N° d'ordre :

THÈSE

présentée à

L'UNIVERSITÉ DE NICE-SOPHIA ANTIPOLIS

pour obtenir le titre de

DOCTEUR EN SCIENCES

spécialité

INFORMATIQUE

par

Michel JACZYNSKI

Sujet de la thèse :

Modèle et plate-forme à objets pour l'indexation des cas par situations comportementales : application à l'assistance à la navigation sur le Web

Soutenue le 3 décembre 1998 devant le jury composé de :

M. Pierre Bernhard	Président
M. Jean-François Perrot	Rapporteur
M. Joël Quinqueton	Rapporteur
M. Alain Mille	Examinateur
M. Michel Plu	Invité
M. Michel Rueher	Examinateur
Mme Brigitte Trousse	Directeur

Remerciements

C’est avec une certaine émotion et beaucoup de sincérité que je voudrais remercier toutes les personnes ayant soutenu et apprécié mon travail.

En premier lieu, je remercie M. Pierre Bernhard de me faire l’honneur de présider le jury. Je le remercie également pour m’avoir accueilli dans l’unité de recherche de l’INRIA Sophia Antipolis. Je remercie M. Jean-François Perrot pour avoir accepté d’être rapporteur de ma thèse et pour m’avoir donné la possibilité de présenter mes travaux dans son équipe. Je remercie M. Joël Quinqueton, également rapporteur de cette thèse, pour ses commentaires qui ont permis d’améliorer mon rapport.

Je tiens tout particulièrement à remercier Mme Brigitte Trousse qui a dirigé cette thèse, pour sa disponibilité, son soutien et ses conseils qui ont été nombreux et importants. Mme Brigitte Trousse a beaucoup œuvré pour la mise en valeur de mon travail, et j’ai apprécié l’ambiance d’équipe qui s'est formée. J’ai beaucoup appris à ses côtés et je lui adresse toute ma reconnaissance.

Je remercie également M. Alain Mille pour sa participation au jury et pour son activité au sein du groupe RàPC français avec lequel j’ai passé des moments agréables dans différents lieux d’Europe. Je remercie M. Michel Plu du CNET pour son intérêt dans les applications de Broadway et pour sa participation au jury. Je remercie enfin M. Michel Rueher pour tous ses conseils au cours de ma thèse, sans oublier que j’ai eu le privilège d’être son élève.

Je remercie Rushed Kanawati, Marc Golinelli, Mourad Elhaddad et Youness Taheri pour leurs commentaires et leur participation à divers projets de l’équipe AID. Je remercie M. Brun de l’INRA Sophia Antipolis pour le temps qu’il m’a consacré. Je remercie également Mme Bonnardel et ses étudiants de l’université de Provence pour leur étude de Broadway.

La réalisation de ce travail s’appuie également sur un environnement qui est essentiel. A ce titre, je voudrais remercier l’INRIA et son personnel, notamment l’équipe SEMIR et l’équipe de la documentation. Je remercie également l’INRIA et la Région PACA pour leur participation financière dans cette étude.

Mes plus profonds remerciements vont à mes parents. Tout au long de mon cursus, ils m’ont toujours soutenu, encouragé et aidé. Ils ont su me donner toutes les chances pour réussir. Qu’ils trouvent, dans la réalisation de ce travail, l’aboutissement de leurs efforts ainsi que l’expression de ma plus affectueuse gratitude.

Je remercie ma famille et en particulier mon frère pour m’avoir fait partager leur joie de vivre et m’avoir ainsi soutenu dans mes efforts. Plus personnellement, je remercie ma fiancée, Anne-Marguerite de Boussineau, pour son aide, son écoute et surtout son amour qui m’a été essentiel durant ces années. Je remercie tous mes plus proches amis, en ayant une pensée pour Rémy à qui je souhaite santé et bonheur.

Je voudrais également remercier tous mes collègues et amis du projet Secoia. Je garderai un bon souvenir des discussions animées au cours des repas ensoleillés près de la piscine… Plus particulièrement, je remercie Stéphane Demphlous à qui je souhaite une très grande réussite personnelle. J'ai également une pensée amicale pour Wided Lejouad, dont je garderai l'exemple ainsi que le souvenir de nos discussions.

Je remercie enfin toutes les personnes intéressées par mon travail, en espérant qu'elles puissent trouver dans mon rapport des explications utiles pour leurs propres travaux.

à ma mère et à mon père

Table des matières

Remerciements 3

Table des matières 7

Introduction 15

1 Problématique 17

2 Contributions 18

2.1 Présentation des contributions 19

2.2 Evaluation expérimentale 20

3 Plans de lecture 20

Chapitre I : le RàPC pour une classe de problèmes 23

1 Motivations 25

1.1 Régulation de la nutrition des plantes 25

1.1.1 Problème de la nutrition 25

1.1.2 Vers la réutilisation d'expériences de régulation 26

1.2 Aide à la navigation sur le World-Wide-Web 27

1.2.1 Présentation du Web 28

1.2.2 Assistance à la navigation sur le Web 30

2 Présentation du raisonnement à partir de cas 31

2.1 Principes du raisonnement 32

2.1.1 Représentation des cas et des indices 32

2.1.2 Cycle du raisonnement 32

2.1.3 Importance de l'indexation pour le raisonnement et terminologie 34

2.2 Utilisation du raisonnement à partir de cas 35

2.2.1 Avantages, inconvénients et conditions d'application 35

2.2.2 Un exemple : le système CLAVIER 37

3 Application du RàPC à une classe de problèmes 38

3.1 Classe des problèmes nécessitant la prise en compte d'historiques 38

3.2 Intérêt et spécificité du RàPC 39

3.2.1 Justification et intérêt du raisonnement à partir de cas 39

3.2.2 Gestion d'indices sous forme d'historiques 39

3.3 Démarche générale de l'étude 40

3.3.1 Nécessité d'un modèle d'indexation générique 40

3.3.2 Besoin d'un outil logiciel ouvert pour la gestion du modèle et du RàPC 41

Chapitre II : modèle d'indexation par situations comportementales 43

1 Approches existantes d'indexation par historiques 45

1.1 Approches basées sur le raisonnement à partir de cas 45

1.1.1 Présentation des systèmes existants 45

1.1.1.1 Prévision de l'évolution des feux de forêts : le système REBECAS 46

1.1.1.2 Commande de robots mobiles : le système SINS 47

1.1.1.3 Autres systèmes et approches de RàPC 49

1.1.2 Limites des systèmes existants 52

1.1.2.1 Absence d'approche générale pour l'indexation 52

1.1.2.2 Gestion des cas insuffisante pour la découverte de connaissances 52

1.1.2.3 Absence d'indices basés sur des observations hybrides 55

1.2 Autres approches 55

1.2.1 Reconnaissance d'historiques 56

1.2.1.1 Approches de traitement du signal 56

1.2.1.2 Approches syntaxiques 57

1.2.1.3 Approches basées sur des scénarios temporels 59

1.2.2 Bases de données temporelles et gestion d'historiques 62
1.2.2.1 Systèmes de gestion de bases de chroniques 62
1.2.2.2 Le système TEMPOS 63
1.2.3 Limites de la reconnaissance d'historiques et des bases de données temporelles 64
2 Proposition du « modèle d'indexation par situations comportementales » 66
2.1 Modèle de représentation des indices et des cas 66
2.1.1 Hypothèses 66
2.1.2 Présentation générale 67
2.1.3 Description détaillée 68
2.1.3.1 Granularité des cas 68
2.1.3.2 Représentation des données d'observation 69
2.1.3.3 Représentation des situations comportementales 72
2.1.3.4 Représentation des cas 76
2.2 Guide d'utilisation de notre modèle de représentation 76
2.2.1 Principes de recherche 77
2.2.1.1 Types de filtrage pour la recherche 77
2.2.1.2 Exemple d'une stratégie séquentielle de recherche 78
2.2.2 Principes de réutilisation 79
2.2.3 Principes d'apprentissage 79
2.3 Un exemple : recherche des cas pour la nutrition des plantes 80
2.3.1 Identification des variables d'observation 80
2.3.2 Détermination de la sémantique d'un enregistrement et de son contexte 81
2.3.3 Définition de la représentation des cas 81
2.3.3.1 Cas cible 82
2.3.3.2 Cas potentiels 82
2.3.3.3 Cas concrets 82
2.3.4 Conception des phases de raisonnement 83
3 Conclusion 84

Chapitre III : plate-forme à objets pour la réalisation de systèmes de RàPC 87
1 Outils logiciels existants pour le RàPC 89
1.1 Présentation des outils 89
1.1.1 REMIND 90
1.1.2 REPRO 93
1.1.3 KATE 94
1.1.4 FABEL 94
1.1.5 Autres outils 96
1.2 Limites des outils existants 99
1.2.1 Difficulté de modifier les composants proposés 99
1.2.2 Difficulté d'intégrer de nouveaux composants 100
2 Proposition d'une plate-forme à objets pour le RàPC 100
2.1 Principes d'une plate-forme à objets 100
2.1.1 Définition et propriétés d'une plate-forme 100
2.1.2 Conception d'une plate-forme 101
2.1.3 Rôles des patrons de conception dans une plate-forme à objets 103
2.1.4 Utilisation d'une plate-forme 104
2.2 Choix d'une plate-forme comme outil réutilisable et ouvert 105
2.3 Analyse des points d'ouverture de notre plate-forme à objets 106
2.3.1 Points d'ouverture du RàPC 107
2.3.1.1 Gestion du raisonnement 107
2.3.1.2 Représentation des cas 108
2.3.1.3 Organisation de la mémoire 109
2.3.2 Points d'ouverture pour l'indexation par situations comportementales 110
2.3.2.1 Gestion du raisonnement 110
2.3.2.2 Représentation des cas 110
2.3.2.3 Organisation de la mémoire 112
3 Conclusion 112

Chapitre IV : patrons de conception et modèles à objets ouverts pour le RàPC 115

1 Modèles à objets ouverts pour le RàPC 117
1.1 Gestion du raisonnement 117
1.1.1 Patrons de conception et représentation du cycle du raisonnement 117
1.1.1.1 Contrôle du raisonnement 118
1.1.1.2 Cohérence de l'assemblage des phases 120
1.1.2 Schémas de base pour les phases du raisonnement 123
1.1.2.1 Schéma de configuration pour la recherche 124
1.1.2.2 Deux schémas pour la réutilisation 125
1.1.2.3 Schéma de décision de l'ajout d'un cas pour l'apprentissage 127
1.2 Représentation des cas 128
1.2.1 Cas et situation 128
1.2.2 Indices structurés et poids d'importance 129
1.3 Organisation de la mémoire 130
1.3.1 Indépendance des index et du stockage des cas 131
1.3.2 Bases de cas structurées : organisation logique et physique des cas 131
1.3.3 Index et stratégie de recherche 133
1.3.3.1 Index et filtrage des indices 133
1.3.3.2 Hiérarchie extensible d'index 133
1.3.3.3 Communication entre les index et cohérence de l'assemblage 136
1.3.4 Similarités structurées, modifiables et extensibles 138
1.3.4.1 Domaines de valeurs des similarités 138
1.3.4.2 Similarités structurées et modifiables 139
1.3.4.3 Hiérarchies extensibles de fonctions élémentaires de similarité et d'agrégation 140

2 Modèles à objets ouverts pour l'indexation par situations comportementales 142
2.1 Représentation des cas 143
2.1.1 Représentation des données d'observation 143
2.1.1.1 Chroniques 143
2.1.1.2 Enregistrements et curseurs 144
2.1.2 Représentation des cas et des situations comportementales 145
2.1.3 Représentation des patrons de cas potentiels 147
2.2 Organisation de la mémoire 148

3 Réalisation et utilisation de CBR*Tools 149
3.1 Réalisation de la plate-forme 149
3.1.1 Structuration en sous-systèmes 150
3.1.2 Implantation en Java 150
3.2 Utilisation de CBR*Tools 152
3.2.1 Environnement d'utilisation 152
3.2.2 Estimation de la complexité d'utilisation de CBR*Tools 153
3.2.2.1 Effort minimum 153
3.2.2.2 Dépendances systématiques entre les points d'ouverture 154

4 Conclusion 156

Chapitre V : application pour l'assistance à la navigation sur le Web 159

1 Présentation de Broadway 161
1.1 Contraintes de l'application 161
1.2 Interface utilisateur et architecture de Broadway 162
1.2.1 Interface utilisateur 162
1.2.1.1 Barre d'outils 162
1.2.1.2 Contrôleur 163
1.2.2 Architecture logicielle 164
1.2.2.1 Architecture globale 164
1.2.2.2 Architecture détaillée de Broadway 166
1.2.2.3 Broadway*Tools : des composants ouverts et réutilisables pour Broadway 167
1.3 Serveur de recommandations 169

2 Evaluation de notre modèle d'indexation et de CBR*Tools 170
2.1 Application du modèle d'indexation par situations comportementales 170

2.1.1 Identification des variables d'observation 170
2.1.1.1 Adresse d'une page Web 170
2.1.1.2 Contenu d'une page Web 172
2.1.1.3 Evaluation explicite des pages 174
2.1.1.4 Evaluation implicite des pages : le ratio d'affichage 174
2.1.2 Détermination de la sémantique d'un enregistrement et de son contexte 174
2.1.3 Définition de la représentation des cas 176
2.1.3.1 Cas cible 176
2.1.3.2 Cas potentiels et cas concrets 176
2.1.4 Conception des phases de raisonnement 178
2.1.4.1 Phase de recherche 178
2.1.4.2 Phase de réutilisation 181
2.1.4.3 Phase de révision 182
2.1.4.4 Phase d'apprentissage et maintenance de la base de cas 183
2.2 Utilisation de la plate-forme CBR*Tools 184
2.2.1 Gestion du raisonnement 184
2.2.2 Représentation des cas 186
2.2.3 Organisation de la mémoire 188
2.3 Bilan 190
2.3.1 Modèle d'indexation par situations comportementales 190
2.3.1.1 Généricité du modèle de représentation 190
2.3.1.2 Mise en évidence de cas : vers la découverte des connaissances 191
2.3.2 Plate-forme à objets CBR*Tools 191
2.3.2.1 Evaluation du degré d'expertise nécessaire de CBR*Tools pour Broadway 192
2.3.2.2 Evaluation de l'aide apportée par CBR*Tools pour la réalisation de Broadway 192
3 Proposition d'un nouveau type d'assistants à la navigation 193
3.1 Evaluation expérimentale de Broadway 193
3.1.1 Objectifs et procédure d'expérimentation 194
3.1.2 Résultats de l'expérimentation 194
3.1.3 Configuration de Broadway utilisée 195
3.2 Comparaison avec les systèmes existants d'aide à la navigation 197
3.2.1 Situation de recommandation 197
3.2.2 Apprentissage 200
3.2.3 Architecture : indépendance des navigateurs, assistance multi-sites et ouverture 200
4 Conclusion 201

Conclusion 203
1 Bilan 205
1.1 Modèle d'indexation par situations comportementales 205
1.2 Plate-forme à objets pour le RàPC 206
1.3 Broadway : un nouveau type d'assistants à la navigation sur le Web 206
2 Perspectives 207
2.1 Vers l'extension et la généralisation de notre modèle d'indexation 207
2.2 Vers des méthodes d'utilisation d'objets-métier pour le RàPC 208
2.3 Vers des assistants basés sur les comportements 209

Références bibliographiques 211
1 Liste des références 213
2 Index des références 224

Table des illustrations 227

Annexe A : éléments de notation UML 233
1 Diagramme de cas d'utilisation 235
2 Diagramme de classes 237
3 Diagramme d'objets 238

4 Diagramme de séquence 239

Annexe B : patrons de conception et CBR*tools 241

Annexe C : exemple d'utilisation de CBR*Tools 255

1 Présentation de l'application 257

2 Construction du système de raisonnement 258

2.1 Représentation des cas 258

2.2 Organisation de la mémoire 260

2.2.1 Base de cas 261

2.2.2 Mesure de similarité 261

2.2.3 Base d'index 262

2.3 Gestion du raisonnement 264

2.3.1 Recherche 265

2.3.2 Réutilisation 265

2.3.3 Apprentissage 265

2.3.4 Fabrique des objets 266

3 Exemples d'exécution 266

3.1 Premier raisonnement 266

3.2 Deuxième raisonnement 267

4 Code source complet de l'exemple 268

4.1 Classe CarCase 268

4.2 Classe CarCaseSituation 268

4.3 Classe CarFileSimpleCaseBase 270

4.4 Classe CarIndexBase 271

4.5 Classe CarReasonerFactory 273

4.6 Classe CarRetain 274

4.7 Classe CarRetrieve 274

4.8 Classe CarReuse 274

4.9 Classe CarsApp 275

4.10 Classe CarSimilarity 277

Annexe D : paramètres et algorithmes de Broadway 279

1 Paramètres de la modélisation 281

2 Algorithmes 284

2.1 Instanciation du patron de cas potentiels 284

2.2 Filtrage sur les comportements élémentaires 285

Introduction

1 Problématique

Notre travail a pour but l'étude du *raisonnement à partir de cas* (paradigme de résolution de problèmes par réutilisation d'expériences passées) pour une classe de problèmes complexes, ainsi que la conception d'outils logiciels pour faciliter la réalisation de systèmes intégrant ce type de raisonnement. Plus précisément, notre étude a été motivée par deux applications concrètes : tout d'abord la régulation de la nutrition des plantes, puis l'assistance à la navigation sur le World-Wide-Web.

La première application s'inscrit dans l'objectif général de réguler la nutrition minérale et hydrique des plantes, de manière à maîtriser les dérives minérales dans le substrat de culture et à assurer une meilleure adaptation de l'offre aux besoins des plantes. Le problème de régulation considéré est issu des travaux menés à l'INRA Sophia-Antipolis (Brun *et al.,* 1993 ; Brun & Settenbrino, 1993) : il s'agit de maintenir autour d'une consigne la concentration en sels minéraux de la solution de drainage qui est récupérée après l'arrosage des plantes. La commande, c'est-à-dire le paramètre contrôlé, est la concentration de la solution d'irrigation. Cette régulation est influencée par des perturbations dont certaines sont effectivement mesurées (température, rayonnement solaire, humidité). Bien que des approches de régulation aient été mises en place, ce problème reste ouvert notamment à cause de l'inertie du processus, du nombre important de variables pouvant entrer en jeu et de la nécessaire prise en compte des prévisions météorologiques. Une approche basée sur la réutilisation d'expériences passées doit permettre de reproduire les expériences de commande réussies, tout en évitant les échecs passés. De plus, cette approche doit permettre l'identification de comportements plus généraux, pour ainsi venir enrichir les connaissances du domaine.

Dans la deuxième application, nous désirons assister un groupe d'utilisateurs durant leurs navigations sur le World-Wide-Web (Web). Le Web est un vaste hypermédia qui est en expansion croissante, et offre une énorme quantité de données. Deux principales méthodes permettent de retrouver des documents pertinents dans cet espace virtuel : l'interrogation et la navigation. L'interrogation suppose l'utilisation d'un moteur de recherche et l'expression d'une requête. Toutefois, l'expression d'une requête n'est pas toujours facile si l'objectif de recherche est mal défini, et les réponses sont généralement accompagnées de beaucoup de bruit. La navigation est alors nécessaire et l'utilisateur peut parcourir à son gré les pages Web en suivant les liens proposés. Cependant, la complexité et l'étendue du Web peuvent entraîner la désorientation de l'utilisateur lors d'une navigation. La navigation est donc importante dans une tâche de recherche d'information sur le Web et nécessite une assistance. Notre approche vise alors une recherche d'information coopérative (de type indirect), où les expériences passées d'un groupe de personnes sont mises à profit. Plus précisément, nous désirons suivre un ensemble d'utilisateurs durant leurs navigations pour déduire leur intérêt potentiel à partir des comportements observés, et réutiliser des expériences passées similaires au sein du groupe pour proposer des recommandations.

Ces deux types d'applications, bien qu'issues de domaines différents, partagent des caractéristiques communes, ce qui nous a amenés à considérer la classe de problèmes définie par les critères suivants :

1. La prise en compte *d'historiques* est choisie ou nécessaire pour formuler le problème.
2. Un *processus dynamique* est observé.
3. Un ensemble *important* de données brutes est collecté.
4. Les données brutes peuvent être obtenues par *échantillonnage* et/ou lors *d'événements*.
5. Les connaissances du domaine sont *réduites* et une meilleure compréhension est recherchée.

Notre travail consiste alors à faciliter la conception et la réalisation de systèmes de raisonnement à partir de cas pour cette classe de problèmes. De manière générale, le raisonnement à partir de cas (RàPC) est une technique de résolution de problèmes basée sur la réutilisation par analogie d'expériences passées appelées *cas*. Un cas représente notamment un problème et la solution qui lui a été appliquée. Un cas est indexé pour permettre de le retrouver suivant certaines caractéristiques pertinentes et discriminantes. Ces caractéristiques, aussi appelées *indices*, déterminent dans quelle situation le cas peut être de nouveau réutilisé. Le raisonnement se décompose habituellement en quatre phases principales (Aamodt & Plaza, 1994) : la *recherche* des cas pertinents dans la mémoire, la *réutilisation* des solutions passées pour constituer une solution au problème courant, la *révision* de la solution pour l'affiner et enfin *l'apprentissage* de la nouvelle expérience pour améliorer le système et enrichir sa mémoire.

L'application du RàPC à des problèmes ayant ces caractéristiques présente une spécificité qui touche principalement la gestion des indices des cas sous la forme d'historiques. Peu de travaux en RàPC ont abordé cette spécificité : prévision de l'évolution de feux de forêt (Rougegrez, 1994a), prévision économique (Nakhaeizadeh, 1994), prévision médicale et épidémiologique (Schmidt *et al.,* 1996 ; Bull *et al.,* 1997), commande de robots mobiles (Ram & Santamaria, 1997), aide à la supervision de processus (Fuchs *et al.,* 1995), et assistance à la navigation sur le Web (Corvaisier *et al.*, 1998). De plus, la plupart de ces approches sont dédiées à une application particulière, et aucun de ces travaux ne vise une classe de problèmes aussi générale, ni ne fournit des outils réutilisables. Enfin, les approches existantes ne permettent pas une gestion des cas soutenant l'effort de découverte des connaissances, nécessaire dans ces problèmes où les connaissances initiales sont réduites.

Nous nous fixons alors deux objectifs principaux :

- la définition d'un *modèle générique d'indexation des cas* adapté à la classe de problèmes visée,
- et la réalisation d'un *outil logiciel ouvert* pour faciliter son utilisation.

