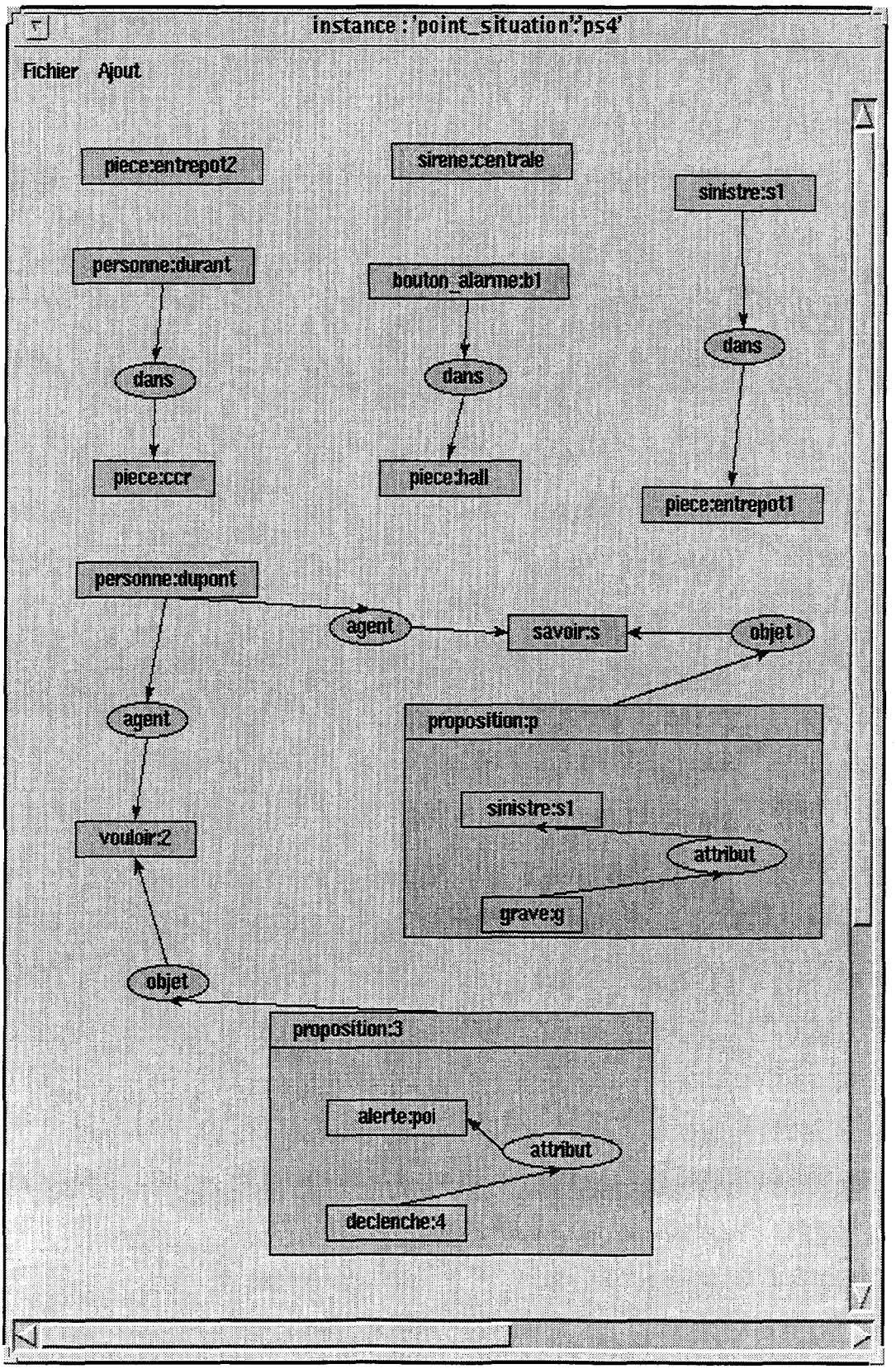


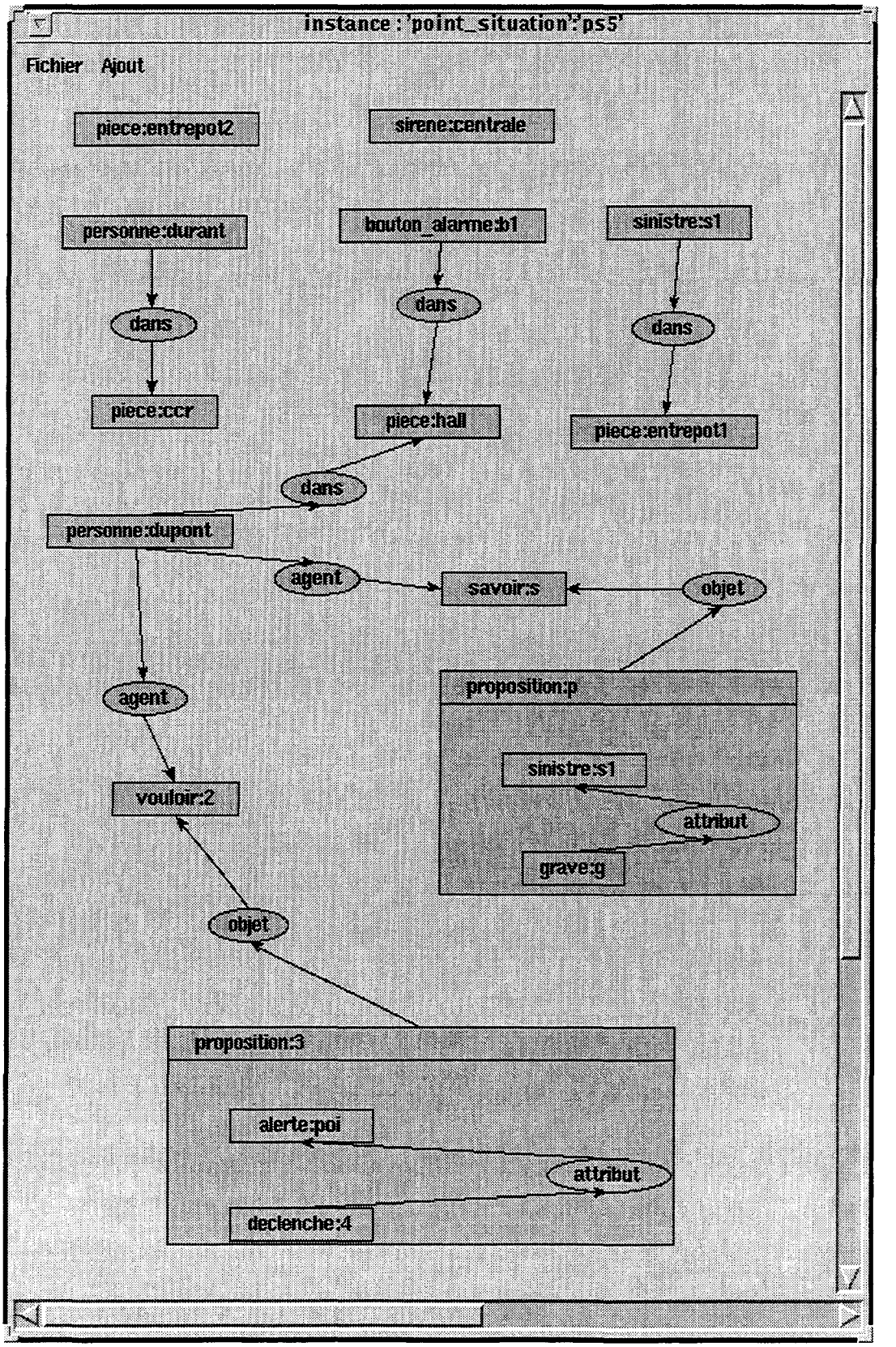

instance : 'point_situation':'ps1'
Fichier Ajout
piece:entrepot2
sirene:centrale
personne:durant
bouton_alarme:b1
dans
dans
sinistre:s1
piece:ccr
piece:hall
personne:dupont
dans
piece:entrepot1
dans
agent
savoir:s
objet
proposition:p
sinistre:s1
attribut
grave:g