En définitive, nous abordons trois des problèmes importants soulevés par Janet Kolodner, un pionnier dans le domaine du raisonnement à partir de cas, qui citait parmi les objectifs de recherche du domaine (Kolodner, 1993, pages 571–576) : « l'extraction de cas depuis des situations continues » et la « définition d'un vocabulaire général d'indexation » qui doivent être considérées dans le modèle d'indexation, ainsi que la conception « d'outils pour la réalisation de systèmes RàPC ».

2 Contributions

Les deux premières contributions de notre étude sont :

- La proposition d'un modèle d'indexation générique, appelé *modèle d'indexation par situations comportementales*, pour faciliter l'application du RàPC au sein de la classe de problèmes visée.
- La proposition d'un nouveau type d'outils logiciels pour le RàPC à travers la conception et la réalisation d'une plate-forme à objets (*object-oriented application framework*), nommée CBR*Tools. Cette plate-forme repose sur des points d'ouverture et des modèles à objets exprimés systématiquement en termes de patrons de conception (*design patterns*). Cet outil facilite l'utilisation de notre modèle d'indexation et son intégration dans des systèmes de RàPC.

Pour évaluer ces deux contributions, nous avons principalement réalisé un *système d'aide à la navigation sur le Web*, nommé Broadway. De par la nature de notre modèle d'indexation, l'approche concrétisée dans Broadway est originale. Nous présentons ainsi une troisième contribution, avec la

proposition d'un nouveau type d'assistants à la navigation sur le Web, basé sur le calcul de recommandations par réutilisation de comportements observés à partir d'un ensemble extensible de variables.

2.1 Présentation des contributions

Premièrement, nous proposons pour la classe de problèmes visée, une approche générale de l'indexation alliant un modèle de représentation générique avec un guide d'utilisation présentant différents principes de gestion des indices dans un cycle de raisonnement à partir de cas. Ces principes donnent des exemples de types de traitements identifiés et analysés lors de l'étude des applications réalisées. Le modèle de représentation générique permet de séparer les données brutes (structurées en variables et en enregistrements) des cas qui référencent ces données à travers des situations comportementales. Une situation comportementale est principalement formée de comportements élémentaires (séquences issues d'historiques) reliés par des contraintes temporelles. Ce modèle permet d'une part de représenter des indices sur des historiques échantillonnés et/ou à événements, et d'autre part de manipuler les cas pour mener à la découverte des connaissances. En effet, l'introduction du concept de patron de cas potentiels permet d'extraire, suivant des hypothèses préétablies, des enseignements structurés à partir des données d'observation. Ces extractions peuvent être stockées sous forme de cas concrets dont les représentations peuvent être modifiées suivant les raisonnements. De plus, ces cas référencent toujours les données brutes d'observation ce qui permet de faciliter leur interprétation.

Nous proposons deuxièmement un nouveau type d'outils logiciels dans le domaine du RàPC tirant profit des concepts de plate-forme à objets (*object-oriented application framework*), de patron de conception (*design pattern*) et de cas d'utilisation (*use case*). Nous proposons tout d'abord une description de haut niveau, sous la forme de cas d'utilisation liant les tâches de conception d'un système de RàPC vers les points d'ouverture nécessaires à l'application du RàPC et de notre modèle d'indexation. Ces points d'ouverture sont intégrés dans une architecture abstraite au sein d'une plate-forme à objets dont la modélisation s'appuie systématiquement sur l'utilisation de patrons de conception. Puis, la plate-forme réalisée, appelée CBR*Tools, fournit également un ensemble de classes concrètes pouvant être directement utilisées (index, similarité, base de cas). Cette plate-forme facilite la spécialisation concrète de notre modèle d'indexation et son intégration avec des méthodes spécifiques de recherche, d'adaptation, de révision et d'apprentissage nécessaires à la réalisation d'une application. Notre approche permet également la capitalisation d'objets-métier réutilisables sous la forme, par exemple, d'index, de fonction de similarité et de schémas de phase de raisonnement.

Enfin, nous proposons un nouveau type d'assistants à la navigation sur le Web dont le prototype Broadway[1] est un premier exemple. Cette assistance est basée sur la réutilisation des navigations passées d'un groupe d'utilisateurs dont les comportements sont observés suivant un ensemble extensible de variables. Dans Broadway, quatre variables sont utilisées : adresse, contenu, évaluation explicite, et évaluation implicite. Cette observation par variables, combinée avec notre modèle d'indexation, permet la gestion de comportements détaillés dans un cadre flexible et générique.

Nos trois contributions ont fait l'objet de sept publications scientifiques nationales et internationales : modèle d'indexation par situations comportementales (Jaczynski, 1997 ; Jaczynski & Trousse, 1997a), plate-forme à objets pour le RàPC (Jaczynski & Trousse, 1999 ; Jaczynski &

[1] Browsing advisor reusing pathways.

Trousse, 1998a) et le système Broadway (Jaczynski & Trousse, 1998b ; Jaczynski & Trousse, 1998c ; Jaczynski & Trousse, 1997b).

2.2 Evaluation expérimentale

L'évaluation de notre modèle d'indexation a été réalisée dans le cadre de la nutrition des plantes et dans celui de l'assistance à la navigation sur le Web. Plus précisément, la première application nous a permis de vérifier que les caractéristiques du processus de nutrition peuvent être prises en compte par le modèle de représentation. Un prototype de recherche des cas a été réalisé en se basant sur des données brutes enregistrées manuellement par R. Brun, chercheur de l'INRA Sophia Antipolis. Toutefois, la spécification et la validation d'un système de régulation avec une approche de RàPC demandent une mise en opération effective qui n'a pas été effectuée. En effet, la réalisation d'un tel système requiert des compétences métier et une lourde disponibilité des experts du domaine. C'est pourquoi nous avons retenu le domaine de la navigation sur le Web : les données réelles sont plus facilement disponibles et nous avons été plus autonomes pour concevoir et tester l'application de notre modèle d'indexation. Nous avons alors réalisé le système d'assistance Broadway qui montre l'utilisation opérationnelle de notre modèle d'indexation ainsi que son intégration dans un cycle complet de raisonnement à partir de cas.

L'approche de plate-forme à objets pour le RàPC et les modèles à objets proposés ont mené à la réalisation concrète de CBR*Tools comprenant plus de 200 classes programmées dans le langage Java et conçues sous l'atelier de génie logiciel Rational Rose (Rose, 1998). En plus des deux applications précédentes réalisées avec CBR*Tools, nous avons également implanté un système classique de RàPC permettant l'affectation d'un facteur de risque à une voiture pour un assureur. Ce système n'utilise pas notre modèle d'indexation par situations comportementales car il avait uniquement pour but de valider l'architecture et les composants de base.

Notre assistant de navigation Broadway fut évalué lors d'une expérimentation qui a été conçue et supervisée par des étudiants en DESS d'ergonomie (Hébraud *et al.*, 1998). Cette expérimentation a mis en jeu dix utilisateurs réels (étudiants en psychologie n'ayant pas de connexion avec nos travaux) et a nécessité l'utilisation continue de deux systèmes Broadway pendant dix jours. Les résultats de cette expérimentation montrent que l'approche est bénéfique dans le cadre restreint choisi.

3 Plans de lecture

Le document est organisé en cinq chapitres et permet de développer notre approche visant des modèles et des outils pour l'application du RàPC à une classe de problèmes. Un aperçu rapide de la thèse peut être obtenu en complétant la lecture de l'introduction et de la conclusion de la thèse, par les résumés placés au début de chaque chapitre.

Nous conseillons à tout lecteur de commencer par le chapitre I qui présente plus précisément nos motivations et notre démarche pour l'application du raisonnement à partir de cas à la classe des problèmes nécessitant la prise en compte d'historiques. Le raisonnement à partir de cas et la notion d'indexation sont également introduits.

Dans le cadre d'une lecture complète, nous proposons ensuite, dans le chapitre II, notre modèle d'indexation des cas par situations comportementales, après avoir mené un état de l'art des approches existantes d'indexation par historiques. Nous décrivons également l'utilisation du modèle d'indexation dans le cadre de la nutrition des plantes. Dans le chapitre III, nous menons un état de l'art des outils

logiciels pour le RàPC et nous présentons le concept de plate-forme à objets qui nous a permis de dépasser leurs limites en terme d'ouverture. Nous analysons alors les points d'ouverture nécessaires pour le raisonnement à partir de cas et plus spécifiquement pour l'indexation par situations comportementales. Dans le chapitre IV, nous présentons l'utilisation des patrons de conception pour la modélisation ouverte du cycle du RàPC et de l'indexation par situations comportementales. Nous introduisons alors la plate-forme CBR*Tools que nous avons réalisée. Enfin, dans le chapitre V, nous détaillons la conception de Broadway, notre assistant à la navigation sur le Web qui met en œuvre le modèle d'indexation par situations comportementales ainsi que la plate-forme CBR*Tools.

Un lecteur intéressé par le raisonnement à partir de cas et par l'indexation par situations comportementales pourra directement lire la proposition du modèle (cf. chapitre II, §2 et §3), les points d'ouverture associés de notre plate-forme (cf. chapitre III, §2.3), le cycle de raisonnement de Broadway (cf. chapitre V, §1.3 et §2.1) et l'évaluation du modèle dans le cadre de Broadway (cf. chapitre V, §2.3.1).

Un lecteur intéressé par notre plate-forme à objets pour la réalisation de systèmes de RàPC pourra directement lire la proposition de cette approche (cf. chapitre III, §2) et les modèles à objets associés (cf. chapitre IV). La description du niveau spécifique pour l'indexation par situations comportementales, présentée séparément (cf. chapitre III, §2.3.2 ; chapitre IV, §2), repose sur les concepts du modèle d'indexation qui peuvent être rapidement cernés (cf. chapitre II, §2.1.2 et §2.2). Nous décrivons également un exemple d'utilisation de la plate-forme pour Broadway ainsi que son évaluation (cf. chapitre V, §2.2 et §2.3.2).

Un lecteur intéressé par notre application Broadway pourra se reporter à la présentation du système (cf. chapitre V, §1) et à l'évaluation de notre approche (cf. chapitre V, §3). Cependant, la représentation des cas et des indices ainsi que la réalisation (cf. chapitre V, §2) s'appuient sur les notions introduites précédemment (cf. chapitre II, §2 ; chapitre III, §2.3 ; chapitre IV, §1 et §2).

Nous fournissons également plusieurs annexes pour compléter notre étude. L'annexe A décrit les éléments de la notation UML (*Unified Modelling Language*) utilisée dans les chapitres III, IV et V. L'annexe B présente les patrons de conception (*design patterns*) utilisés dans CBR*Tools. L'annexe C décrit un exemple complet de l'utilisation de CBR*Tools dans le cadre du calcul du facteur de risque d'une voiture pour un assureur. Et enfin, l'annexe D liste principalement les paramètres de Broadway permettant de modifier le cycle du raisonnement suivi.

Chapitre I : le RàPC pour une classe de problèmes

Résumé

Notre étude a été motivée par deux applications : tout d'abord la régulation de la nutrition de plantes, puis l'assistance à la navigation sur le World-Wide-Web. Ces deux applications, bien qu'issues de domaines différents, partagent des contraintes communes, ce qui nous a amenés à considérer une *classe de problèmes* plus générale dans laquelle la prise en compte d'historiques est choisie ou nécessaire pour formuler le problème. Notre objectif est de faciliter l'utilisation du raisonnement à partir de cas pour cette classe de problèmes. Dans ce but, il est premièrement nécessaire de définir un *modèle générique d'indexation des cas* permettant la gestion d'historiques. Ce modèle doit fournir un formalisme générique de représentation des indices et doit permettre le développement d'une compétence méthodologique accumulée à travers l'analyse de différentes applications. Deuxièmement, il s'agit également de concevoir un *outil ouvert* facilitant l'utilisation pratique de ce modèle pour la réalisation de systèmes de raisonnement à partir de cas. Cet outil doit permettre la spécialisation du modèle pour prendre en compte les caractéristiques spécifiques d'une application. Il doit également faciliter son intégration cohérente dans un système complet de RàPC. Enfin, cet outil doit permettre la capitalisation de composants pour enrichir, au cours des développements, l'ensemble des objets-métier réutilisables.

Dans ce premier chapitre, nous décrivons tout d'abord le contexte des deux applications qui ont motivé notre étude : la régulation de la nutrition des plantes et l'assistance à la navigation sur le World-Wide-Web. La première application servira de cadre pour un premier exemple de notre approche dans le chapitre suivant. La deuxième application sera détaillée dans le chapitre V. Nous présentons ensuite les principes du raisonnement à partir de cas en introduisant notamment les étapes du raisonnement et les conditions de son application. Enfin, nous expliquons notre démarche et les objectifs qui ont guidé toute notre étude et la rédaction de la thèse.

1 Motivations

La classe de problèmes abordée dans cette thèse est principalement motivée par l'étude de deux applications : la régulation de la nutrition des plantes basée sur les travaux des chercheurs de l'INRA[2] de Sophia-Antipolis, et l'assistance à la navigation sur le World-Wide-Web. Nous présentons ces deux applications en montrant l'intérêt d'adopter une approche basée sur la réutilisation d'expériences passées.

1.1 Régulation de la nutrition des plantes

Nous donnons, dans un premier temps, le contexte général de la nutrition des plantes, puis nous présentons plus particulièrement le problème de régulation que nous avons étudié et l'approche que nous visons.

1.1.1 Problème de la nutrition

Dans le cadre donné par l'INRA (Brun *et al.,* 1993 ; Brun & Settenbrino, 1993), l'objectif général est de *réguler* la nutrition minérale et hydrique du rosier en culture hors sol de manière à maîtriser les dérives minérales dans le substrat de culture. Plus précisément, il s'agit d'assurer une meilleure adaptation de l'offre aux besoins de la plante (pour éviter les carences et les stress[3]) tout en limitant les rejets dans l'environnement. La régulation d'un processus dynamique, comme celui de la nutrition, consiste à déterminer les valeurs d'une ou plusieurs variables contrôlées du système de manière à optimiser un critère d'évaluation dépendant d'une *consigne*. La technique utilisée de culture hors sol sous serre permet, d'une part, de contrôler à l'aide d'ordinateurs certains paramètres (nutrition minérale et hydrique, température) et d'autre part, de recueillir régulièrement, grâce à des capteurs ou des mesures manuelles, un ensemble de données issues de la plante et de son environnement (par exemple le diamètre des tiges, la température, l'humidité, le rayonnement solaire, la concentration et le volume du drainage).

Ce problème général de régulation a été décomposé par les chercheurs de l'INRA en sous-problèmes à des échelles de temps différentes. Trois niveaux de régulation sont identifiés (Baille *et al.,* 1990 ; Baille, 1991) : contrôle du microclimat effectué presque de manière instantanée, contrôle de la croissance de la culture avec des décisions journalières, et contrôle de la croissance à long terme où l'unité de temps est le mois. Au premier niveau, nous notons deux régulateurs importants, l'un chargé de l'irrigation et l'autre de la température. Le premier régulateur permet de calculer le volume de la solution nutritive de manière à maintenir dans des limites fixées le volume du drainage (20 à

[2] Institut National de Recherche en Agronomie.
[3] Toutefois, un stress hydrique appliqué à un stade déterminé de la croissance est parfois favorable pour le rendement.

30% du volume de l'irrigation) : l'irrigation de doses de concentration constante est déclenchée en relation avec le rayonnement global. La température à l'intérieur de la serre est également régulée dans une certaine limite : un chauffage peut être déclenché la nuit lorsque la température passe en dessous d'un seuil (14º), et si la température dépasse un autre seuil durant la journée (26º), l'ouverture du toit de la serre et l'activation de la ventilation permettent de la faire baisser.

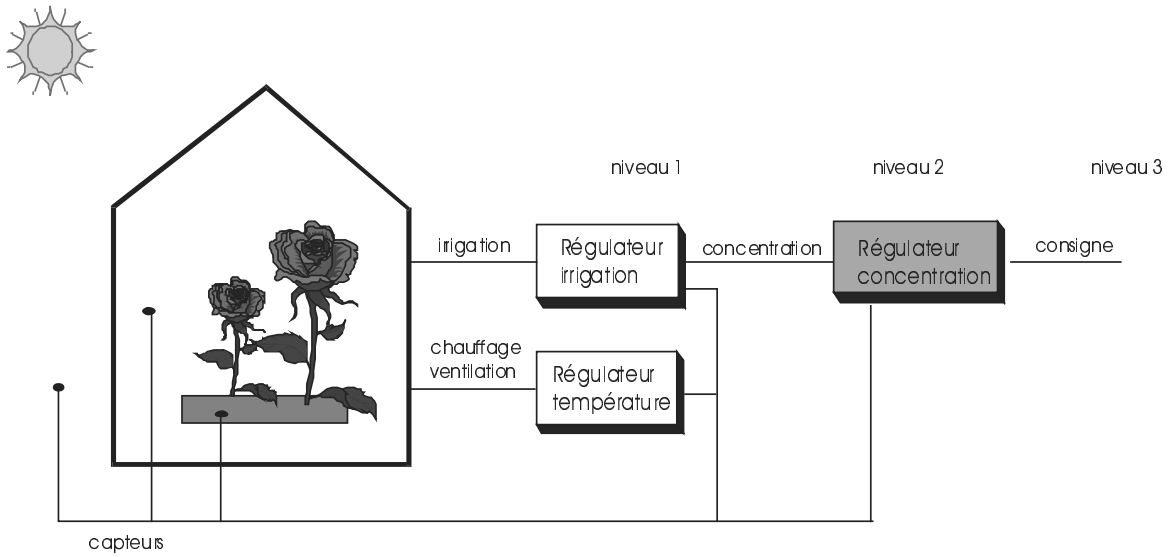


Figure I-1 : régulateur de concentration

Nous nous sommes intéressés plus particulièrement à un problème de régulation de la concentration journalière de drainage (Brun *et al.,* 1993 ; Brun & Settenbrino, 1993). Dans ce régulateur, situé au deuxième niveau (cf. Figure I-1), il s'agit de maintenir autour d'une consigne (déterminée au troisième niveau) la concentration en sels minéraux de la solution de drainage qui est récupérée après l'arrosage des plantes. La commande, c'est-à-dire le paramètre contrôlé, est la concentration de la solution d'irrigation. Les concentrations sont évaluées grâce à l'électro-conductivité des solutions ; la consigne est ainsi déterminée *a priori*, d'une part à partir des besoins estimés pour une bonne croissance de la plante qui donne une certaine électro-conductivité, et d'autre part à partir de l'électro-conductivité de l'eau servant à préparer la solution. Cette régulation est influencée par des perturbations dont certaines sont effectivement mesurées sur la base de la journée (température, rayonnement solaire, humidité).

1.1.2 Vers la réutilisation d'expériences de régulation

Pour mieux comprendre le fonctionnement du processus de nutrition, nous avons dressé les principales relations qualitatives entre les différents paramètres à partir des discussions que nous avons eues avec M. Brun, chercheur de l'INRA. La plante reçoit un rayonnement et produit de la vapeur d'eau par transpiration. Au niveau des racines, la plante absorbe de manière sélective de l'eau et différents sels minéraux. Ainsi, plus le rayonnement est fort, plus la transpiration est forte et la plante absorbe plus d'eau que de sels minéraux. Dans ce cas, la concentration de l'apport à l'intérieur de la plante est faible, « elle consomme plus dilué ». Par contre, si le rayonnement est faible, la transpiration est faible et le besoin en eau également, « la plante consomme plus concentré ». Connaissant ce phénomène, il est inutile de trop concentrer l'apport lors d'un rayonnement fort, car beaucoup de sels minéraux se retrouveraient dans le drainage. Les besoins en eau sont évalués par une formule prenant en compte le rayonnement et parfois l'humidité : plus le rayonnement est fort, plus l'espacement entre les doses est faible, et plus l'apport journalier est grand.

Le processus est également caractérisé par une inertie : la concentration de l'apport dans les racines ne peut pas être changée immédiatement. En effet, il s'écoule un certain temps entre le changement de concentration de l'irrigation et celle dans le milieu racinaire. Cette inertie peut être évaluée en mesurant le temps entre une modification importante de la concentration de l'irrigation qui correspond à un certain degré d'électro-conductivité et à sa réponse au niveau de l'électro-conductivité du drainage. Des expériences montrent que ce retard dépend de l'ordre de grandeur des concentrations mises en jeu et du volume de l'irrigation : plus la concentration est grande, plus le retard est grand, et d'un autre côté lorsque l'irrigation est plus forte (durant l'été), l'inertie baisse. Ainsi, cette inertie varie de 2 à 5 jours suivant les paramètres climatiques (Brun *et al.,* 1993). Il est donc important de prendre en compte les prévisions climatiques à court terme pour anticiper les effets de l'inertie dans la stratégie de régulation.

De manière générale, la régulation d'un système peut s'effectuer en boucle ouverte (*feedforward control*), en boucle fermée (*feedback control*) ou par une régulation mixte (*feedforward-feedback control*) (Box & Jenkins, 1976). En boucle ouverte, la régulation se base sur les mesures des perturbations de manière à ce que la commande annule leurs effets. La régulation en boucle fermée permet de prendre en considération les perturbations non mesurées qui influencent le fonctionnement du système. En effet, dans ce cas, la loi de commande du régulateur prend comme paramètre l'écart entre la consigne et la valeur réelle. La régulation mixte permet de tirer avantage des deux approches pour donner généralement de meilleurs résultats : les perturbations mesurées et non mesurées sont prises en compte par le régulateur. Les problèmes de régulation sont généralement résolus par des approches issues de l'Automatique où l'identification des modèles formels du procédé et des perturbations est nécessaire (Landau, 1988 ; Faurre & Robin, 1986). Ces approches permettent de garantir certaines caractéristiques de la commande (robustesse). Ainsi, une approche de régulation mixte, inspirée des régulateurs PID (actions Proportionnelles Intégrales et Dérivées), a été mise en place (Brun *et al.,* 1993 ; Brun & Settenbrino, 1993). Toutefois, les caractéristiques du processus de nutrition (inertie, perturbations, prise en compte des prévisions climatiques, nombre important de variables) et l'absence de modèles formels complets laissent ouvert ce problème de régulation pour l'application d'autres techniques, de manière à améliorer les résultats obtenus. C'est pourquoi, plus généralement, des approches issues de l'Intelligence Artificielle ont déjà été appliquées dans des contextes de nutrition de plantes : systèmes experts (Fynn *et al.*, 1989), techniques d'apprentissage (Kurata, 1988) et réutilisation d'expériences passées (Chiriatti & Plant, 1993).

Contrairement à ces approches, nous proposons de réutiliser des expériences passées concrètes de régulation en prenant en compte les *évolutions* passées de variables ainsi que leurs prévisions. Ces évolutions sont déterminantes dans la prise de décision pour la régulation notamment à cause du phénomène d'inertie. De plus, notre approche vise à réutiliser les expériences passées de régulation pour reproduire les séquences de commandes réussies, tout en évitant celles dont la qualité n'est pas suffisante. Dans un premier temps, un atelier de commande mettant en œuvre une coopération homme-machine pour opérer la régulation est nécessaire, et permettra la mise en évidence de comportements plus généraux du processus. Ces connaissances générales pourront ensuite permettre la mise en place d'une automatisation complète.

1.2 Aide à la navigation sur le World-Wide-Web

Le World-Wide-Web, ou plus communément le Web, est un gigantesque hypermédia dont nous présentons tout d'abord les principales caractéristiques. Puis, nous introduisons nos motivations pour la réalisation d'un assistant de navigation réutilisant les expériences d'un groupe d'utilisateurs.

1.2.1 Présentation du Web

Le World-Wide-Web est né des recherches débutées en 1989 au CERN par Tim Berners-Lee. L'objectif était de disposer d'un système permettant à un ensemble de chercheurs, répartis sur plusieurs sites, de pouvoir accéder et publier des informations de manière simple et unifiée, pour constituer une base documentaire partagée. Une contrainte majeure était de pouvoir relier des documents de projets *a priori* indépendants, si des liens pertinents étaient identifiés par la suite, sans avoir à effectuer des modifications lourdes ou la mise à jour d'une base centrale (Berners-Lee *et al.*, 1994). Le Web repose ainsi sur la notion d'*hypertexte* (cf. Figure I-2) : un hypertexte est un ensemble de nœuds contenant des informations textuelles dans lequel certaines expressions sont liées à d'autres nœuds. Ce type de document électronique permet une lecture non linéaire, et la consultation des documents se fait alors en « navigant » au travers des liens pour rechercher l'information voulue suivant son intérêt.

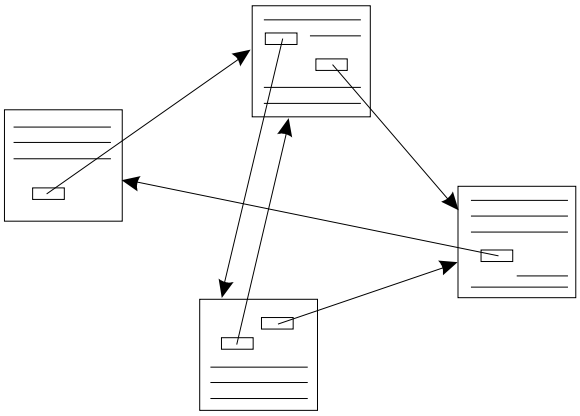


Figure I-2 : exemple simple d'hypertexte (Vanoirbeek, 1996)

Tim Berners-Lee définit le Web comme étant « un système d'information réparti, coopératif, hétérogène et multimédia »[4]. Le Web est en effet un système d'information réparti qui s'étend dans le monde entier puisque les serveurs de documents peuvent être physiquement localisés à différents endroits et que les liens entre les documents peuvent référencer indifféremment n'importe quel autre serveur. Le Web permet aussi de mettre en œuvre la constitution coopérative de documents, car chaque utilisateur peut mettre à jour ses propres données en les liant à celles des autres. Le Web est ensuite hétérogène puisqu'un même document peut être consulté depuis des ordinateurs et des plates-formes différents. Un utilisateur possède également un accès identique aux documents issus des différents serveurs bien que ces derniers soient d'architectures différentes. Le Web est enfin un multimédia puisqu'il permet de relier des documents de différents types : textes bien sûr mais aussi images, animations, sons... Le Web est donc plus qu'un hypertexte, c'est un *hypermédia*.

Les logiciels utilisés pour consulter le Web sont appelés des *navigateurs*, et la Figure I-3 donne l'exemple d'un navigateur affichant une page[5] Web (nœud de l'hypermédia) présentant les différents pointeurs (expressions soulignées) sur d'autres pages à propos de l'INRIA : il suffit alors de cliquer sur l'expression choisie pour automatiquement afficher la page correspondante qui peut se trouver physiquement sur le serveur de l'INRIA ou sur un autre serveur distant. Certains liens peuvent aussi référencer des sous-parties de la page courante.

[4] http://www.w3.org/Talks/General.html

[5] Nous préférons le terme de *page* plutôt que celui de *document* qui fait référence à « une unité de sens autonome et complète liée au médium papier » (Chartron, 1996). En fait, un document Web est très souvent constitué de plusieurs pages.

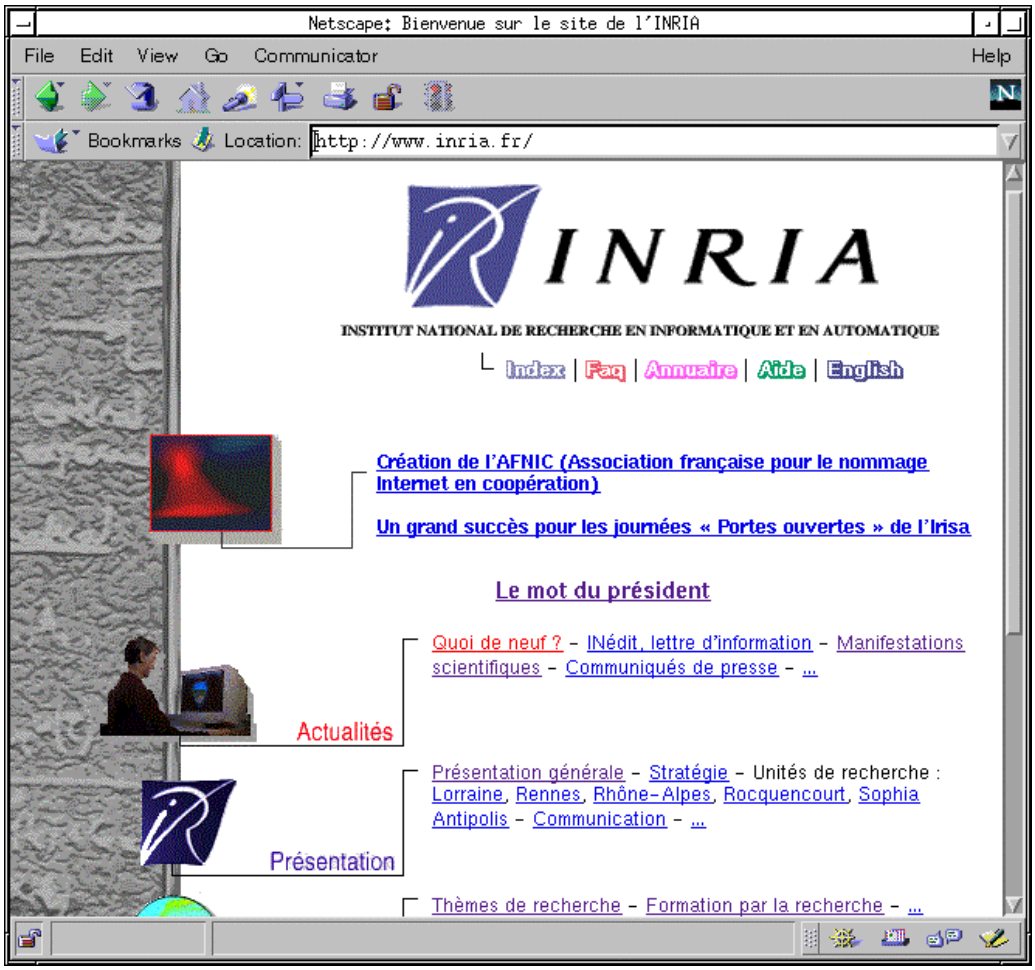


Figure I-3 : affichage d'une page Web avec le navigateur Netscape

Sur le plan de la communication entre les clients et les serveurs, le Web s'appuie sur Internet, une « interconnexion de réseaux couvrant la planète » (Huitema, 1995). Avec des débuts en 1969 par le réseau de l'armée américaine Arpanet, Internet regroupe un ensemble d'ordinateurs et de protocoles de communications permettant leur connexion (dont notamment les protocoles *Internet Protocol* et *Transmission Control Protocol*). Au-dessus de ces protocoles, un protocole spécifique, nommé HTTP (*Hypertext Transfer Protocol*), a été ajouté pour l'échange des pages hypertextes entre les serveurs du Web et les clients (RFC 2068, 1997). Toutefois, le Web ne se limite pas aux pages accessibles directement par ce protocole, mais définit un espace virtuel dans lequel chaque élément est accessible par une adresse appelée URI[6] (*Uniform Resource Identifier*), où l'utilisation d'autres protocoles existants est possible (RFC 1738, 1994). Cet espace est souvent défini par les éléments accessibles par les protocoles FTP (*File Transfer Protocol*) et Gopher, en plus du protocole HTTP (cf. Figure I-4). Cependant d'autres protocoles sont aussi utilisés : WAIS (*Wide Area Information Server*), Telnet, NNTP (*Network News Transport Protocol*).

Les différentes pages du Web sont généralement écrites dans un langage de formatage nommé HTML[7] (*Hypertext Mark-up Language*) permettant de standardiser l'affichage (Raggett, 1997 ; Raggett *et al.*, 1998). Certaines pages sont stockées dans les fichiers des serveurs et d'autres sont

[6] Pour des raisons historiques, URL (*Uniform Resource Locator*) est également très souvent employé. Toutefois, nous avons opté pour le nouveau terme URI (RFC 2068, 1997).
[7] Un autre langage, nommé XML (*eXtensible Mark-up Language*), est également à l'étude pour améliorer la gestion des documents, notamment en terme de structuration.

construites dynamiquement suivant le contexte (interrogation d'une base de données par exemple). Enfin, de nombreux formats sont également utilisés pour les documents non hypertexte : texte non formaté, Postscript, PDF (*Portable Document Format*)...

Bien que l'évaluation de l'étendue du Web soit une tâche difficile, certaines estimations (Bray, 1996) datées de 1995 montraient que le Web comportait 223 851 serveurs différents et 11 366 121 URI accessibles ; cependant les auteurs précisaient que des estimations plus récentes de 1996 portaient déjà ce dernier nombre à plus de 50 millions. Cet espace virtuel est en expansion croissante et offre ainsi une énorme quantité de données.

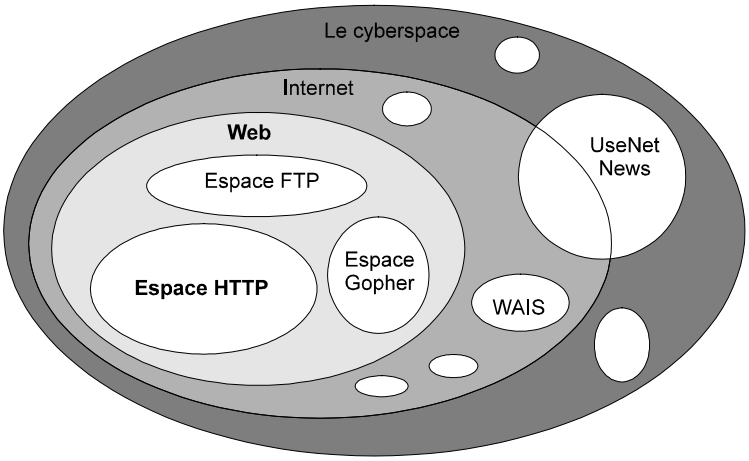


Figure I-4 : l'espace virtuel du Web pour Lycos[8]

1.2.2 Assistance à la navigation sur le Web

La recherche d'informations pertinentes sur le Web est une activité importante de l'utilisateur qui dispose d'un ensemble d'outils généraux ou spécialisés (Chartron, 1996). Les outils généraux regroupent : les répertoires thématiques construits en partie grâce à une intervention humaine (Yahoo[9], QuiQuoiOu[10] de France Télécom), les moteurs de recherche faisant une indexation automatique grâce à des robots qui parcourent le Web (Altavista[11], Lycos[12]), et les métaoutils permettant de combiner plusieurs autres outils (MetaCrawler[13], CBKB (Andreoli *et al.*, 1996) de Xerox). Les outils spécialisés utilisent des techniques similaires mais restent liés à un pays, à un espace linguistique ou à un type de contenu. Par exemple, les pages regroupant un ensemble de liens pertinents établis par un spécialiste constituent des outils très spécialisés.

Du point de vue de l'utilisateur, ces outils proposent deux types d'interaction : la *formulation de requêtes* (moteurs de recherche et métaoutils existants) et la *navigation hypertextuelle* (répertoires thématiques, pages spécialisées, serveurs Web). Cependant, la navigation reste le moyen principal de recherche. En effet, les moteurs de recherche nécessitent l'expression d'une requête généralement sous la forme de mots clefs. Cette requête peut être difficile à formuler si l'utilisateur n'a pas un but très précis. De plus, les résultats obtenus présentent un bruit important provenant de l'indexation automatique utilisée. C'est pourquoi l'utilisateur est très souvent obligé de naviguer, les résultats des outils servant tout de même de points de départ raisonnables (Jerke *et al.*, 1990). Toutefois, la

[8] Lycos est un robot de recherche indexant les informations disponibles par les protocoles HTTP, Gopher et FTP (cf. http://www.vperson.com/mlm/lycos-websize-9510.html).
[9] http://www.yahoo.com
[10] http://www.wanadoo.fr/bin/frame.cgi?service=quiquoiou
[11] http://altavista.digital.com/
[12] http://www.lycos.fr/
[13] http://www.metacrawler.com/

navigation est également une tâche difficile. Dans les répertoires, le facteur humain induit des biais dans la réalisation de la classification et ne permet pas d'être aussi exhaustif qu'une indexation automatique par robots. De plus, la complexité et l'étendue du Web peuvent entraîner la désorientation de l'utilisateur lors d'une navigation. Ce dernier problème est d'ailleurs général aux systèmes hypertextes (Foss, 1989; Dillon *et al.*, 1990).

Ainsi, la navigation hypertextuelle est nécessaire pour la recherche d'information sur le Web, et reste difficile. Pour faciliter cette navigation, une première approche est d'améliorer la conception des pages Web en définissant des règles pour les auteurs (Bachiochi *et al.,* 1997), mais ces dernières seront de toute manière plus ou moins suivies. C'est pourquoi, dans notre travail, nous nous orientons vers des systèmes de recommandations comme WebWatcher (Armstrong *et al.*, 1997) et Letizia (Lieberman, 1995), dans lesquels l'utilisateur est observé, et des recommandations de pages pertinentes lui sont faites. S'il juge ces recommandations intéressantes, l'utilisateur peut alors suivre les liens proposés. Ces approches permettent de guider l'utilisateur durant sa navigation.

Plus précisément, nous visons un assistant permettant une recherche d'information coopérative, où l'expérience d'un groupe de personnes est mise à profit. La prise en compte du *comportement des utilisateurs* (cf. Figure I-5), constitué notamment de la séquence des pages visitées et de leurs contenus, doit également permettre de mieux cerner l'intérêt de l'utilisateur pour affiner les recommandations. Notre approche soutient l'hypothèse suivante : *si deux utilisateurs ont navigué suivant une séquence similaire de pages, c'est qu'ils ont des intérêts similaires, nous pouvons alors proposer au second utilisateur les pages jugées pertinentes par le premier*. Ainsi, cet assistant doit permettre la réutilisation de navigations passées d'un groupe d'utilisateurs pour réduire la longueur des navigations et faciliter ainsi la recherche d'information.

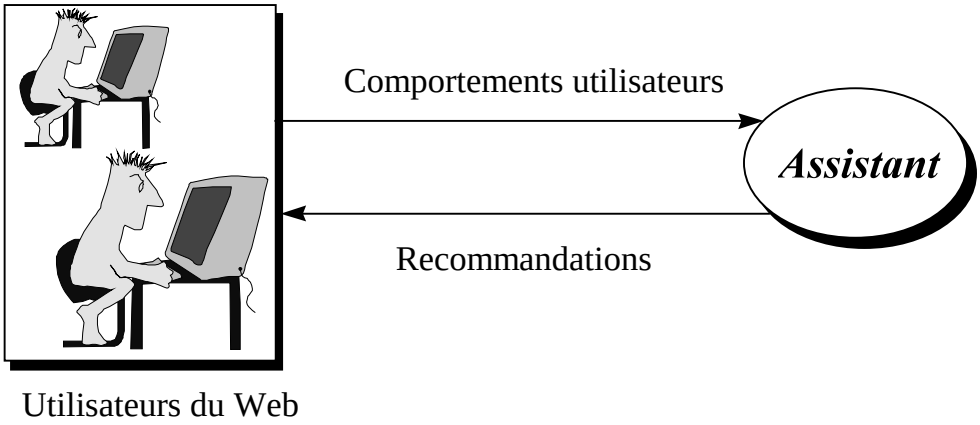


Figure I-5 : principe d'un assistant à la navigation

2 Présentation du raisonnement à partir de cas

Après avoir décrit nos motivations, nous introduisons les notions importantes du raisonnement à partir de cas. De manière générale, le raisonnement à partir de cas est une approche de résolution de problèmes basée sur la réutilisation par analogie d'expériences passées appelées *cas* au cours d'un cycle de raisonnement. Nous présentons tout d'abord les principes de ce type de raisonnement puis nous donnons les caractéristiques de son utilisation. Nous ne donnons pas dans cette section une

description exhaustive du raisonnement à partir de cas, notre objectif étant uniquement de présenter les concepts et la problématique de cette approche pour introduire le vocabulaire utilisé par la suite[14].

2.1 Principes du raisonnement

Le raisonnement à partir de cas est une approche de résolution de problèmes datant du début des années 1980 et issue de travaux en Sciences Cognitives et en Intelligence Artificielle (Schank, 1982 ; Kolodner, 1983 ; Riesbeck & Schank, 1989 ; Kolodner & Leake, 1996). Bien que le raisonnement à partir de cas soit inspiré de certaines observations sur le raisonnement humain, le RàPC prend aussi en compte le besoin pragmatique de techniques et de processus opérationnels.

2.1.1 Représentation des cas et des indices

Schématiquement, un cas est composé de deux parties décrivant : un *problème* et la *solution* qui lui a été appliquée. Sous ces deux termes simples, une représentation complexe est souvent nécessaire pour tirer parti des enseignements d'un cas de manière opérationnelle. La description du problème peut recouvrir les objectifs à atteindre, les contraintes sur ces objectifs et le contexte dans lequel le problème a été posé. La solution peut regrouper la description de la solution effectivement apportée, les étapes qui ont mené à cette solution, son évaluation et les effets de son application. Cette représentation dépend de la tâche à accomplir, du type des données disponibles et des connaissances du domaine. Bien que le raisonnement à partir de cas repose sur la réutilisation d'enseignements spécifiques, des cas abstraits (prototypes de cas) peuvent être définis (Bergman & Wilke, 1996). Cependant ces représentations plus générales doivent rester à un niveau *opérationnel*, c'est-à-dire que leur utilisation doit être uniquement basée sur des mécanismes d'instanciation et d'adaptation (Kolodner, 1996).

Idéalement, un cas est une expérience passée permettant au système de raisonnement à partir de cas de résoudre des problèmes plus efficacement ou d'éviter les échecs passés : un cas doit donc être *utile* pour le raisonnement (Kolodner, 1993). Dans ce but, un ensemble de caractéristiques est extrait ou sélectionné dans la représentation de chaque cas. Ces caractéristiques, appelées *indices*, déterminent dans quelle *situation* un cas est applicable et utile. Les indices doivent tout d'abord être prédictifs, c'est-à-dire qu'ils doivent être responsables de l'élaboration de la solution mise en œuvre dans le cas et permettront ainsi de déterminer dans quelle mesure une solution est applicable pour un autre problème. Ils doivent également être suffisamment abstraits pour permettre la réutilisation des enseignements du cas pour différents problèmes, tout en restant suffisamment concrets pour être facilement identifiables dans un nouveau problème.

2.1.2 Cycle du raisonnement

Lorsqu'un *nouveau problème* est à résoudre, il est intégré dans un *cas cible*[15] où la partie solution est inconnue et doit être apportée par le raisonnement. Les *cas sources* représentent des expériences passées qui sont stockées dans une *mémoire*. Les cas sources et les cas cibles partagent le même formalisme de représentation. L'objectif du raisonnement est de transférer des enseignements

[14] Pour plus de renseignements sur le raisonnement à partir de cas, le lecteur pourra se reporter aux ouvrages et publications spécialisées (Gebhardt *et al.,* 1997 ; Watson, 1997 ; Aamodt & Plaza, 1994 ; Kolodner, 1993 ; Slade, 1991 ; Kolodner & Leake, 1996).

[15] Le terme « nouveau cas » est parfois employé (Aamodt & Plaza, 1994). Nous préférons le terme de « cas cible » qui convient mieux à l'objectif du raisonnement où il s'agit de transférer des connaissances depuis des anciens cas, les cas sources, vers un cas courant, le cas cible. De plus, le terme « nouveau cas » peut induire en erreur, en suggérant que ce cas sera obligatoirement ajouté à la mémoire, or seule la phase d'apprentissage décidera en fin de raisonnement si l'expérience courante doit effectivement devenir un nouveau cas de la mémoire.

pertinents des cas sources pour élaborer la solution du cas cible. Ce raisonnement se décompose habituellement en quatre phases principales (Aamodt & Plaza, 1994) qui peuvent être totalement ou partiellement automatisées (cf. Figure I-6) :

1. phase de *recherche* dont le but est de sélectionner un ou plusieurs cas utiles pour la suite du raisonnement, tout en prenant en compte les indices du cas cible déterminés par une étape d'élaboration ;
2. phase de *réutilisation* permettant de construire une solution au problème courant en se basant sur les cas identifiés dans la phase précédente ;
3. phase de *révision* de la solution qui permet d'affiner la solution grâce à son évaluation ;
4. phase *d'apprentissage* chargée de mettre à jour les éléments du raisonnement en prenant en compte l'expérience qui vient d'être réalisée, et qui pourra ainsi être utilisée dans les raisonnements futurs.