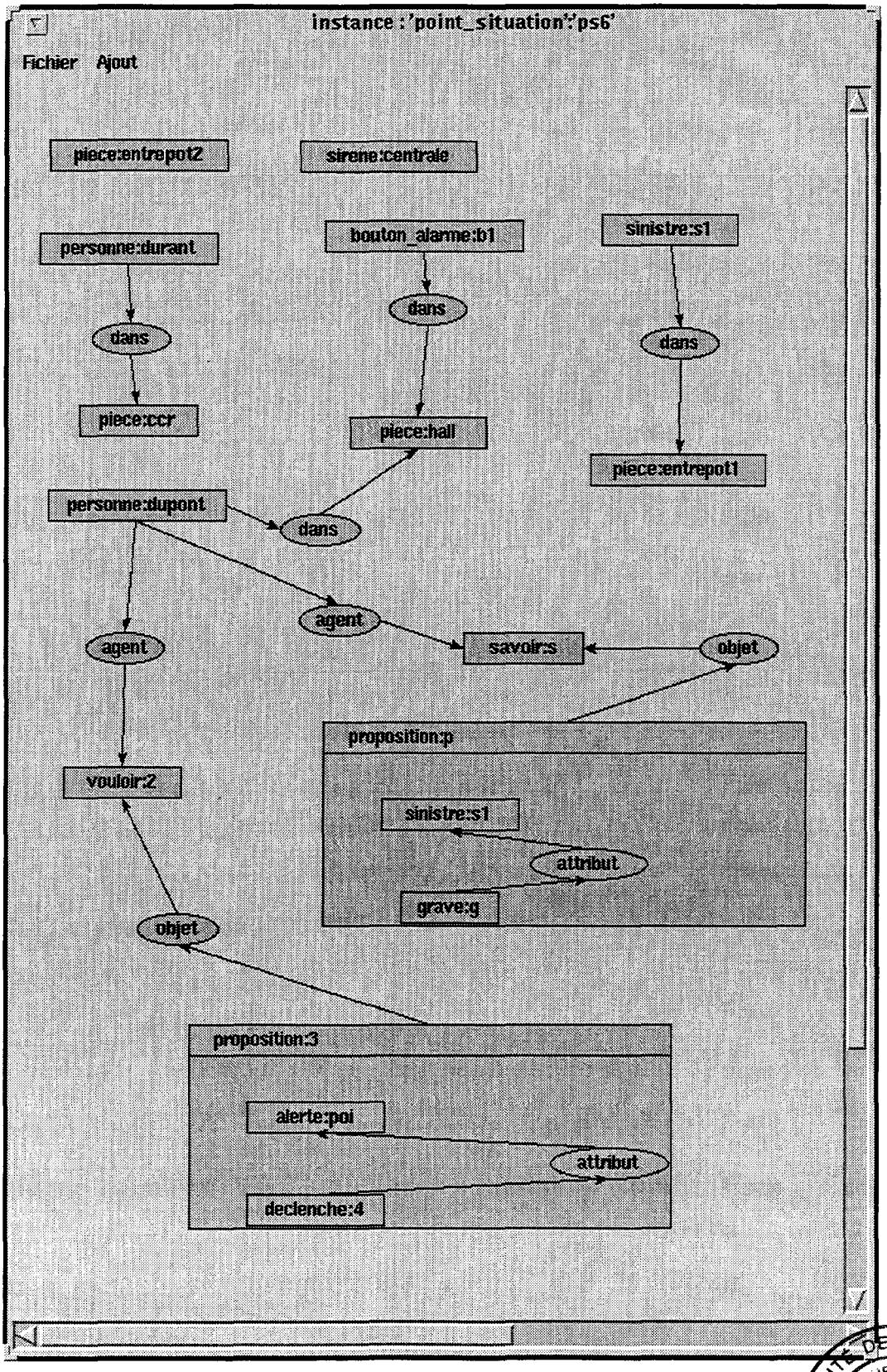
E.9. Résultats de simulation

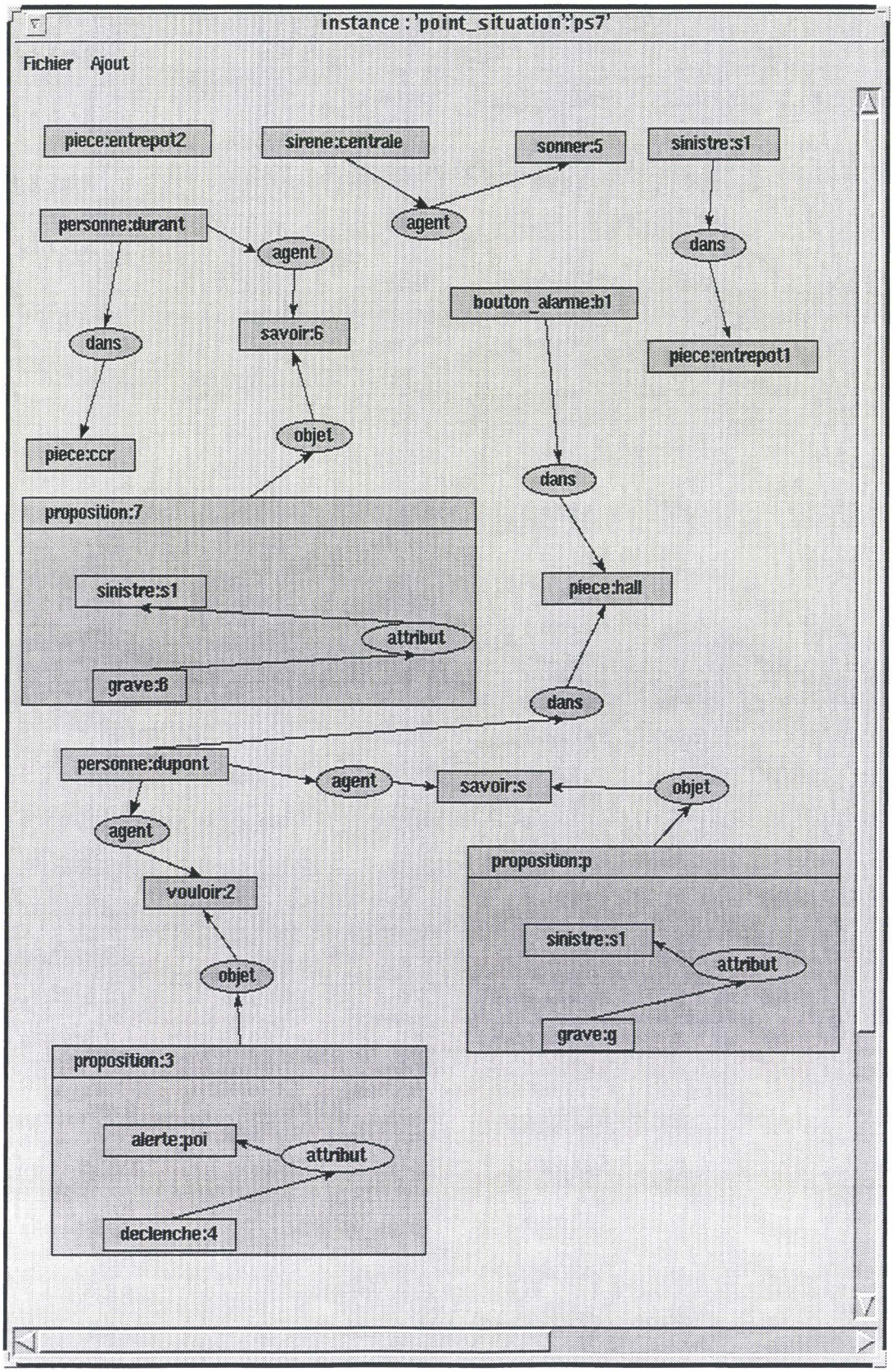



instance : 'point_situation':'ps3'
Fichier Ajout
piece:entrepot2
sirene:centrale
personne:durant
bouton_alarme:b1
dans
dans
sinistre:s1
piece:ccr
piece:hall
personne:dupont
dans
piece:entrepot1
dans
agent
agent
savoir:s
objet
vouloir:2
proposition:p
objet
sinistre:s1
attribut
grave:g
proposition:3
alerte:poi
attribut
declenche:4


instance : 'point_situation':'ps4'
Fichier Ajout
piece:entrepot2
sirene:centrale
sinistre:s1
personne:durant
bouton_alarme:b1
dans
dans
dans
piece:ccr
piece:hall
piece:entrepot1
personne:dupont
agent
savoir:s
objet
agent
proposition:p
sinistre:s1
vouloir:2
attribut
grave:g
objet
proposition:3
alerte:poi
attribut
declenche:4


instance : 'point_situation':'ps5'
Fichier Ajout
piece:entrepot2
sirene:centrale
personne:durant
bouton_alarme:b1
sinistre:s1
dans
dans
dans
piece:ccr
piece:hall
piece:entrepot1
dans
personne:dupont
agent
savoir:s
objet
agent
proposition:p
sinistre:s1
vouloir:2
attribut
grave:g
objet
proposition:3
alerte:poi
attribut
declenche:4




instance : 'point_situation':'ps7'
Fichier Ajout
piece:entrepot2
sirene:centrale
sonner:5
sinistre:s1
agent
personne:durant
agent
dans
bouton_alarme:b1
dans
savoir:6
piece:entrepot1
objet
piece:ccr
dans
proposition:7
sinistre:s1
piece:hall
attribut
grave:8
dans
personne:dupont
agent
savoir:s
objet
agent
proposition:p
vouloir:2
sinistre:s1
attribut
objet
grave:g
proposition:3
alerte:poi
attribut
declenche:4