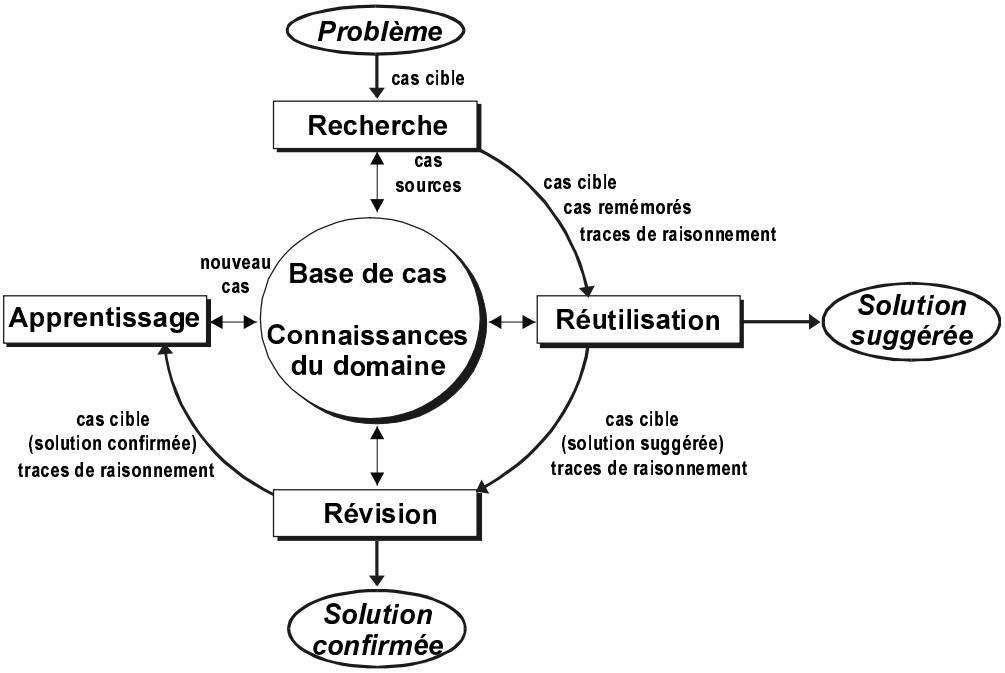


Figure I-6 : cycle du raisonnement à partir de cas

La problématique de la phase de recherche est de permettre d'identifier un ou plusieurs cas utiles pour le raisonnement en cours. La mesure de cette utilité peut être basée sur la similarité entre les indices du cas cible et des cas sources, ou sur des critères d'adaptabilité (Leake *et al.*, 1997). Généralement, deux étapes sont réalisées : *filtrage* des cas sur des indices de surface, puis *sélection* fine des cas. Toutefois, nous pouvons considérer l'étape de sélection comme un filtrage particulier, et en généralisant, nous dirons que la phase de recherche se compose d'une succession de filtrages. La réalisation de ces filtrages s'appuie sur la notion d'index qui permet d'organiser les cas sources au sein de la mémoire, en utilisant leurs indices. Un index est une structure de données utilisée par un algorithme offrant une recherche efficace, tout en utilisant des connaissances du domaine et/ou des connaissances induites pour identifier les cas les plus utiles. Dans ce sens, l'objectif d'un index dépasse les approches classiques de recherche des bases de données, même si des techniques issues de ce domaine sont parfois utilisées. Par exemple, un *index par plus proches voisins* s'appuie sur une structure linéaire des cas et sur une mesure de similarité. Dans cet index, la similarité entre le cas cible et chaque cas source est calculée, pour ne retenir que les k meilleurs (algorithme en $O(n)$ où n est le nombre de cas sources). D'autres index sont couramment utilisés (cf. chapitre III) : index par arbre d'induction ou index par hiérarchie de prototypes.

Etant donné qu'il est peu probable de retrouver un cas correspondant exactement au problème courant, la phase de réutilisation permet de construire une solution par adaptation des solutions passées des cas sources identifiés. Cette réutilisation se base principalement sur les différences entre les cas sources et le cas cible pour déclencher des méthodes d'adaptation. Après adaptation, la solution suggérée au problème courant peut être donnée à l'utilisateur uniquement si la phase de révision n'est pas effectuée ou si elle est manuelle.

La phase de révision permet l'évaluation et la réparation de la solution déterminée à la phase de réutilisation. Cette évaluation peut être basée sur la vérification de la solution par un expert, par une simulation, ou même par une exécution réelle. Si la solution est mauvaise, des explications peuvent être fournies ou générées pour que le système (ou un opérateur) puisse effectuer des actions de réparation afin d'améliorer la solution et de fournir la solution confirmée.

Dans la phase d'apprentissage, le système doit faire la synthèse du raisonnement qui a été suivi de manière à sauvegarder les nouveaux enseignements pertinents qui permettront d'améliorer le raisonnement par la suite. Plusieurs actions peuvent alors être effectuées : indexation du cas cible dans la mémoire, amélioration et modification de l'indexation des cas existants dans la mémoire (par exemple création de prototypes regroupant un ensemble de cas similaires). Si le cas cible est jugé intéressant et qu'il doit être indexé dans la mémoire, la méthode d'apprentissage doit déterminer quels sont les indices du cas cible de manière à définir dans quelles situations le cas sera de nouveau utile pour le raisonnement.

2.1.3 Importance de l'indexation pour le raisonnement et terminologie

L'indexation en RàPC consiste à définir les indices d'un cas, c'est-à-dire les caractéristiques d'un cas qui indiquent ses conditions d'application et d'utilité (les indices). Cette tâche peut être appelée le « problème de l'accessibilité » (Kolodner, 1996, page 355) étant donné que la phase de recherche est effectuée en premier dans le cycle du raisonnement et que les indices doivent mener, dans cette phase, à l'identification des cas à utiliser par la suite. Cependant, le but du raisonnement est clairement de réutiliser les connaissances issues des cas, soit manuellement soit automatiquement. C'est pourquoi les indices d'un cas doivent définir la situation dans laquelle ses enseignements peuvent être réutilisés : un cas de succès est réutilisé pour atteindre les objectifs courants, et un cas d'échec est réutilisé pour prévenir un échec potentiel. Ainsi, le problème de l'indexation doit aborder plus que la problématique de l'accessibilité et nous pouvons le nommer *problème de la réutilisabilité*.

Les cas sources et le cas cible courant partagent un même formalisme de représentation, généralement formé d'attributs, pour que les opérateurs de mise en correspondance et d'adaptation puissent être appliqués. Dans ce formalisme, trois types d'attributs sont identifiés en fonction de leur rôle pour l'indexation :

- *attributs toujours ignorés* : par exemple dans le cadre d'une application donnée, le nom du cas et sa date de création peuvent ne jamais être pertinents comme indices,
- *indices potentiels* : ces indices, provenant principalement de la description d'un problème, peuvent être utilisés pour déterminer l'utilité d'un cas et ses conditions d'application,
- *indices effectifs* ou simplement indices : ces indices déterminent précisément, pour un cas donné, les indices potentiels qui sont effectivement sélectionnés pour guider l'utilisation du cas (situation du cas). Les indices effectifs d'un cas source peuvent être déterminés *a priori* ou en fonction du cas cible courant.

Durant le raisonnement, un ou plusieurs cas sources sont réutilisés pour calculer la solution du cas cible, et les indices interviennent dans les phases de recherche, de réutilisation et d'apprentissage. La phase de révision ne fait généralement pas appel aux indices dans la mesure où les méthodes de réparation se basent uniquement sur la solution qui vient d'être construite. Durant la phase de

recherche, un sous-ensemble des indices effectifs des cas sources et du cas cible, que nous appelons *indices de recherche*, est mis en correspondance pour identifier les cas à utiliser dans la suite du raisonnement. Puis, dans la phase de réutilisation, certains indices effectifs, les *indices d'adaptation*, sont comparés entre le cas source et le cas cible pour participer au choix des actions d'adaptation à effectuer. En effet, un des aspects de la réutilisation est d'analyser « les différences entre le cas passé et le cas courant » (Aamodt & Plaza, 1994). Des indices d'adaptation peuvent ne pas être des indices de recherche : ils permettent une analyse plus fine des différences entre un cas source et un cas cible mais ne sont pas suffisamment discriminants pour être utilisés dans la recherche. Enfin, durant l'apprentissage, la mémoire peut être modifiée par l'ajout d'un cas ou la réorganisation de l'indexation grâce à la modification des indices effectifs au sein des indices potentiels et de leurs poids d'importance. Ainsi, le problème de l'indexation est au cœur du raisonnement à partir de cas et intervient dans les trois phases importantes du raisonnement (recherche, réutilisation, apprentissage).

Enfin, nous résumons la définition des termes introduits pour l'indexation, qui seront utilisés tout au long du document[16] :

- *Indice*. Un indice (effectif) est une caractéristique d'un cas permettant de définir ses conditions d'applicabilité et d'utilité.
- *Situation*. Une situation définit l'ensemble des indices effectifs d'un cas.
- *Filtrage*. La phase de recherche est décomposée au niveau connaissance en étapes de filtrage. Un filtrage permet de retourner un sous-ensemble de cas sources applicables et jugés utiles pour la suite du raisonnement. La dernière étape de filtrage est appelée sélection.
- *Index*. Un index est une structure de données utilisée par un algorithme permettant la réalisation effective d'un ou de plusieurs filtrages, tout en prenant en compte les connaissances du domaine et les contraintes de performance. Un index est également mis à jour lors de l'apprentissage.

2.2 Utilisation du raisonnement à partir de cas

Comme toute méthode de résolution de problèmes, le raisonnement à partir de cas présente des avantages, des inconvénients et des conditions d'application que nous décrivons brièvement. Nous donnons alors l'exemple d'un système de RàPC, nommé CLAVIER (Mark *et al.*, 1996), pour illustrer le cycle du raisonnement et sa problématique.

2.2.1 Avantages, inconvénients et conditions d'application

Le raisonnement à partir de cas présente de nombreux atouts et avancées dans le domaine de l'Intelligence Artificielle (Leake, 1996 ; Slade, 1991 ; Kolodner, 1992) portant notamment sur : l'acquisition des connaissances, la maintenance de ces connaissances, l'efficacité et la qualité du raisonnement et enfin son explication. Tout d'abord, sur le plan de l'acquisition des connaissances, le raisonnement à partir de cas demande généralement moins d'efforts que la constitution d'une base de règles. D'une part, dans certaines applications, les personnes raisonnent explicitement en utilisant des expériences passées et des dossiers existants peuvent être utilisés (Mark *et al.,* 1996). D'autre part, les cas représentent des enseignements spécifiques plus faciles à expliciter que des règles générales.

[16] La définition de ces termes est importante car il n'existe pas de consensus au sein de la communauté RàPC. Par exemple, le terme « index » peut représenter, suivant les auteurs, ce que nous appelons une situation ou encore un indice. De plus, en anglais les termes « *indices* », « *indexes* » et « *index* » ont la même signification, ce qui amène parfois une confusion regrettable.

Le raisonnement à partir de cas offre également des possibilités de maintenance de la connaissance puisque de nouveaux cas peuvent être ajoutés à la mémoire ou bien retirés (Smyth & Keane, 1995). Par exemple, un ensemble de cas doit être initialement conçu puis, suivant les besoins et les échecs du raisonnement, de nouveaux cas peuvent venir enrichir les connaissances du système.

De plus, l'utilisation du raisonnement à partir de cas peut permettre une plus grande efficacité du processus de résolution de problèmes (Veloso, 1996) car celui-ci n'est pas entièrement répété à chaque fois à partir des principes de base lorsqu'un problème similaire se présente. Dans les systèmes experts, par exemple, le même ensemble de règles doit être déclenché. La qualité du raisonnement est également améliorée, d'une part, grâce aux échecs passés qui peuvent être évités et, d'autre part, grâce à l'utilisation d'expériences passées réelles qui offrent une validité au raisonnement dans des domaines où les connaissances sont réduites.

Enfin, l'explication du raisonnement est générée beaucoup plus facilement et de manière plus satisfaisante que dans les systèmes experts ou d'autres systèmes apprentis comme les réseaux de neurones : les cas sources qui sont généralement des expériences vécues servent de justification. De plus, le fonctionnement du raisonnement étant intuitivement compréhensible par les utilisateurs, ces justifications sont d'autant mieux acceptées, et les utilisateurs peuvent être associés au raisonnement dans des systèmes interactifs (Mark *et al.,* 1996).

Toutefois, cette technique présente quelques inconvénients et des conditions d'application. Le problème majeur est l'application aveugle des expériences passées : notamment l'utilisation de mécanismes d'adaptation peut rendre obscures les justifications issues des cas sources aux yeux des utilisateurs (Mark *et al.,* 1996). De plus, le système peut ne pas proposer la solution optimale, puisque l'espace des problèmes peut ne pas être entièrement connu : ceci pose le problème de la *couverture du domaine*. L'évaluation de cette couverture est difficile et ne peut pas être réduite à un critère simple comme la taille de la base de cas (Kolodner, 1996). Le raisonnement à partir de cas doit être considéré lorsqu'il est difficile de formuler des règles et que des cas peuvent être représentés ou sont disponibles. Toutefois, l'application de ce type de raisonnement repose sur des hypothèses sur le modèle de l'univers en cours d'étude (Kolodner, 1996) :

- *Régularité* : les mêmes actions menées dans les mêmes conditions ont des effets identiques ou proches.
- *Typicalité* : les problèmes se répètent et les expériences apprises peuvent être utiles dans le futur.
- *Cohérence* : de petits changements dans l'univers nécessitent seulement de petits changements dans la façon dont on l'interprète et dans les solutions.
- *Facilité d'adaptation* : bien que tout ne se répète pas à l'identique, les différences tendent à être petites, et ces petites différences peuvent être facilement prises en compte.

Très souvent une modélisation trop simpliste de cet univers ne permet pas de garantir ces hypothèses. Il s'agit donc premièrement de modéliser cet univers en choisissant une représentation suffisamment *abstraite* pour les cas et notamment pour les indices. Ainsi, même si des *caractéristiques de surface* peuvent facilement être identifiées, il s'agit de vérifier leur qualité en tant qu'indices et notamment leur prédictivité, sinon des caractéristiques plus abstraites sont nécessaires (Kolodner, 1996). Deuxièmement, les traitements effectués pendant le raisonnement sont déterminants. Par exemple, au lieu de réutiliser directement une ancienne solution par transformation, *l'analogie par dérivation* (Carbonell, 1986) propose de réutiliser les étapes qui ont mené à l'élaboration de cette solution. Enfin, dans un domaine où les experts ne raisonnement pas eux-mêmes en réutilisant des expériences passées, toutes ces hypothèses sont difficilement vérifiables *a priori* et devront généralement être évaluées par la réalisation et l'évaluation expérimentale d'un prototype.

2.2.2 Un exemple : le système CLAVIER

Le système de raisonnement à partir de cas CLAVIER est en utilisation journalière depuis les années 1990 chez Lockheed et permet la configuration d'un autoclave (Mark *et al.*, 1996 ; Althoff *et al.*, 1995, chapitre 6 ; Hennessy & Hinkle, 1992). Un autoclave est une sorte de four nécessaire à la fabrication d'éléments composites qui sont utilisés dans les avions et les missiles (cf. Figure I-7). Un autoclave est une ressource coûteuse, c'est pourquoi plusieurs éléments sont placés ensemble pour une même cuisson. Cependant, le résultat de la cuisson dépend grandement de la façon de positionner les éléments entre eux. Un seul défaut dans les pièces obtenues entraîne leur rejet et des pertes financières considérables. Les connaissances sur ce processus sont très limitées et une tentative de modélisation par règles avait d'ailleurs échoué. Comme les opérateurs utilisaient déjà des descriptions sous forme de fiches proches de la notion d'un cas, l'utilisation du raisonnement à partir de cas dans CLAVIER fut alors naturelle.

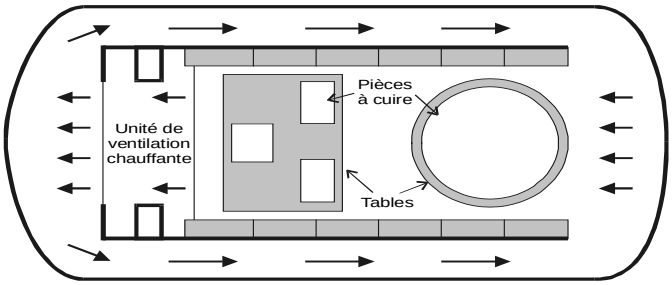


Figure I-7 : chargement de l'autoclave

Le système CLAVIER permet d'automatiser partiellement le positionnement des éléments dans l'autoclave. Un cas source représente une expérience passée donnant d'une part une liste d'éléments avec leur type et, d'autre part le positionnement utilisé dans l'autoclave et son résultat (valide ou invalide). Un cas cible est constitué d'une liste d'éléments à cuire avec leurs priorités et du positionnement à déterminer. Le raisonnement suit alors les phases suivantes :

- *Recherche*. La phase de recherche permet d'identifier le plus proche cas source valide proposant un positionnement pour le plus grand nombre d'éléments courants ayant les plus fortes priorités.
- *Réutilisation*. Ensuite, la phase de réutilisation permet d'adapter le positionnement en substituant les éléments utilisés dans le cas source et non mis en correspondance, par des éléments du cas cible qui n'ont pas encore été disposés. Cette adaptation peut être automatique ou manuelle. En fait, l'adaptation était au départ automatisée mais fut mal acceptée par les utilisateurs qui préféraient effectuer l'adaptation eux-mêmes.
- *Révision*. Dans ce dernier cas, le système vérifie alors la solution dans une phase de révision pour éviter de tomber sur un positionnement invalidé dans le passé. La révision est alors dans tous les cas finalisée par la mise en place de la disposition dans l'autoclave. A la suite de l'utilisation de l'autoclave, la disposition est validée ou invalidée.
- *Apprentissage*. Enfin, le cas cible courant est ajouté dans la mémoire qu'il soit valide ou invalide.

Dans cet exemple, les différents ensembles d'indices (indices potentiels et effectifs ainsi que les indices de recherche et d'adaptation) introduits précédemment, sont tous identiques : ils donnent la liste des éléments et leur type. Le type des éléments a en effet été jugé suffisant pour guider les phases du raisonnement ; les informations sur la géométrie ou la constitution des éléments sont ignorées.

3 Application du RàPC à une classe de problèmes

L'étude des problèmes de la régulation de la nutrition des plantes et de l'assistance à la navigation nous a permis de mettre en évidence une classe de problèmes dans laquelle l'observation d'un processus dynamique et la prise en compte d'historiques sont importantes. L'application du raisonnement à partir de cas dans ce contexte est intéressante et présente une spécificité notamment liée à la gestion d'indices reposant sur des historiques. Pour aborder cette classe de problèmes avec du RàPC, nous visons la définition d'un modèle générique d'indexation et la conception d'un outil ouvert facilitant son utilisation.

3.1 Classe des problèmes nécessitant la prise en compte d'historiques

Bien que les applications ayant motivé notre étude, la régulation de la nutrition des plantes et l'assistance à la navigation sur le Web, soient des problèmes issus de domaines bien différents, nous avons identifié certaines caractéristiques pertinentes et communes. Nous avons ainsi défini une classe de problèmes ayant les cinq propriétés suivantes :

1. La prise en compte *d'historiques* est choisie ou nécessaire pour formuler le problème.
2. Un *processus dynamique*, c'est-à-dire un processus évoluant dans le temps, est observé.
3. Un ensemble *important* de données brutes est collecté.
4. Les données brutes peuvent être obtenues par *échantillonnage* et/ou lors *d'événements*.
5. Les connaissances du domaine sont *réduites* et une meilleure compréhension est recherchée.

Nous pouvons alors vérifier que les approches prises dans les applications précédemment décrites font partie de cette classe de problèmes. Pour la nutrition, le processus est constitué par la boucle de régulation opérant sur des plantes dans un environnement. Pour la navigation sur le Web, le processus observé intègre les utilisateurs qui naviguent sur le Web avec l'aide de l'assistant. Il est à noter que, dans les deux cas, le processus observé comprend le système chargé de résoudre le problème posé : le régulateur ou l'assistant. En effet, ces systèmes sont actifs, modifient le comportement global du processus, et peuvent avoir des variables d'état dont les valeurs évoluent dans le temps. Dans les deux cas, un nombre important de données est collecté (perturbations, commandes, pages visitées, caractéristiques des documents...). Ces données peuvent être échantillonnées avec une période donnée (par exemple le jour), ou obtenues lors d'événements (déclenchement de la ventilation, alarme, visualisation de pages spécifiques). Dans les deux domaines, peu de connaissances formelles sont disponibles et la prise en compte d'historiques permet une meilleure description du problème à résoudre qu'une description par un ensemble figé de valeurs instantanées de variables d'état. Pour la nutrition, l'inertie du système implique la prise en compte des commandes passées et des évolutions des perturbations pouvant remonter à plusieurs jours. De même, pour la navigation, une meilleure description de l'intérêt de l'utilisateur lors d'une recherche d'informations est obtenue en prenant en compte les événements pertinents de l'historique de ses déplacements.

Enfin, l'intérêt d'aborder cette classe de problèmes dépasse le cadre de nos deux applications. Pour de nombreux problèmes, une composante essentielle réside dans l'observation d'un processus avec la prise en compte d'historiques. De manière générale en médecine, l'évolution de l'état d'un patient est souvent plus pertinente qu'un état particulier (Keravnou, 1995), et les fiches médicales ou les enregistrements doivent être consultés. Dans les problèmes d'analyse et de compréhension du comportement d'un être humain utilisant une machine, il est nécessaire de prendre en compte des séquences temporelles d'interactions (Ritter & Larkin, 1994). Dans le domaine de la conduite de procédés, des délais de réaction importants et variables peuvent être observés, ce qui oblige à prendre

en compte des événements passés de plusieurs minutes et même de plusieurs heures : par exemple, dans la conduite de hauts fourneaux, une erreur d'alimentation peut être observée six heures plus tard (Jakob & Suslenschi, 1990).

3.2 Intérêt et spécificité du RàPC

L'application du raisonnement à partir de cas pour cette classe de problèmes soulève une certaine spécificité dont l'impact majeur repose sur la représentation et la gestion des indices durant les raisonnements. Il s'agit en effet de représenter des historiques comme indices, tout en gardant une représentation compréhensible des indices permettant d'utiliser les cas comme explication du raisonnement.

3.2.1 Justification et intérêt du raisonnement à partir de cas

Nous abordons ainsi des problèmes liés à un processus dynamique pour lequel de nombreux paramètres peuvent être identifiés et observés, mais dont leurs influences sur le comportement du processus sont mal connues. Dans ce cadre, le rôle d'une approche basée sur le raisonnement à partir de cas est double :

- raisonner pour élaborer des solutions aux problèmes à résoudre (prévision, diagnostic, commande, assistance),
- mettre en évidence de nouvelles connaissances.

Le raisonnement à partir de cas est justifié puisque les connaissances du domaine sont réduites (absence de règles) et que l'acquisition des données brutes, nécessaires à la représentation des cas, ne pose en général aucune difficulté (présence de capteurs). L'identification des enseignements utiles issus de la masse de données brutes collectées ainsi que leurs réutilisations sont les points clefs d'une telle approche.

Le raisonnement à partir de cas est également adapté pour la mise en évidence de nouvelles connaissances en utilisant les cas comme explications et justifications du comportement du processus. Cet intérêt peut être illustré dans le cadre des deux applications ayant motivé notre étude. Pour la régulation de la nutrition des plantes, notre objectif est de réaliser un système intégrant l'utilisateur notamment dans la phase d'apprentissage pour l'enrichissement et la modification contrôlés de l'indexation. Ce processus permet de faire émerger des connaissances à propos de l'influence des paramètres sur l'évolution de la concentration du drainage, en utilisant les cas sources comme explications. Pour l'assistance à la navigation, l'identification de parcours typiques est capitale pour l'amélioration de la conception des documents mis à disposition sur le Web (pour des objectifs commerciaux par exemple).

3.2.2 Gestion d'indices sous forme d'historiques

Comme l'énoncé du problème à résoudre comprend des historiques, le formalisme de représentation des indices, nécessaire à la représentation des cas cible et sources, doit être adapté et permettre la représentation de séquences de valeurs pertinentes. Nous faisons ainsi une distinction forte entre d'une part une *situation instantanée*, ensemble d'indices effectifs définissant un état à un instant particulier et d'autre part, une *situation comportementale*, ensemble d'indices effectifs décrivant une évolution dans le temps de cet état. L'utilisation de situations instantanées est l'approche la plus répandue dans les systèmes de raisonnement à partir de cas : par exemple les

systèmes classiques[17] CHEF (Hammond, 1986), JULIA (Hinrichs, 1989) et CASEY (Koton, 1989) utilisent des situations instantanées. Le système CHEF permet de dériver des recettes suivant notamment les ingrédients et le mode de cuisson désirés en se basant sur d'autres recettes stockées dans des cas. Les cas sont indexés suivant un ensemble de buts satisfaits ou d'échecs à éviter. Les indices utilisés décrivent uniquement les caractéristiques pertinentes à l'instant où le cas a été appris : il s'agit d'une situation instantanée. De même, dans le système CLAVIER que nous avons brièvement présenté, les indices décrivent encore une situation instantanée en donnant une liste d'éléments. Ainsi, l'application du RàPC à notre classe de problèmes diffère des applications classiques et présente une spécificité relative à la gestion d'indices sous forme d'historiques. Cette spécificité a un impact sur l'ensemble des tâches de conception du système : représentation, organisation de la mémoire et phases de raisonnement.

3.3 Démarche générale de l'étude

Notre démarche propose de dépasser les limites des approches existantes qui restent liées à des applications spécifiques au sein de la classe de problèmes visée (cf. chapitre II). Plus précisément, nous proposons de définir un modèle d'indexation *générique* ainsi qu'un outil logiciel *ouvert* facilitant son utilisation au sein de systèmes de RàPC.

3.3.1 Nécessité d'un modèle d'indexation générique

La gestion de situations comportementales a un impact majeur sur les solutions à employer pour le *problème de l'indexation des cas* dans un système RàPC. L'identification des indices effectifs, parmi le nombre important de données d'observation disponibles (indices potentiels), est rendue plus difficile. La comparaison des indices intégrant des *séquences de valeurs* est nécessaire pour la phase de recherche (indices de recherche) et pour la phase de réutilisation (indices d'adaptation). Durant l'apprentissage, la sélection des indices effectifs qui guideront les raisonnements futurs est déterminante et la représentation spécifique des indices doit être prise en compte. En définitive, la classe de problèmes abordée est plutôt caractérisée par un certain type de données, les historiques, que par un type de tâches (diagnostic, interprétation, planification) ou par un type de connaissances. La définition du formalisme de représentation des indices et de leur gestion, que nous appelons *modèle d'indexation*, est centrale dans le cadre de cette classe de problèmes. De plus, un modèle d'indexation est aussi un prérequis sur lequel les méthodes de raisonnement spécifiques ou génériques viendront s'appuyer.

Un exemple d'approche similaire est donné par le modèle d'indexation UIF (*Universal Index Frame*, Kolodner, 1993, pages 219–244). Ce modèle d'indexation aborde la classe des problèmes liant des agents et des buts à atteindre. Un ensemble d'attributs est alors défini pour indexer des expériences : l'émotion, la tâche, l'objectif de la personne engagée dans une situation. Ces descriptions générales peuvent alors être utilisées dans différentes applications issues de cette classe de problèmes. Plus généralement, il apparaît nécessaire « de pouvoir identifier des classes de problèmes et de créer un modèle d'indexation pour chacun » (Kolodner, 1993, page 244). Le modèle UIF aborde une classe de problèmes très différente de la nôtre, dans laquelle il n'y a pas de notion d'historique. Ce modèle n'aborde pas non plus le problème de l'indexation dans son ensemble comme nous l'avons défini (cf. §2.1.3). Cependant, cette approche partage avec la nôtre la poursuite de deux objectifs importants :

[17] Ces systèmes sont décrits et analysés dans (Kolodner, 1993).

- faciliter l'analyse de l'application du RàPC à un type de problèmes par la définition d'un formalisme générique,
- permettre le développement d'une compétence méthodologique accumulée à travers l'analyse de différentes applications.

3.3.2 Besoin d'un outil logiciel ouvert pour la gestion du modèle et du RàPC

Le besoin d'outils est reconnu dans la communauté scientifique pour faciliter la spécification, le développement et la mise en place de systèmes RàPC. Janet Kolodner s'exclame d'ailleurs dans la conclusion de son livre « Nous avons besoin d'outils ! » (Kolodner, 1993, page 571) et ce besoin a été plus récemment souligné (Leake, 1996, pages 17–18). Dans notre démarche, nous visons la conception d'un outil pour faciliter :

- l'application du modèle d'indexation proposé,
- la capitalisation de composants logiciels liés à notre modèle et plus largement au RàPC.

La définition d'un modèle d'indexation permet de construire un cadre adapté à une classe de problèmes. Cette démarche descendante permet de faciliter le développement d'applications particulières issues de cette classe de problèmes. Cependant, l'utilisation concrète de ce type de modèle doit prendre en compte les connaissances spécifiques et les techniques applicables dans le cadre d'une application donnée. Ainsi, il est possible d'optimiser certaines opérations ou d'utiliser des hypothèses particulières. L'outil gérant ce modèle doit donc permettre sa *spécialisation* (cf. Figure I-8). De plus, un modèle d'indexation n'aborde qu'une partie de la problématique de la construction d'une application de raisonnement à partir de cas. Il s'agit alors d'effectuer une *intégration* de ce modèle avec d'autres techniques et approches nécessaires pour réaliser le système complet (contrôle et exécution des phases de recherche, de réutilisation, de révision et d'apprentissage). Ainsi, durant les opérations de spécialisation et d'intégration, cet outil doit aider à la gestion de la cohérence et de l'implantation d'un système final. Enfin, cet outil ne doit pas se restreindre à la gestion du modèle d'indexation proprement dit, et doit également gérer l'application du RàPC de manière générale pour effectuer l'intégration.

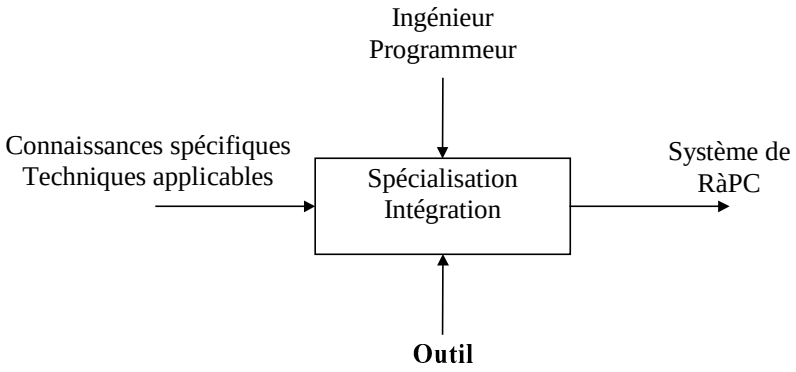


Figure I-8 : tâche de réalisation d'un système de RàPC[18]

Lors de ce processus de spécialisation et d'intégration, de nouveaux composants logiciels sont conçus : ils permettent d'étendre ou d'enrichir l'ensemble des composants existants. C'est pourquoi, nous voulons compléter cette démarche descendante par une *démarche ascendante* de capitalisation de composants. Au fur et à mesure de la réalisation d'applications, il s'agit de constituer un ensemble

[18] Ce diagramme utilise la notation SADT (Lissandre, 1990).

de composants réutilisables pour faciliter progressivement la gestion du modèle d'indexation et du RàPC.

Ainsi, un outil facilitant le processus de spécialisation et d'intégration ainsi que la capitalisation de composants, doit être réutilisable dans différents contextes d'applications et surtout *ouvert* : il doit en effet permettre l'extension des composants logiciels existants ainsi que l'ajout de nouveaux composants.

Chapitre II : modèle d'indexation par situations comportementales

Résumé

Nous abordons le problème de l'indexation pour une classe de problèmes dans laquelle les indices des cas doivent reposer sur des historiques. Peu de travaux existants en RàPC ont abordé cette problématique et ils sont généralement dédiés à des applications particulières. De plus, les approches de reconnaissances d'historiques et les systèmes de gestion de bases de données temporelles ne prennent pas en compte les besoins spécifiques liés à la réutilisation et à la mise à jour des connaissances durant un cycle de raisonnement. Nos apports se situent à deux niveaux. Premièrement, nous proposons alors le « *modèle d'indexation par situations comportementales* » qui est la seule approche générique en RàPC pour l'indexation par historiques. Notre approche comprend le modèle de représentation proprement dit des indices, et un guide d'utilisation. Dans ce modèle, les enregistrements contiennent, dans des chroniques, les observations effectuées suivant différentes variables. La situation comportementale d'un cas définit des comportements élémentaires (séquences de valeurs ou événements) et des contraintes temporelles pour caractériser une expérience utile et précise au sein d'un enregistrement. Deuxièmement, le modèle de représentation des indices permet la gestion d'historiques hybrides (historiques échantillonnés et à événements) et présente de meilleures possibilités pour la découverte de connaissances. Nous permettons en effet l'extraction structurée de cas grâce à la notion de *patron de cas potentiels*. Les cas potentiels instanciés par un patron peuvent ensuite être sauvegardés sous forme de cas concrets, puis mis à jour suivant les raisonnements. Enfin, l'interprétation des cas concrets est facilitée puisque ces cas référencent toujours les données brutes des enregistrements.

Nous abordons dans ce chapitre le problème de l'indexation des cas. Comme nous l'avons montré dans le chapitre précédent, ce problème est crucial pour le raisonnement à partir de cas puisqu'il a des conséquences sur les phases de recherche, de réutilisation et d'apprentissage. Généralement, l'étude de la tâche à aborder par le système mène à la définition d'un vocabulaire d'indexation, donnant les dimensions d'un cas et leurs domaines de valeurs selon lesquelles il est possible d'exprimer les indices du cas. Dans la majorité des systèmes de raisonnement à partir de cas, des indices instantanés sont utilisés : ils représentent les caractéristiques de l'univers étudié à l'instant où le problème a été posé. Contrairement à ces approches, nous étudions une classe de problèmes dans laquelle la prise en compte de l'évolution de certaines variables est requise pour formuler le problème à résoudre. Il s'agit donc de représenter et de gérer des indices sous forme *d'historiques*.

Dans la première partie, nous menons l'état de l'art des approches abordant l'indexation par historiques au sens large. Ces approches sont issues non seulement du domaine du raisonnement à partir de cas, mais aussi de la reconnaissance d'historiques et des bases de données temporelles. Dans la deuxième partie, nous proposons une démarche originale dans la définition du *modèle d'indexation par situations comportementales*. Notre modèle propose une solution générique pour l'indexation des cas pour la classe de problèmes étudiée. A la fin de cette partie, nous illustrons les notions utilisées sur l'exemple de la nutrition des plantes. L'utilisation concrète de ce modèle est facilitée par notre plate-forme à objets que nous présentons dans les chapitres III et IV. Le chapitre V donnera enfin une application complète de notre modèle d'indexation dans la réalisation de notre prototype d'assistant à la navigation sur le Web.

1 Approches existantes d'indexation par historiques

Dans le cadre de l'état de l'art sur l'indexation de connaissances par historiques, nous regroupons les approches existantes en deux groupes : d'une part les approches en RàPC et d'autre part celles liées à la reconnaissance d'historiques et aux systèmes de gestion de bases de données temporelles.

1.1 Approches basées sur le raisonnement à partir de cas

Peu de systèmes de RàPC ont abordé l'indexation par historiques qui nécessite la prise en compte d'une dimension temporelle dans les cas. Toutefois, les systèmes de raisonnement à partir de cas donnant une dimension temporelle à leurs cas n'ont pas tous recours à une indexation par historiques. Par exemple, dans certaines approches (Brandau *et al.,* 1991 ; Kopeikina *et al.*, 1988 ; Deter, 1994) l'observation d'un réseau de communication fait partie inhérente de l'application mais les historiques des observations n'ont pas été jugés pertinents pour l'indexation, bien que la dimension temporelle ait été prise en compte dans la description incrémentale du problème (Deter, 1994) ou dans la représentation de plans et de comportements attendus dans la solution des cas (Brandau *et al.,* 1991 ; Kopeikina *et al.*, 1988). Ainsi, nous présentons uniquement les approches dans lesquelles les historiques sont des indices, c'est-à-dire qu'ils déterminent dans quel cadre un cas est utile et applicable. Nous montrons alors les limites de ces systèmes dans l'objectif de la définition d'un modèle général d'indexation par historiques, adapté à la classe de problèmes visée.

1.1.1 Présentation des systèmes existants

Dans un premier temps, nous présentons plus en détail le système REBECAS (Rougegrez, 1994a) qui aborde la prévision de l'évolution des incendies de forêt et le système SINS (*Self-Improving Navigation System,* Ram & Santamaria, 1997) qui a été conçu dans le cadre de la commande d'un

robot mobile. Ces deux systèmes abordent des problèmes complexes et apportent des solutions complètes et originales. Nous présenterons ensuite brièvement le contexte et les objectifs des autres systèmes et approches ayant utilisé des historiques dans l'indexation des cas.

1.1.1.1 *Prévision de l'évolution des feux de forêts : le système REBECAS*

Le but du système REBECAS (Rougegrez, 1994a ; Rougegrez, 1994b) est de prédire l'évolution d'un processus complexe et en particulier de prédire l'évolution des incendies de forêt. Le domaine des feux de forêt est complexe car les paramètres agissant sur l'évolution du feu sont nombreux et leurs effets sont mal connus. De plus, la plupart des modèles de feux de forêt ne prennent pas en compte les évolutions de l'environnement (du vent par exemple). Le système REBECAS permet de prendre en compte ces évolutions sous la forme d'historiques d'événements pour essayer de donner une meilleure prévision. L'observation du feu se fait grâce à des relevés effectués sur le front du feu et à l'utilisation de cartes pour connaître le relief et la végétation parcourue. Les relevés ne sont pas systématiques, il peut donc y avoir une incertitude sur l'évolution entre deux relevés successifs et il n'est pas possible de savoir si un changement a été brusque ou progressif.

Dans REBECAS, un cas représente l'ensemble des données d'observation disponibles pour le déroulement complet d'un feu. Un cas est décomposé suivant plusieurs directions de propagation du feu. Pour chaque direction, un cas comprend les historiques (spatio-temporels) des paramètres pris en compte (relief, vent, végétation) et de la description de la propagation depuis son point de départ jusqu'à sa fin (cf. Figure II-1). Pour une direction donnée, un ensemble *d'événements* est généré pour chaque variable observée :

- *Relief* : un événement est un point donnant l'altitude et la distance au prochain point.
- *Végétation* : un événement donne le type de végétation (broussailles, conifères, vignes...) et son étendue (distance).
- *Vent* : un événement daté contient le lieu de la mesure ainsi que la direction et la vitesse du vent.
- *Propagation* : un événement daté est constitué notamment de la distance parcourue depuis le départ du feu.

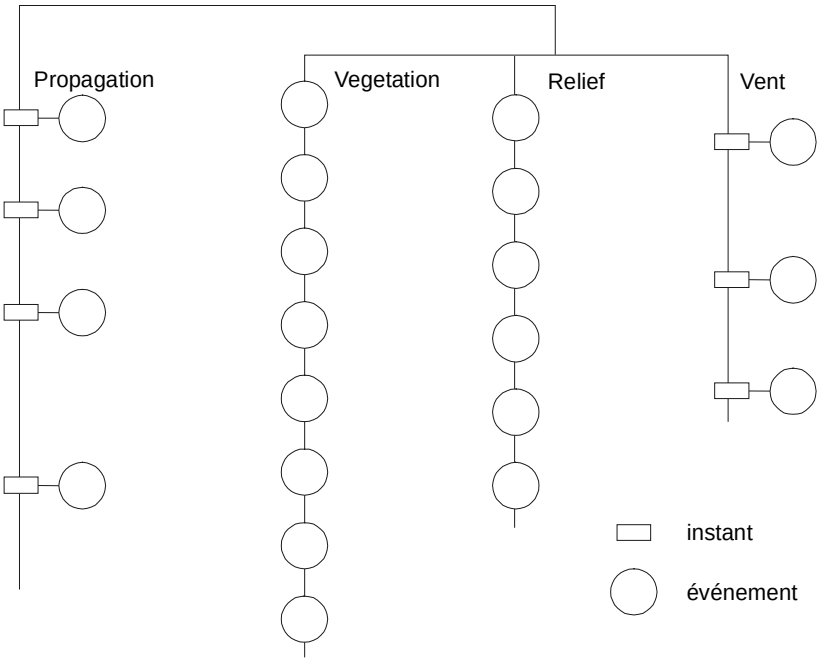


Figure II-1 : historiques d'un cas suivant une direction de propagation

Le système cherche à produire une prévision à chaque nouvelle entrée de données pour la direction de propagation correspondante. Le problème courant est alors constitué des différents historiques de l'évolution des variables depuis le départ du feu. La structure de la mémoire de cas est linéaire, et le but de la phase de recherche est d'identifier un cas qui, suivant une direction de propagation, présente les historiques les plus similaires à ceux du feu courant. Toutefois, selon l'auteur, la phase de recherche ne peut pas prendre en compte tous les paramètres simultanément car, d'une part il n'est pas possible d'agréger les comparaisons des variables de nature différente, et d'autre part la recherche serait trop restrictive pour retrouver une situation similaire. Ainsi, la recherche s'effectue d'abord selon chaque *point de vue* c'est-à-dire selon un paramètre (vent, végétation et relief). Les meilleures séquences identifiées dans chaque cas et dans chaque direction, suivant tous les points de vue, sont alors retournées. Puis, le meilleur cas est sélectionné grâce à une mesure cherchant notamment à minimiser le décalage temporel entre ces séquences (cf. Figure II-2).

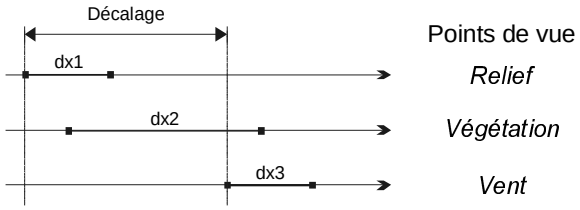


Figure II-2 : choix du cas minimisant le décalage

Pour la mise en correspondance des historiques, des algorithmes de comparaisons approximatives de chaînes de caractères sont utilisés après des étapes de transformation des événements. En effet, pour rendre cette comparaison possible et plus significative, une étape d'interprétation et une étape d'adéquation sont effectuées. L'étape d'interprétation du relief consiste, par exemple, à transformer les événements dans leur forme initiale, en une succession de formes de relief (crête, col, falaise...) grâce à l'utilisation d'un moteur d'inférence. Pour le vent, il s'agit d'exprimer les variations entre deux mesures successives plutôt que leurs valeurs absolues. L'étape d'adéquation consiste alors à préparer concrètement la mise en correspondance : vérification des longueurs des séquences à comparer et ajout de mesures hypothétiques pour compléter les séquences d'observation du vent par exemple. Puis la mise en correspondance est effectuée. Pour le relief et la végétation une distance est définie pour les événements, et une méthode de recherche de motifs approximatifs dans une chaîne de caractères est utilisée. Cette méthode permet de sélectionner la meilleure séquence en tenant compte du fait que les événements (pris comme des caractères) peuvent être inversés ou manquants. Pour le vent, le système calcule les distances pour la direction et la force pour chaque événement et agrège ces résultats en calculant leur somme, la plus petite somme étant alors sélectionnée.

Une fois le meilleur cas identifié suivant cet algorithme, la phase d'adaptation consiste d'abord à déterminer l'événement du cas sélectionné qui sera utilisé comme repère, puis les événements suivant ce repère sont transformés de manière à générer des événements de prévision dans le cas cible courant. Cette transformation permet de générer des événements qui gardent la même distance relative et la même durée relative. La phase d'apprentissage de REBECAS est effectuée manuellement par la description de nouveaux cas à partir des dossiers rédigés par les sapeurs pompiers.

1.1.1.2 *Commande de robots mobiles : le système SINS*

Le but du système SINS (*Self-Improving Navigation System*, Ram & Santamaria, 1993 ; Ram & Santamaria, 1997) est de commander de manière réactive un robot mobile dans un environnement réel. L'architecture générale est présentée dans la Figure II-3. Le module de navigation permet de contrôler le robot suivant des schémas de réaction comme « avancer vers le but », « éviter un obstacle », « mouvement aléatoire ». Le module d'apprentissage et d'adaptation dans lequel le raisonnement à partir de cas est effectué doit déterminer les meilleurs paramètres des schémas, étant

donné l'environnement courant. Contrairement à ACCBARR (Moorman & Ram, 1992), son prédécesseur, SINS prend en compte les historiques de l'évolution des paramètres décrivant une situation et possède un module d'apprentissage automatique.

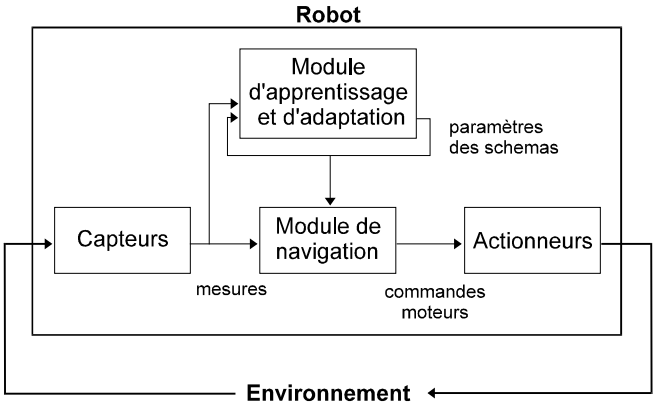


Figure II-3 : architecture de SINS

Dans l'observation du processus nécessaire au raisonnement à partir de cas, deux types de variables peuvent être identifiés : les variables de mesure décrivant l'environnement (par exemple la densité des obstacles et la position par rapport au but), et les variables représentant les paramètres des schémas de réaction. Les valeurs des différentes variables sont connues à des intervalles de temps fixes et le nouveau problème est également construit à intervalles de temps fixes. Un nouveau problème est représenté par les évolutions des variables pour une durée donnée. De manière identique, un cas source représente ces évolutions issues d'expériences passées pour une durée fixée. Basée sur une structure linéaire de la base de cas, la phase de recherche permet d'identifier le cas dont le recouvrement (cf. Figure II-4) avec le problème courant est optimum. Plus précisément, par une mise en correspondance progressive de chaque cas en remontant dans le temps, le meilleur recouvrement est identifié en utilisant une mesure de similarité. Toutes les valeurs étant numériques, la similarité repose sur le calcul des écarts quadratiques sur chaque variable, qui sont ensuite agrégés par une moyenne pondérée.

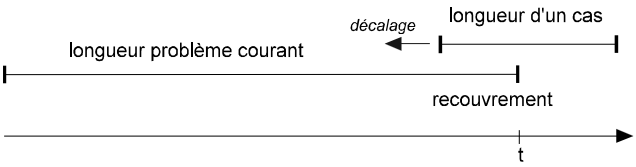


Figure II-4 : principes de la recherche dans SINS

La partie du cas non comprise dans le recouvrement est alors utilisée dans l'adaptation pour déterminer les séquences de réglages (cf. Figure II-5) en tenant compte de la similarité calculée dans la phase de recherche et également d'un facteur aléatoire permettant de modifier les paramètres pour explorer les réactions du système.

La phase d'apprentissage permet finalement d'exécuter deux types d'actions :

- Si le meilleur cas utilisé dans ce raisonnement présente une similarité trop éloignée du cas cible courant, SINS enregistre le cas cible courant dans sa mémoire. Le nombre de cas de la base peut toutefois être limité pour empêcher la création d'une base de cas trop importante.
- Dans le cas contraire, le cas source est modifié ou complété, sans sauvegarder le cas cible courant. Si les paramètres déterminés par le cas source ont été bénéfiques, le cas source est renforcé et/ou allongé dans le temps sinon il est modifié pour explorer les réactions du robot dans les raisonnements suivants. Dans tous les cas, le cas source devient alors un *cas*

virtuel dans lequel les données ne correspondent plus à de réelles observations puisqu'elles sont modifiées en tenant compte de l'évaluation de la réutilisation du cas source.

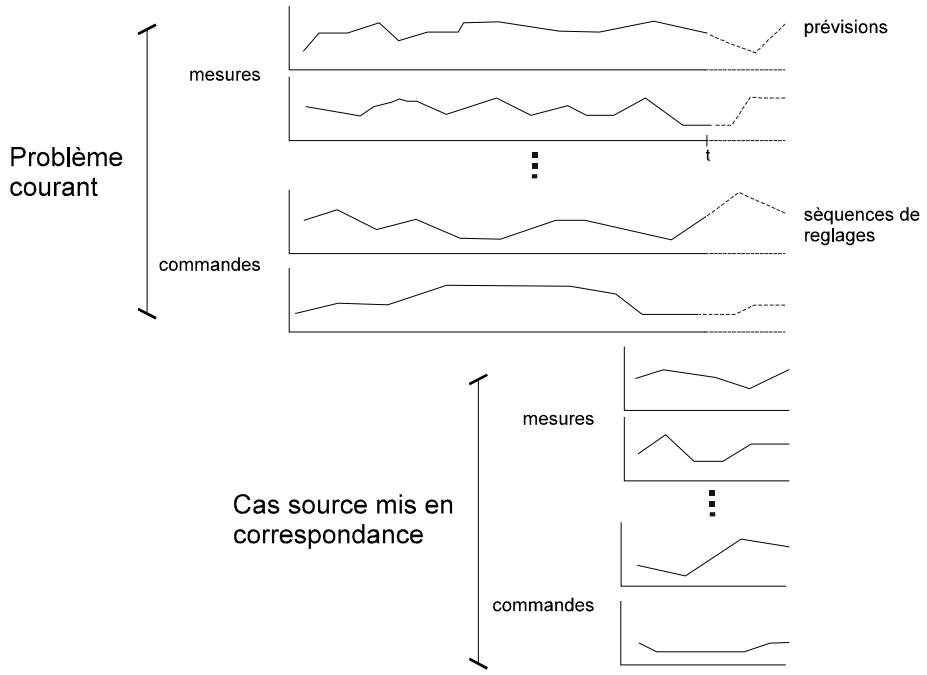


Figure II-5 : mise en correspondance dans SINS

1.1.1.3 *Autres systèmes et approches de RàPC*

Dans le domaine de la prévision de l'évolution de processus, l'approche de Nakhaeizadeh (Nakhaeizadeh, 1994) a été appliquée dans le domaine économique. Le système de Bull *et al* (Bull *et al.,* 1997) détecte des risques épidémiologiques et permet également la prévision de l'expansion des épidémies ou de leurs conséquences. Dans le domaine médical, le système ICONS (Schmidt *et al.,* 1996) effectue des estimations de l'évolution de la fonction rénale d'un patient. Le système PADIM « Poste d'Aide à la Décision Immédiate » (Fuchs *et al.,* 1995) aide un opérateur en charge de la supervision d'un procédé complexe en lui proposant les descriptions visuelles les plus adaptées. Enfin, le système RADIX (Corvaisier *et al.*, 1998) est un assistant de navigation sur le Web.

1.1.1.3.1 Prévision économique : l'approche de Nakhaeizadeh

L'approche de Nakhaeizadeh (Nakhaeizadeh, 1993) a été appliquée, dans le domaine économique, à la prévision de taux d'intérêt et à la prévision du nombre d'immatriculations de voitures et de camions en France. Une méthode générale est proposée pour la prévision de l'évolution de processus. Etant donné Y une variable échantillonnée, le système doit prévoir, à l'instant t , l'évolution de Y pour l'instant suivant $t+1$. En respectant, la structuration d'un cas en (*problème*, *solution*), un cas source est alors un couple $((Y_{s-k},...,Y_s),Y_{s+1})$ et le cas cible $((Y_{t-k},...,Y_t),Y^*)$ dans lequel Y^* doit être estimée. Le système recherche alors dans la base de cas (ensemble des séquences passées de l'évolution de la variable Y) en les comparant avec la séquence du problème courant avec un algorithme de plus proches voisins. La prévision peut alors être faite par adaptation de la solution de chaque cas identifié (avec une moyenne par exemple). Dans (Nakhaeizadeh, 1994), cette approche est généralisée dans le sens où la partie problème d'un cas est composée de toutes les valeurs ayant un impact significatif sur la variable à estimer : ceci inclut différentes valeurs, y compris les valeurs passées de Y.

1.1.1.3.2 Prévision épidémiologique : l'approche de Bull *et al.*

Le système de Bull *et al* (Bull *et al.,* 1997) permet de détecter des risques épidémiologiques et de prédire l'expansion des épidémies et leurs conséquences. L'entité d'observation est le *scénario*, noté $s \in S$, qui donne l'état de la santé publique et de la charge des services de santé pour une semaine et pour une localité. Une séquence de scénarios successifs est noté $\sigma = s_0\, s_1 \ldots s_n \in S^*$ avec $s_i \in S$ pour tout i. Une base de cas $\sum \in S^*$ est alors en ensemble de séquences de scénarios. Les cas sources n'ont pas d'existence concrète et représentent n'importe quelle sous-séquence de la base : $s\sigma \in \sum \Rightarrow \sigma \in \sum$ et $\sigma s \in \sum \Rightarrow \sigma \in \sum$.

Le problème du cas cible est alors exprimé par une séquence particulière, nommée σ. L'objectif du raisonnement est d'identifier la séquence la plus similaire σ' dont la longueur est plus grande que celle de σ.

$$\sigma' = u_0\, u_1 \ldots u_m\, u_{m+1} \ldots u_{m+k}$$

$$\downarrow$$

$$\sigma = s_0\, s_1 \ldots s_n \ldots\ldots\ldots$$

La séquence solution ($u_{m+1} \ldots u_{m+k}$) est utilisée par adaptation suivant l'objectif du raisonnement (détection ou prévision). L'approche générale reste très proche de celle de Nakhaeizadeh.

1.1.1.3.3 Prévision médicale : le système ICONS

Dans le domaine médical, le système ICONS (Schmidt *et al.,* 1996) permet de prévoir l'évolution de la fonction rénale d'un patient. Chaque jour, plusieurs dizaines de paramètres sont mesurées ou calculées pour représenter l'état détaillé d'un patient. Ces informations sont ensuite traitées pour obtenir une séquence de symboles donnant les états des sept derniers jours pour former le problème courant, comme par exemple la séquence suivante :

(*selective tubular damage, reduced kidney function, reduced kidney function, selective tubular damage, reduced kidney function, reduced kidney function, sharply reduced kidney function*).

L'objectif du système est alors de prévoir la suite de l'évolution. Cette représentation de base est alors analysée pour générer la représentation de l'état du patient suivant quatre *tendances* sur des horizons temporels différents : état courant, état suivant les deux derniers jours, état sur les sept derniers jours, et l'état moyen sur les sept derniers jours. Un cas source comprend une description de structure identique et donne comme solution, l'évolution effective des trois jours suivants. Le raisonnement effectue alors une phase de recherche suivant les neuf états possibles de l'évolution du patient. Pour chaque état possible, le cas le plus similaire suivant les différentes tendances est retourné. Les cas retrouvés sont alors présentés à l'utilisateur suivant leur degré de similarité, et donnent une évolution possible de l'état du patient. L'apprentissage est effectué par l'ajout de cas dans la base, et l'organisation de la mémoire qui repose sur une hiérarchie de prototypes est mise à jour.

1.1.1.3.4 Aide à la supervision de procédés : le système PADIM

Le système PADIM (Poste d'Aide à la Décision Immédiate, Fuchs *et al.,* 1995) aide un opérateur en charge de la supervision d'un procédé complexe en lui proposant les descriptions visuelles les plus adaptées. Dans un procédé industriel complexe, il existe de multiples façons de visualiser les informations de supervision, et comme il est impossible de tout visualiser, la sélection des

informations pertinentes est un problème important. Cette sélection est effectuée par PADIM en réutilisant des cas sources indexés par une situation composée :

- d'une partie instantanée donnant notamment l'état du système, le profil de l'utilisateur et le but de la supervision,
- et d'une partie comportementale regroupant les événements pertinents issus de l'évolution du procédé et des actions de l'opérateur.

Un cas source associe à cette situation un environnement de supervision (éléments graphiques) comme solution. Etant donné la situation courante, PADIM commence d'abord par rechercher le cas le plus similaire en effectuant un filtrage sur la partie instantanée puis une sélection en prenant en compte la partie comportementale (séquence d'événements). La mesure de similarité pour comparer deux séquences suppose que les événements de la séquence du cas cible se retrouvent tous et dans le même ordre dans la séquence d'un cas source. Cette mesure prend en compte la représentativité et la dispersion relative de deux séquences à comparer (cf. Figure II-6).

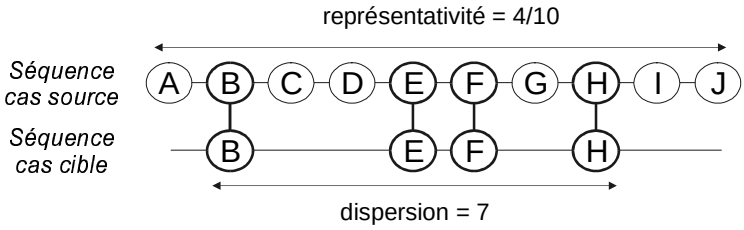


Figure II-6 : mise en correspondance de deux séquences dans PADIM

Suite à la phase de recherche, la disposition et le choix des éléments graphiques à utiliser sont déterminés par l'adaptation du cas source identifié, notamment en prenant en compte les résultats de la mise en correspondance et des connaissances spécifiques sur les différents éléments. L'utilisateur a alors la possibilité de réviser la solution proposée en la modifiant. Enfin, la phase d'apprentissage, contrôlée par un expert, permet d'enrichir la base de cas avec cette nouvelle expérience de supervision.

1.1.1.3.5 Assistance à la navigation : le système RADIX

Le système RADIX (Corvaisier *et al.*, 1998 ; Corvaisier *et al.*, 1997) est un assistant à la navigation sur le Web en cours de conception et n'intègre pas encore toutes les phases du raisonnement à partir de cas. Cependant, ce système aborde plus particulièrement les problèmes de représentation des sessions de navigation et de définition des mesures de similarité à utiliser pour la phase de recherche. Un cas représente une session de navigation qui est composée d'un ensemble ordonné de tentatives unitaires de recherche (TU). Chaque TU se décompose à son tour en une séquence d'unité de recherche (UR) reliant deux pages Web par une transition. Au sein d'une session, les changements de TU sont détectés par des actions spécifiques comme la sélection d'un signet, le choix d'un lien dans l'historique ou encore l'édition manuelle d'une adresse. La phase de recherche va permettre d'identifier des sessions qui présentent une TU similaire à la TU courante de l'utilisateur. Cette recherche commence par un filtrage sur la signature des TU formée du vocabulaire (ensemble de mots pertinents issus des pages visitées), du type d'information recherchée et du type de besoin d'information (précis, thématique ou exploratoire). Puis une phase de sélection est effectuée pour prendre en compte de manière plus précise : la séquence ordonnée de pages et de transitions, le type et le besoin d'information et la date de la TU. La mesure de similarité, utilisée pour la séquence de pages et de transitions, est identique au système PADIM (cf. Figure II-6). A terme, le système vise la réutilisation des sessions ainsi retrouvées, pour proposer à l'utilisateur des pages pertinentes à visiter, des libellés de requêtes et des listes de serveurs à interroger.

1.1.2 Limites des systèmes existants

Au regard de notre classe de problèmes et de notre démarche visant à faciliter la mise en place des représentations et des mécanismes liés à l'indexation, nous mettons en évidence trois limitations des approches existantes : l'absence d'approche générale pour l'indexation, l'insuffisance de la gestion des cas pour la découverte de connaissances et l'absence de la gestion d'indices basés sur des observations hybrides.

1.1.2.1 *Absence d'approche générale pour l'indexation*

Parmi les démarches prises pour l'indexation dans les travaux existants, nous pouvons identifier deux groupes :

- les approches spécifiques à un problème bien défini : ICONS pour la prévision de l'évolution de la fonction rénale d'un patient, PADIM pour l'aide à la supervision et RADIX pour l'assistance à la navigation,
- les approches présentant certains aspects réutilisables pour l'indexation :
  - REBECAS qui est appliqué à la prévision des incendies de forêt mais dont les contraintes de sa généralisation ont été brièvement explicitées (Rougegrez, 1994a, chapitre 6),
  - SINS dont l'approche dépasse la commande de robots pour aborder la classe des problèmes *continus* (Ram & Santamaria, 1997),
  - et enfin les approches de Bull *et al.* et de Nakhaeizadeh dont les principes simples peuvent être réutilisés dans d'autres domaines.

Toutefois, aucune des approches précédentes ne présente un modèle d'indexation générique. C'est à chacun d'extraire de ces travaux, les processus d'indexation généraux touchant la recherche, la réutilisation et l'apprentissage, tout en identifiant leurs conditions d'application. Enfin, aucun outil ni méthode n'est proposé pour mettre en pratique ces processus généraux.

1.1.2.2 *Gestion des cas insuffisante pour la découverte de connaissances*

La classe de problèmes visée est notamment caractérisée par une nécessité d'expliciter des nouvelles connaissances parmi les nombreuses données collectées. Par exemple, dans la nutrition des plantes, il s'agit de présenter aux utilisateurs des comportements provenant de cas enregistrés pour permettre d'enrichir les connaissances du domaine. La découverte de connaissances à l'aide d'un système de raisonnement à partir de cas est selon nous basée sur trois opérations importantes :

- extraction automatique d'enseignements (cas ou parties de cas) à partir des données brutes,
- mise à jour des enseignements suivant les raisonnements effectués,
- interprétation par des experts du domaine des connaissances issues des enseignements grâce à leurs indices.

Nous montrons dans cette section que les approches existantes ne permettent pas d'effectuer l'ensemble de ces opérations de manière satisfaisante sur le plan de la gestion des cas.

1.1.2.2.1 Extraction automatique d'enseignements à partir des données brutes

L'extraction automatique d'enseignements permet durant un raisonnement de préciser un ensemble d'indices effectifs pour une solution proposée. Les systèmes ICONS et PADIM ne permettent pas cette extraction et utilisent des indices effectifs figés avant le raisonnement. Par exemple, dans PADIM, l'historique des événements liés au système et à l'opérateur est explicitement associé au cas de supervision. Dans le système ICONS, l'expérience représentée par un cas contient une description comportementale donnant l'évolution de l'état du patient dans les sept jours précédents. Ces systèmes

ne permettent donc pas d'extraire automatiquement des comportements pertinents en fonction du raisonnement courant. Il se borne à réutiliser directement les comportements décrits dans les cas existants et explicites. De par le manque de connaissances et le nombre important de données brutes enregistrées, nous supposons qu'il n'est pas possible, dans le cadre de notre classe de problèmes, d'identifier de manière exhaustive tous les enseignements disponibles.

D'un autre côté, les cinq autres systèmes permettent cette extraction suivant deux méthodes différentes. Premièrement, dans SINS, REBECAS, et RADIX, un cas source sera réutilisé si une partie *variable* de sa description correspond au problème courant. Par exemple, dans REBECAS, un cas représente l'évolution d'un feu tout entier depuis son début jusqu'à sa fin. Un cas source est alors séparé dynamiquement en une partie qui est mise en correspondance (les indices effectifs) et une partie solution qui donnera par adaptation une prévision de l'évolution de l'incendie. De même, dans SINS, un cas est constitué des évolutions des attributs de l'environnement et des paramètres de commande pour une durée déterminée. Suivant la mise en correspondance, le cas source est également séparé dynamiquement en deux parties comme dans REBECAS (cf. Figure II-7). Deuxièmement, dans les approches de Bull *et al.* et de Nakhaeizadeh, un cas n'a pas d'existence propre au préalable, il est identifié dynamiquement avec ses indices effectifs au sein de la base de cas suivant le problème courant.

En définitive, nous pouvons définir deux types de situations d'un cas : les situations *non explicites* et les situations *explicites*. Une situation est dite non explicite lorsque les indices effectifs qui la composent doivent être identifiés dynamiquement pendant la phase de recherche, soit parmi les indices potentiels d'un cas, soit par l'identification d'un nouveau cas. Une situation est dite explicite lorsque ses indices effectifs sont connus avant le raisonnement.

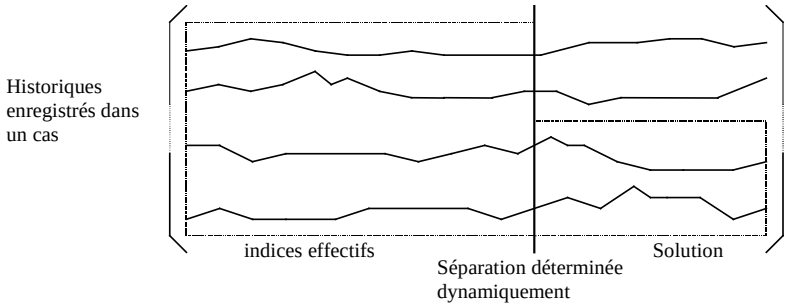


Figure II-7 : exemple de cas avec une situation non explicite dans SINS

1.1.2.2.2 Mise à jour des enseignements

A la suite de l'extraction des ces enseignements et à l'exécution d'un raisonnement qui a mené à un échec ou à un succès, la phase d'apprentissage doit pouvoir modifier les enseignements et leur organisation pour refléter ces résultats. Par exemple, l'échec du raisonnement peut provenir de la prise en compte d'indices effectifs non pertinents ou, au contraire, est révélateur du manque d'indices effectifs pertinents. Au niveau de la gestion des indices, il apparaît donc important de pouvoir modifier la sélection des indices effectifs parmi les indices potentiels d'un enseignement. Il est aussi important de pouvoir associer des indications sur la qualité de cet enseignement (nombres d'échec et de succès par exemple). Ces opérations sont primordiales pour mettre en évidence de nouvelles connaissances valides. Nous distinguons d'une part les systèmes qui n'effectuent aucune mise à jour, et d'autre part SINS qui opère des modifications.

Les systèmes REBECAS et RADIX ainsi que les approches de Bull *et al*. et de Nakhaeizadeh ne font aucune mise à jour des enseignements extraits et utilisés. Ce constat est très négatif sur la qualité et l'efficacité du raisonnement. Tout d'abord, la qualité du raisonnement progresse peu puisque

l'apprentissage se limite à l'ajout d'un nouveau cas. L'identification des enseignements qui auront mené à un échec ou à un succès ne sera pas prise en compte pour les raisonnements futurs. Ainsi, le même échec pourra se reproduire et un succès ne sera pas renforcé. Ensuite sur le plan de l'efficacité, l'identification des enseignements est souvent coûteuse, et si aucune information n'est retenue, il faudra exécuter à chaque fois les mêmes opérations. Par exemple dans REBECAS, l'extraction des enseignements dans les incendies passés et leurs utilisations dans le raisonnement ne mènent à aucune modification. Il n'est donc pas possible de mémoriser si une prévision fut un succès ou si les hypothèses de mise en correspondance étaient justifiées. En définitive, il n'est pas possible de mettre à jour ni les indices effectifs d'un enseignement, ni sa qualité. De plus, cette extraction est coûteuse car il s'agit de mettre en correspondance l'évolution du feu courant sur plusieurs paramètres (relief, vent et végétation). Ces opérations seront toujours à effectuer même pour un problème très similaire.

Seul SINS effectue une mise à jour à chaque raisonnement en modifiant les cas (cas virtuels). Ainsi, il est capable de mettre en évidence et de faire converger les cas vers des comportements plus généraux du processus étudié.

1.1.2.2.3 Interprétation des enseignements par analyse des indices

Après les deux premières opérations, l'interprétation des enseignements par des experts est nécessaire pour enrichir les connaissances du domaine. Le rôle des indices d'un enseignement est alors central dans cette interprétation. Les indices permettent en effet de définir le plus précisément possible le cadre d'application d'un enseignement, c'est-à-dire les conditions d'application de la connaissance que cet enseignement porte sur le processus observé.

SINS présente sur ce point des limitations. En effet, les mécanismes de modification de l'indexation sont basés sur des calculs numériques qui vont permettre de modifier le cas lui-même. Une partie du caractère explicatif des indices est alors perdue puisqu'il ne s'agit plus d'une expérience directement vécue. Une telle méthode pourra poser des problèmes d'intégration dans une application interactive par exemple. L'auteur du système précise lui-même que ces mécanismes rendent difficile une interprétation de plus haut niveau. Toutefois, cette interprétation n'était pas un objectif premier de SINS, le système cherchant en premier lieu l'efficacité du raisonnement.

1.1.2.2.4 Synthèse

	Extraction	***Mise à jour***	***Interprétation***
REBECAS (Rougegrez, 1994a)	✓	-	n/a
SINS (Ram & Santamaria, 1997)	✓	✓	-
Nakhaeizadeh (Nakhaeizadeh, 1994)	✓	-	n/a
Bull *et al*. (Bull *et al.,* 1997)	✓	-	n/a
ICONS (Schmidt *et al.,* 1996)	-	n/a	n/a
PADIM (Fuchs *et al.,* 1995)	-	n/a	n/a
RADIX (Corvaisier *et al.,* 1998)	✓	-	n/a

Tableau II-1 : gestion des cas et découverte de connaissances dans les approches existantes[19]

[19] La notation n/a indique qu'il n'est pas possible de comparer un système puisqu'il ne permet pas l'opération précédente.

Le Tableau II-1 donne la synthèse de notre analyse et répond aux questions suivantes pour chacun des travaux existants :

- L'approche permet-elle d'extraire automatiquement des enseignements ?
- Si oui, sont-ils mis à jour suivant les raisonnements ?
- Si des mises à jour sont effectuées, les enseignements gardent-ils leur caractère explicatif pour une interprétation ?

Suite à notre analyse, nous concluons qu'aucun des systèmes existants ne permet de manière satisfaisante la gestion des cas pour la découverte de connaissances.

1.1.2.3 *Absence d'indices basés sur des observations hybrides*

Deux types d'historiques peuvent être identifiés dans les travaux existants : les historiques *échantillonnés* et les historiques à *événements*. La majorité des systèmes ne gère que des historiques échantillonnés, c'est-à-dire que les valeurs de l'historique sont obtenues à une fréquence fixe. Par exemple, dans le système de Bull *et al*, les indices sont constitués d'une séquence de scénarios où chaque scénario donne l'état de la santé publique et de la charge des services de santé, pour chaque semaine. L'observation du système ICONS est basée sur plusieurs dizaines de paramètres qui sont traités pour obtenir l'état journalier d'un patient. Seuls PADIM, RADIX et REBECAS gèrent des historiques à événements, où un événement est un changement d'état ou une action considérée comme instantanée. PADIM intègre, dans la description de la situation, des séquences d'événements qui sont issus de l'évolution du procédé et des actions de l'opérateur. RADIX prend en compte les événements de navigations (changement de page). Dans REBECAS, le feu est observé grâce à des relevés effectués sur le front du feu, et les informations continues provenant de cartes sont transformées en historiques à événements.

Toutefois aucun système ne propose un modèle de représentation des indices *intégrant à la fois des historiques échantillonnés et à événements*. Cette possibilité est utile dans le cadre de la classe de problèmes visée pour deux raisons. Premièrement, le processus observé peut être hybride c'est-à-dire qu'il présente dans son fonctionnement à la fois des variables échantillonnées (humidité, rayonnement solaire) et des variables à événements (déclenchement du chauffage ou alarmes). Deuxièmement, dans le cadre d'un système de RàPC, il est important de représenter le comportement du processus de manière pertinente pour le raisonnement. Ainsi, même si toutes les variables de base sont échantillonnées, il peut être utile de transformer ces observations en événements de plus haut niveau pour former de meilleurs indices. Cependant toutes les mesures échantillonnées ne peuvent pas toujours être transformées en événements car cette transformation nécessite des connaissances spécifiques. Par exemple, REBECAS utilise un moteur d'inférence pour transformer des relevés de cartes topographiques en formes de reliefs, considérés par la suite comme des événements. L'absence de ces connaissances peut donc mener à l'étude de variables uniquement échantillonnées ou hybrides.

1.2 Autres approches

En dehors des approches de raisonnement à partir de cas, nous avons étudié deux autres groupes de travaux qui abordent l'indexation par historiques, au sens large, de connaissances ou de données : les approches de reconnaissance d'historique et les systèmes de gestion de bases de données temporelles. Nous présentons tout d'abord ces deux types d'approches puis nous faisons une synthèse montrant que leur problématique sur l'indexation est restreinte par rapport à celle nécessaire pour un cycle de raisonnement à partir de cas.

1.2.1 Reconnaissance d'historiques

Les approches de reconnaissance d'historiques mettent l'accent sur la mise en correspondance d'un historique source et d'un historique cible, et nous les classons en trois types :

- approches de traitement du signal qui opèrent sur des historiques purement numériques,
- approches syntaxiques qui utilisent des historiques formés de séquences de symboles,
- approches à base de scénarios temporels qui mettent l'accent sur la gestion de contraintes temporelles.

1.2.1.1 *Approches de traitement du signal*

Dans ce type d'approches, un historique est représenté par les valeurs successives prises par une variable numérique échantillonnée, et nous parlons alors de *signal numérique.* Ce signal est généralement issu d'un signal analogique qui aura été préalablement numérisé. Le problème de la reconnaissance d'historiques se pose alors dans ces termes : étant donné un signal cible, il s'agit de reconnaître son évolution par rapport à un ou plusieurs signaux ou modèles de signaux de référence (sources). De multiples approches sont proposées en fonction des propriétés du signal et des objectifs de la mise en correspondance. Une synthèse des fonctions de comparaison existantes est présentée dans (Bassaville, 1988).

Nous donnons dans la suite l'exemple d'un système qui a été appliqué à la reconnaissance d'un signal transitoire rapide (Charles, 1975). Soit le signal cible X = (x_1 ,...,x_I ,..., x_M) et un ensemble de signaux de référence S dont les éléments sont notés Y_i= (y_1 ,...,y_I ,..., y_N) avec M ≥ N. L'objectif du système est de reconnaître dans X, l'un des signaux de référence, sachant que les signaux de référence peuvent se reproduire dans X avec un facteur de proportionnalité et un décalage temporel quelconques. La reconnaissance s'effectue alors en trois étapes :

1. *Prétraitement*. Le maximum de X (position et valeur) et sa polarité sont déterminés.
2. *Sélection grossière*. Tout d'abord, seuls les signaux Y_i ayant la même polarité que X sont retenus. Puis, X est positionné avec chaque Y_i de manière à ce que leurs maximums respectifs coïncident. Le calcul du carré du coefficient de corrélation, noté ρ_i^2, entre chaque couple de signaux est alors effectué :

$$\rho_i^2 = \frac{\sum_{I=1}^{N} x_I \cdot y_I}{\sum_{I=1}^{N} x_I^2 \sum_{I=1}^{N} y_I^2}$$

   Ce coefficient est compris entre 0 et 1, et est égal à 1 lorsque deux signaux sont identiques, (avec un facteur de proportionnalité près). Enfin, seuls les signaux pour lesquels ce coefficient est supérieur à un seuil (0,9) sont gardés pour l'étape suivante de la reconnaissance. Si aucun coefficient est supérieur à ce seuil, la reconnaissance a échoué.
3. *Sélection définitive*. Il se peut que la mise en correspondance basée sur le maximum de chaque signal soit imprécise. C'est pourquoi, dans cette étape, chaque signal est déplacé autour de ce maximum. Le coefficient de corrélation est calculé pour chaque décalage J :

$$\rho_{iJ}^2 = \frac{\sum_{I=1}^{N} x_{I+J} \cdot y_I}{\sum_{I=1}^{N} x_{I+J}^2 \sum_{I=1}^{N} y_I^2}$$

   Dans l'expérimentation du système, J varie ente -6 et +6. Le plus grand de ces coefficients est alors associé à chaque signal restant. Enfin, le signal de référence dont le coefficient de corrélation est le plus grand, est donné comme résultat de la reconnaissance.

1.2.1.2 *Approches syntaxiques*

Dans ce type d'approches, un historique est représenté par une chaîne de caractères issus d'un alphabet. Dans cette modélisation, chaque caractère représente soit un événement soit une mesure du processus observé ou de son environnement. Le problème de la reconnaissance d'un historique revient alors à mettre en correspondance une chaîne de caractères source ou un type de chaîne de caractères, avec une chaîne de caractères cible.

Dans un premier temps, les données brutes, qu'elles soient numériques ou symboliques, issues d'échantillonnage ou de l'occurrence d'un événement, sont traitées pour former la chaîne de caractères cible. Par exemple, les données numériques échantillonnées issues de l'observation d'un électrocardiogramme humain peuvent être segmentées et transformées en une succession de caractères (cf. Figure II-8). De manière similaire, le système Patterns (Morrill, 1998) permet d'exprimer des primitives (signal constant, dépassement de seuils, signal concave ou convexe) et de les relier par des opérateurs (conjonction, disjonction, boucle, espacement) pour former un patron d'évolution à reconnaître.

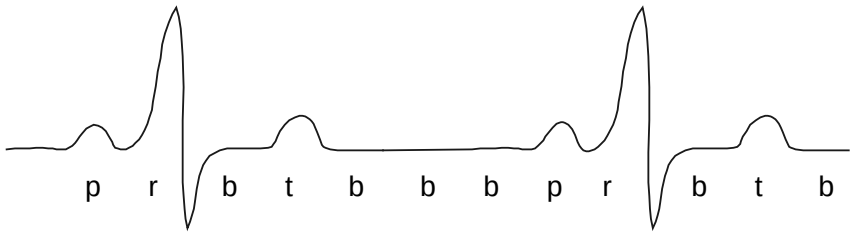


Figure II-8 : électrocardiogramme humain normal (Gonzalez, 1978)

Puis, pour la mise en correspondance, de nombreuses techniques sont applicables : mesure de similarité entre chaîne de caractères (Hall & Dowling, 1980) ou techniques issues de la théorie des langages formels (Gonzalez, 1978). Afin de présenter un exemple d'utilisation de cette dernière approche, nous rappelons brièvement les notions utilisées par cette théorie (alphabet, langage, grammaire et automates).

Un *alphabet*, noté Σ, est un ensemble fini de symboles. Sur cet alphabet, un *mot* ou *phrase* est une succession finie de symboles. Un mot a une longueur qui correspond au nombre de symboles qui le constituent, et ε est le mot vide. Si u et v sont deux mots, la concaténation de u et v, notée uv, est un opérateur qui forme un nouveau mot en juxtaposant les symboles de u puis les symboles de v. Σ^* est l'ensemble des mots pouvant être formés à partir de l'alphabet Σ, y compris le mot vide ε. Σ^* est un ensemble dénombrable et infini. Par exemple si $\Sigma=\{a,b\}$, a, b, ab, $abba$ sont des mots sur Σ. Un *langage* L est un ensemble de mots ou phrases sur un alphabet Σ, c'est un sous-ensemble de Σ^*. En reprenant l'exemple précédent, un langage peut être défini en extension par l'ensemble de mots $\{abba, baab, abbba, baaab\}$. Une grammaire $G=(N,\Sigma,P,S)$ définit les règles de syntaxe ou axiomes qui permettent de générer un ensemble de mots avec : N un ensemble fini de symboles non terminaux, Σ l'alphabet (symboles terminaux), P l'ensemble des axiomes, S élément de N le symbole de départ.

Le langage défini par une grammaire G est noté $L(G)$. Plusieurs types de grammaires existent suivant les restrictions sur les axiomes utilisés : grammaire sans restriction (type 0), grammaire contexte-lié (type 1), grammaire contexte libre (type 2), et grammaire régulière (type 3). Nous allons nous intéresser dans cette présentation uniquement aux grammaires régulières. Dans le cas d'une grammaire régulière à droite, les axiomes sont de la forme $A \rightarrow aB$ ou $A \rightarrow a$ où A et B sont éléments de N, et a est élément de Σ^*. Une fois la grammaire et le langage ainsi définis, le problème pratique

de la reconnaissance des mots d'un langage se pose. Suivant le type de la grammaire, une machine de reconnaissance particulière est utilisée :

- Type 0 : machine de Turing.
- Type 1 : automate linéaire (*Linear-bounded automaton*).
- Type 2 : automate à pile (*Push-down automaton*).
- Type 3 : automate fini (*Finite automaton*).

Ainsi pour le langage régulier généré par une grammaire régulière, nous pouvons utiliser un automate fini A que l'on définit par $A= (Q,\Sigma,\delta,q_0,F)$ avec : Q un ensemble d'états, Σ un alphabet, δ une fonction de transitions de $\Sigma\times Q$ dans Q, q_0 élément de Q l'état de départ, et F le sous-ensemble de Q des états finaux. Un mot $w \in \Sigma^*$ est accepté (reconnu) par un automate fini déterministe si et seulement si, après avoir lu le mot, l'automate se trouve dans un état final.

Prenons l'exemple introduit dans la Figure II-8 dans lequel il s'agit de reconnaître un historique normal et un historique anormal. A chaque symbole de l'alphabet $\Sigma=\{p,r,t,b\}$ est associée une évolution primitive de l'électrocardiogramme et la grammaire définit les juxtapositions valides de ces primitives. La grammaire régulière utilisée est $G=(\{S,A,B,C,D,E,H\},\Sigma,P,S)$ où P est défini par les axiomes suivants :

$S \rightarrow p\ A$	$A \rightarrow r\ B$	$B \rightarrow b\ C$
$C \rightarrow t\ D$	$D \rightarrow b$	$D \rightarrow b\ E$
$E \rightarrow b$	$E \rightarrow b\ H$	$E \rightarrow p\ A$
$H \rightarrow b$	$H \rightarrow b\ S$	$H \rightarrow p\ A$

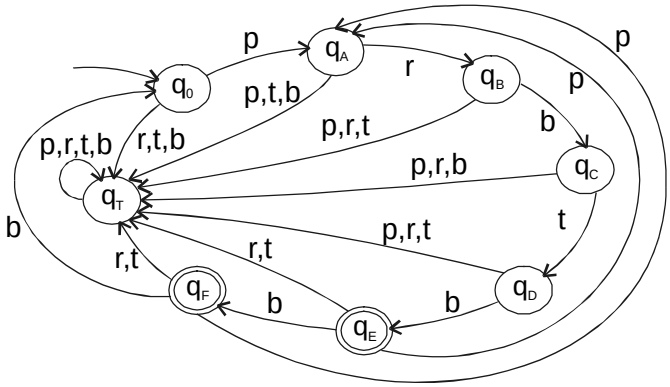


Figure II-9 : automate fini pour la reconnaissance d'un électrocardiogramme normal (Gonzalez, 1978)

L'automate fini permettant de reconnaître le langage L(G), c'est-à-dire l'évolution normale de l'électrocardiogramme, est donné dans la Figure II-9. L'état initial est q_0, les états finaux sont q_E et q_F c'est-à-dire que si l'automate est dans l'un de ces états, une évolution normale a été reconnue. En revanche, l'état q_T est un état « poubelle », dans le sens où dès qu'une évolution anormale est constatée, l'automate tombe dans cet état.

Nous avons présenté les techniques de base, mais dans le cadre de la reconnaissance de formes, d'autres méthodes plus complexes peuvent être utilisées comme des automates stochastiques prenant en compte les fréquences d'apparition des événements (i.e. des symboles de l'alphabet) pour donner plus d'importance à certains événements dans la reconnaissance, ou bien des automates permettant de reconnaître des phrases imparfaites (absence, insertion ou différence de symboles) (Gonzalez, 1978).

1.2.1.3 *Approches basées sur des scénarios temporels*

Dans ce type d'approches, un scénario est essentiellement constitué par un graphe de contraintes temporelles entre des observations qu'il s'agit de mettre en correspondance avec les observations effectives (Fontaine, 1996 ; Mounir-Alaoui, 1990 ; Haimowitz & Kohane, 1993a). Pour illustrer ce type d'approches, nous prenons l'exemple de deux systèmes représentatifs IxTeT (Mounir-Alaoui, 1990 ; Dousson & Ghallab, 1994) et $TrenD_x$ (Haimowitz & Kohane, 1993a ; Haimowitz & Kohane, 1993b).

1.2.1.3.1 Le système IxTeT

Le système IxTeT (Mounir-Alaoui, 1990) est un système de gestion de contraintes temporelles qui a été appliqué dans le cadre de la reconnaissance d'historiques. Plus précisément, ce système a été utilisé dans des problèmes de surveillance d'un environnement bien structuré comme un atelier, un parking ou un aéroport. Puis, basée sur les mêmes principes, l'application du système IxTeT a été étendue à la supervision de processus continus (Dousson & Ghallab, 1994) : supervision de turbines à gaz et filtrage d'alarmes dans une unité pétrochimique. Nous présentons ce système en montrant son fonctionnement sur un exemple concret (Mounir-Alaoui, 1990), dans lequel il s'agit de reconnaître le comportement d'un convoyeur dans une usine automatisée. Ce convoyeur doit entrer dans une pièce, effectuer un chargement puis ressortir (cf. Figure II-10).

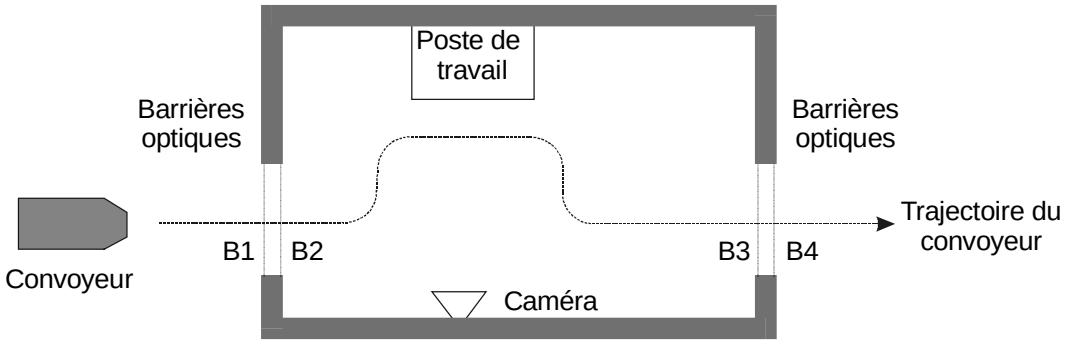


Figure II-10 : déplacement d'un convoyeur dans une usine automatisée (Mounir-Alaoui, 1990)

```
(Situation (S1)
        (Uses
                (fact f1 (on barrier 1) 1)
                (fact f2 (on barrier 2) 1)
                (fact f3 (on inroom 1 conv) 0.8 focus)
                (event e1 (on addweight) 1)
                (fact f4 (on barrier 3) 1)
                (fact f5 (on barrier 4) 1))
        (Such That
                (f1 o f2) (f2 o f3) (e1 o f3) (f4 o f5)
                ((duration f1) < 5 otherwise continue print « Problem at B1 »)
                ((duration f2) < 5 otherwise continue print « Problem at B2 »)
                ((duration f4) < 5 otherwise continue print « Problem at B3 »)
                ((duration f5) < 5 otherwise continue print « Problem at B4 »)
                (20 < (duration f3) < 60 otherwise exit event (on failure))
                ((e1 - (start f2)) < 35 otherwise exit event (on problem))
                (skip f3 if (f1 f2 e1) print « Camera failure »)))
```

Figure II-11 : définition d'une situation dans IxTeT

IxTeT permet de représenter des comportements du convoyeur, appelés situations, en reliant par des contraintes temporelles des faits et des événements. Un fait est une association entre une proposition et un intervalle de temps pendant lequel cette proposition est vraie. Un événement

représente le changement daté de l'état d'une proposition. Une situation est alors décrite dans une syntaxe qui sépare la définition des faits et des événements (clause Uses), des contraintes liant ces faits et événements pour caractériser la situation (clause Such That). La Figure II-11 donne l'exemple d'une situation représentant le comportement normal du convoyeur.

Dans cette situation, les faits f1, f2, f4 et f5 représentent respectivement, le passage du convoyeur au travers des barrières optiques 1,2,3 et 4. Le fait f3 représente la présence du convoyeur dans la pièce qui est détectée par une caméra. Le mot clef focus indique que ce fait est très pertinent dans la reconnaissance de cette situation, et le facteur de 0.8 représente le degré de confiance des observations liées à la caméra. Enfin, l'événement e1 est transmis par un détecteur du poste de travail pour indiquer que le transfert du chargement a été effectué. Ensuite, plusieurs contraintes temporelles sont posées. Premièrement, cette situation définit des contraintes de chevauchement (relation o pour « *overlaps* ») : par exemple, le franchissement de la barrière 2 (fait f2) doit débuter alors que le franchissement de la barrière 1 (fait f1) n'est pas encore fini. Puis, des contraintes sur les durées des faits imposent que le franchissement des barrières optiques (faits f1, f2, f4 et f5) se déroule en moins de 5 secondes. La présence du convoyeur (fait f3) doit avoir une durée comprise entre 20 et 60 secondes, et le début du franchissement de la deuxième barrière ne doit pas être séparé de plus de 35 secondes du transfert du chargement. Enfin, une clause skip permet de ne pas attendre le déroulement du fait f3, si les faits f1 et f2 ainsi que l'événement e1 se sont produits. Ainsi, la syntaxe utilisée est très flexible, et permet de manipuler des événements aussi bien que des faits, ainsi que des contraintes temporelles symboliques ou numériques.

Ce système permet la définition de plusieurs situations à reconnaître qui sont d'abord compilées pour vérifier la cohérence des contraintes et pour construire des structures de données qui permettront de simplifier le processus de reconnaissance (Ghallab & Mounir-Alaoui, 1989). De plus, même si l'expression des situations utilise à la fois des événements et des faits, le système transforme les faits en deux événements (événement de début et de fin) pour gérer de manière plus efficace uniquement un ensemble d'événements. Le système intègre également plusieurs filtres opérant en amont de la reconnaissance proprement dite : filtres temporels pour assurer que les événements sont triés suivant leur date d'occurrence et non suivant leur ordre d'arrivée, et un moteur d'inférence temps réel permettant de générer des événements additionnels suivant des axiomes du domaine.

La reconnaissance se fait alors de manière incrémentale, au fur et à mesure que les événements se produisent. A l'arrivée d'un nouvel événement, trois cas peuvent se présenter :

- s'il fait partie de la description d'une situation et que son occurrence est compatible avec les contraintes de celle-ci, la reconnaissance se poursuit,
- s'il fait partie de la description d'une situation mais que des contraintes sont violées, la situation en cours de reconnaissance peut éventuellement être rejetée,
- s'il ne fait partie d'aucune description de situation, il est rejeté.

Ainsi, durant cette analyse en continue, des situations sont reconnues et les connaissances qui y sont associées permettent de déclencher une action appropriée (alarme par exemple). Le système est conçu pour pouvoir fonctionner dans le cadre d'une application comportant une douzaine de descriptions de situations et quelques dizaines de types d'événements, avec un temps de réponse inférieur à la seconde.

1.2.1.3.2 Le système $TrenD_x$

$TrenD_x$ est un système de reconnaissance d'historiques fondés sur la représentation et la gestion de *patrons de tendance* (*trend templates*) (Haimowitz & Kohane, 1993a ; Haimowitz & Kohane, 1993b). Bien que l'approche soit plus générale, le système a été étudié et évalué dans le domaine médical, et en particulier sur le diagnostic des troubles de croissance chez l'enfant. Une *tendance* est un motif

d'évolution cliniquement pertinent, issu d'une séquence de données observées et ordonnées chronologiquement. De plus, ces tendances peuvent prendre en compte de multiples variables d'observation. Un patron de tendance définit alors un type de tendance auquel est associée une hypothèse de diagnostic. De manière générale, un patron de tendance est composé de quatre parties :

- informations liées à ce patron (type de diagnostic par exemple).
- ensemble de points de repère et d'intervalles sur une échelle temporelle. Ces points de repère sont imprécis et possèdent une tolérance.
- ensemble de contraintes de valeurs portant sur des variables d'observation pour un intervalle particulier. Une contrainte est de la forme :

$$m \leq f(D) \leq M$$

  où D est l'ensemble des données d'observation lié à un intervalle, f représente une fonction à valeurs réelles, m et M sont des nombres réels ou sont égaux à l'infini.
- ensemble de liens avec d'autres patrons qui seront activés par l'échec d'une reconnaissance.

La Figure II-12 donne un extrait de la description d'un patron de tendances pour la reconnaissance de l'évolution normale de la croissance chez un enfant mâle. Trois points de repère sont définis : naissance, début de la puberté et arrêt de la croissance. Puis trois intervalles temporels sont définis (Int1, Int2 et Int3) permettant d'identifier des périodes pertinentes pour la description du patron. Sur ces intervalles, différentes contraintes de valeurs sont alors posées. Sur l'intervalle Int1, une première contrainte indique que le poids et la taille doivent varier d'une façon similaire. Puis, sur l'intervalle int2, les écarts types de vitesse de variation du poids et de la taille doivent rester proche de zéro. Enfin, sur l'intervalle Int3, d'autres contraintes sont posées sur l'âge osseux, les hormones de croissance (GH) et les hormones de la thyroïde (T4). Des liens entre ces contraintes et d'autres patrons permettront d'aiguiller la reconnaissance vers d'autres hypothèses envisageables dans le cas de leur violation. Par exemple, si la contrainte portant sur la taille durant l'intervalle Int2 n'est pas satisfaite, le patron pour diagnostiquer une puberté en retard sera étudié.

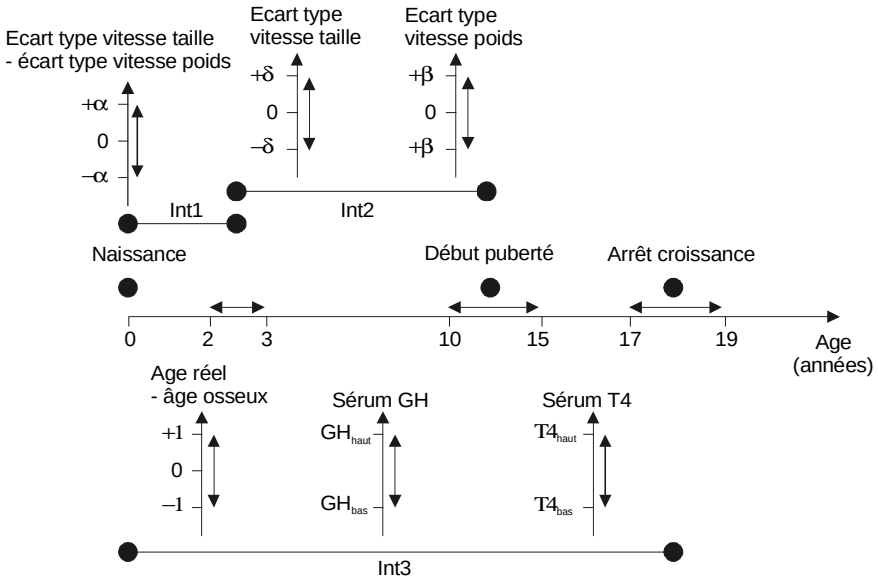


Figure II-12 : extrait d'un patron de tendances (Haimowitz & Kohane, 1993b)

Pratiquement, $TrenD_x$ permet d'établir un diagnostic en mettant en correspondance les données disponibles du patient avec un ensemble de patrons. Le système gère un ensemble d'hypothèses issues de l'instanciation de différents patrons. Chaque patron peut également être instancié plusieurs fois,

s'il y a différentes façons de vérifier les contraintes. L'instanciation de nouvelles hypothèses est guidée par les violations des contraintes d'un patron.

1.2.2 Bases de données temporelles et gestion d'historiques

Les bases de données conventionnelles ont été conçues pour stocker et manipuler les données les plus récentes à propos d'une organisation ou d'une activité : au fur et à mesure que les données sont mises à jour, les anciennes données sont perdues. Ce type de bases de données, appelé bases de données instantanées (*snapshot databases*), ne sont pas adaptées pour la gestion des données liées au passé, au présent et au futur, qui est nécessaire pour la prise en compte d'historiques. Les bases de données temporelles abordent cette problématique. Nous décrivons tout d'abord les axes de recherches sur les bases de données temporelles spécialisées pour la gestion d'historiques (aussi appelés *chroniques*). Nous présentons ensuite un exemple récent d'un de ces systèmes, nommé TEMPOS (Canavaggio, 1997).

1.2.2.1 *Systèmes de gestion de bases de chroniques*

Les systèmes de gestion de bases de données de chroniques ou TSMS (*Time Series Management System*) forment un domaine particulier au sein des recherches en bases de données temporelles et présentent de nombreuses spécificités (Segev *et al.,* 1995), dont notamment :

- la gestion de la notion de groupe de chroniques,
- l'association de calendriers aux chroniques,
- la nécessité d'opérateurs pour des requêtes d'agrégation et de statistique.

Une analyse précise des besoins pour la gestion de ce type de données est menée dans (Dreyer *et al.,* 1994) et la plupart des systèmes et approches existants pour les bases de données temporelles ne sont pas adaptés à ces spécificités. Seuls les travaux décrits dans (Segev & Shoshani, 1993 ; Segev & Chandra, 1993 ; Canavaggio, 1997) proposent un modèle de chroniques et un ensemble d'opérateurs spécifiques. Dans le premier modèle, décrit dans (Segev & Shoshani, 1993), une chronique ou TS (*Time Sequence*) représente un ensemble ordonné de couples (t,a), dans lequel t représente le temps et a la valeur d'un attribut. Un TSC (*Time Series Collection*) regroupe un ensemble de chroniques homogènes (par exemple les historiques des salaires des employés d'une société). Un ensemble d'opérateurs est alors défini pour la manipulation des TSC : sélection, agrégation, accumulation, restriction et composition.

Ce modèle a été étendu par la suite (Segev & Chandra, 1993) notamment avec la notion de *concept*. Dans ce modèle, une chronique est associée à un ensemble d'informations qui peut être étendu par l'utilisateur :

- le nom de la chronique,
- le calendrier définissant un ensemble d'instants pour lesquels la chronique doit avoir une valeur,
- un ensemble d'instants d'exception issus du calendrier pour lesquels la chronique ne peut fournir de valeur,
- la période (*life span*) donnant les instants de début et de fin de la chronique,
- le mode de mise à jour de la chronique (chronique dérivée d'autres chroniques ou issue de données brutes),
- la fréquence de mise à jour (par exemple une fois par mois),
- le type de la chronique (cf. Figure II-13) : en *escalier* (la dernière valeur est valide jusqu'à la prochaine mesure), *discrète* (entre deux mesures, la valeur ne peut être calculée ou n'a pas de sens), *continue* (une fonction d'interpolation est utilisée pour obtenir la valeur de la chronique entre deux mesures), ou alors définie par l'utilisateur.

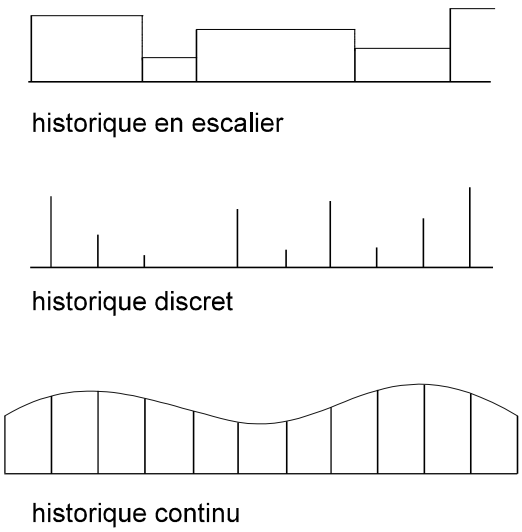


Figure II-13 : types de chroniques (Segev & Shoshani, 1993)

La notion de *concept* permet alors de grouper des chroniques non homogènes et de créer des *concepts complexes* au travers d'une hiérarchie d'héritage. De plus, une même chronique peut appartenir à plusieurs *concepts*. Des opérateurs spécifiques aux chroniques sont également proposés : moyenne mobile, auto-régression, agrégation (par exemple minimum, maximum et moyenne sur une période et/ou suivant une propriété), conversion (par exemple passage d'une chronique de fréquence journalière à une fréquence mensuelle).

1.2.2.2 *Le système TEMPOS*

TEMPOS « *Temporal Extension Model for Persistent Object Servers* » est un système basé sur le SGBD O_2, qui permet notamment la gestion multigranulaire des valeurs temporelles et la manipulation d'historiques (Fauvet *et al.,* 1997 ; Canavaggio, 1997). Comme dans la plupart des systèmes d'information, TEMPOS est basé sur un modèle granulaire du temps : le temps est représenté par la droite des réels qui est discrétisée par une partition en une suite d'intervalles consécutifs disjoints. Le *chronon* est l'unité la plus petite, c'est un intervalle de temps non décomposable et de durée fixe. D'autres unités peuvent alors être définies par une partition de l'ensemble des chronons. Dans TEMPOS, l'ensemble des unités est muni d'une relation d'ordre partiel « est plus fine que » (cf. Figure II-14) et des opérateurs de conversion et de manipulation ont été définis. Les différentes unités peuvent alors être utilisées dans les requêtes et dans la représentation des historiques.

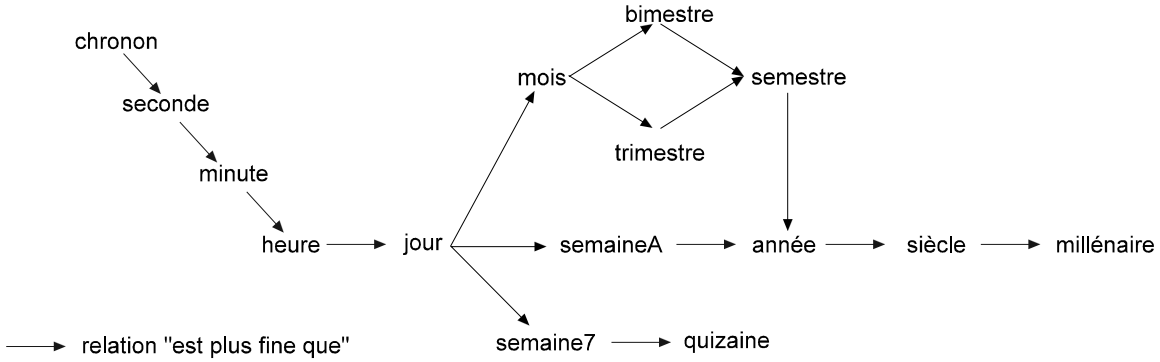


Figure II-14 : différentes unités de temps dans TEMPOS (Canavaggio, 1997, page 48)

Dans TEMPOS, un *instantané* définit un couple <M,V> où M est un *moment* et V une valeur d'un attribut au moment M. Un moment désigne soit un instant, soit un intervalle d'instants, soit encore

une séquence d'intervalles discontinus. Un historique est défini sur un domaine temporel qui peut être décomposé en deux sous-ensembles : le *domaine temporel effectif* et le *domaine temporel potentiel*. Le domaine temporel effectif est constitué de tous les instants pour lesquels une saisie de données a été effectuée. Le domaine temporel potentiel est la différence du domaine temporel avec le domaine temporel effectif. TEMPOS distingue alors deux types principaux d'historiques (cf. Figure II-15) : les historiques *en extension* et les historiques *en compréhension.* Un historique en extension est appelé *chronique* et son domaine temporel est égal à son domaine effectif. Trois types de chroniques sont définis suivant la notion de moment utilisé, par exemple une I_chronique est une séquence de couples associant un instant à une valeur :

I_chronique : [<1,v1>,<2,v1>,<3,v1>,<4,v2>,<5,v2>,<7,v3>,<8,v1>,<9,v1>]
X_chonique : [<[1…3], v1>,<[4…5],v2>,<[7…7],v3>,<[8…9],v1>]
D_chronique : [<[[1…3],[8…9]], v1>,<[[4…5]],v2>,<[[7…7]],v3>]

Un historique en compréhension associe une chronique et une fonction d'interpolation. La fonction d'interpolation est alors utilisée pour calculer la valeur de l'historique sur le domaine temporel potentiel à partir des instantanés du domaine effectif. Les historiques en escalier et discret sont traités comme des cas particuliers.

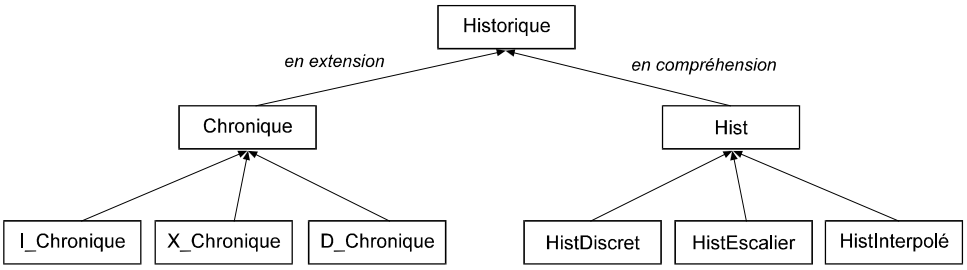


Figure II-15 : hiérarchie de types d'historiques dans TEMPOS (Fauvet *et al.*, 1997)

Le regroupement d'un ensemble d'historiques peut alors être effectué par la définition d'une classe d'objets dans laquelle des attributs constants, fugitifs (seule la dernière valeur est sauvegardée) et historiques peuvent se côtoyer :

```
Classe Employé <
        nom : Constant (string),
        adresse : Historique (jour, string),
        salaire : Historique (mois,real),
        dateEmbauche : Fugitif (Instant(jour)),
        affectation : Historique (mois, Rayon)>
```

Enfin, TEMPOS définit un ensemble d'opérations sur les historiques :

- opérateurs temporels de l'algèbre relationnelle (projection temporelle, restriction temporelle et produit naturel),
- fonctions générales sur les historiques (application d'une fonction aux éléments de l'historique, sélection suivant un prédicat, découpage en deux historiques suivant un élément, construction des couples consécutifs).

1.2.3 Limites de la reconnaissance d'historiques et des bases de données temporelles

Les approches de reconnaissances d'historiques et les systèmes de gestion de bases de données temporelles n'abordent pas l'intégralité de la problématique de l'indexation telle qu'elle est nécessaire durant un cycle de raisonnement à partir de cas. Pour effectuer la comparaison des approches, nous utilisons un vocabulaire commun issu du RàPC (phases de recherche, réutilisation,

apprentissage) mais qui n'est pas utilisé à l'origine dans ces travaux. Le Tableau II-2 donne la synthèse de cette comparaison.

	Raisonnement à partir de cas	***Reconnaissance d'historiques***	***Bases de données temporelles***
Représentation des indices	✓	✓	✓
Mise en correspondance des indices (recherche)	✓	✓	✓ (requêtes)
Utilisation de la mise en correspondance (réutilisation)	✓	Pas d'analyse des indices	Pas d'analyse des indices
Modification de l'indexation (apprentissage)	✓	Ajout, pas de modification des indices	Ajout, pas de modification des indices

Tableau II-2 : synthèse des approches d'indexation

La reconnaissance d'historiques met l'accent sur la mise en correspondance, et les mécanismes utilisés peuvent être présentés sous la forme d'un raisonnement dont les étapes ne sont généralement pas séquentielles :

- *Recherche* : la mise en correspondance de l'historique cible avec un historique source, ou un type d'historiques sources, est effectuée et donne un résultat généralement booléen.
- *Réutilisation* : la réutilisation se borne à retourner les informations attachées à l'historique source (diagnostic par exemple) sans analyser les différences entre des historiques sources et cible.
- *Apprentissage* : l'apprentissage est uniquement effectué par l'ajout d'un nouvel historique, mais il n'y a jamais modification automatique des historiques existants.

Ces mécanismes sont insuffisants pour pouvoir gérer les indices des cas et dénotent en fait une différence d'approche plus profonde. De manière générale, les approches de reconnaissance d'historiques supposent leurs connaissances exactes et n'ont donc pas le besoin de les réviser comme dans un système de RàPC.

Les approches liées aux bases de données temporelles offrent des techniques intéressantes pour le stockage des historiques et leur manipulation par un ensemble d'opérateurs. Cependant ces systèmes n'abordent pas non plus complètement la problématique de l'indexation par historiques et le raisonnement est réduit aux mécanismes suivants :

- *Recherche* : la recherche correspond à l'exécution d'une requête qui ne retourne que les historiques répondant strictement aux contraintes exprimées,
- *Réutilisation* : la réutilisation se borne à retourner les informations attachées à l'historique source.
- *Apprentissage* : l'apprentissage est uniquement effectué par l'ajout d'un nouvel historique.

2 Proposition du « modèle d'indexation par situations comportementales »

Pour faciliter l'application de solutions récurrentes d'indexation dans le cadre de la classe de problèmes visée, nous proposons le *modèle d'indexation par situations comportementales*. Ce modèle comprend :

- le modèle de représentation proprement dit des indices et des cas,
- et un guide d'utilisation présentant certains principes de gestion des indices et des cas pour la conception d'un cycle de raisonnement.

Nous introduisons tout d'abord le modèle de représentation qui définit formellement la structuration des indices et leur intégration dans les cas. Puis, nous présentons le guide d'utilisation qui formule la synthèse des problématiques d'indexation abordées pour la nutrition des plantes et pour l'assistance à la navigation sur le Web. Ce guide a pour objectif d'une part de définir un cadre unifié qui nous servira à présenter les applications réalisées, et d'autre part de fournir une base méthodologique pour l'utilisation du modèle dans d'autres applications. Nous illustrons enfin notre modèle d'indexation avec la spécification d'un système de recherche des cas pour la nutrition des plantes. Dans cet exemple, nous vérifions également que les concepts introduits permettent de prendre en compte les caractéristiques de cette application.

2.1 Modèle de représentation des indices et des cas

Notre modèle de représentation s'appuie tout d'abord sur des hypothèses que nous précisons. Nous donnons alors une brève description du modèle pour introduire les principaux concepts. Nous décrivons enfin de manière détaillée la représentation des données d'observation, des situations comportementales et des cas.

2.1.1 Hypothèses

Les hypothèses prises par notre modèle de représentation reprennent les cinq caractéristiques déjà introduites de la classe de problèmes visée (cf. chapitre I, §3.1). Brièvement, nous rappelons que la prise en compte d'historiques est choisie ou nécessaire pour formuler les problèmes dans lesquels un processus dynamique est observé. Cette observation mène à la collecte d'un ensemble important de données brutes par échantillonnage et/ou lors d'événements. Enfin, les connaissances du domaine sont réduites et une meilleure compréhension est recherchée.

La prise en compte de ces caractéristiques se traduit au niveau des indices par deux hypothèses supplémentaires. En effet, dû au manque de connaissances du domaine et à la nécessité de prendre en compte les historiques, nous supposons que :

- Les indices effectifs d'un cas source ne peuvent pas être décrits par un ensemble de descripteurs toujours identiques. Les indices effectifs doivent principalement être extraits des historiques passés des variables d'observation, dont les valeurs sont considérées comme des indices potentiels.
- Les indices effectifs d'un cas cible ne peuvent pas être définis *a priori* : c'est au raisonnement d'analyser le cas cible pour déterminer les cas sources applicables en fonction de leurs propres indices effectifs.

2.1.2 Présentation générale

Notre modèle repose sur une représentation à deux niveaux :

- Représentation des *données brutes d'observation* sous la forme de chroniques correspondant aux évolutions des variables d'observation du processus, qui sont alors regroupées dans un enregistrement pour une période de temps définie et pour un contexte donné.
- Représentation des *connaissances* sous la forme de cas qui associent la définition d'un problème à l'expression d'une solution spécifique à la tâche de résolution de problèmes à accomplir. Le problème comprend une situation comportementale référençant les données brutes des enregistrements.

Cette séparation permet une meilleure gestion des données et des indices. Premièrement, la définition des situations est modifiable par simple changement de la sélection des données brutes au sein des enregistrements. Deuxièmement, la mise en œuvre de cette séparation évite la duplication des données brutes dans les situations qui peuvent se recouvrir dans un même enregistrement.

Une situation comportementale est constituée d'une *composante instantanée* et d'une *composante comportementale*. La composante instantanée contient tous les indices instantanés liés au cas. En particulier, cette composante contient le contexte de l'enregistrement référencé. La composante comportementale constitue le centre de notre modèle d'indexation, d'où le nom de « situation comportementale ». Cette composante détermine à partir d'un instant de référence les indices effectifs d'un cas, parmi les indices potentiels correspondants aux données issues des enregistrements. Plus précisément, cette composante définit un ensemble de *comportements élémentaires* pour chaque variable, ainsi que des *contraintes temporelles* entre ces comportements. Un comportement élémentaire sélectionne une sous-séquence d'une chronique. Chaque comportement élémentaire possède une contrainte de localisation qui permet d'abstraire la raison pour laquelle ce comportement est sélectionné, par exemple : à cause des valeurs de la séquence, de sa position temporelle ou de sa position en terme du nombre d'événements par rapport à l'instant de référence. Ce formalisme de représentation permet d'exprimer des situations simples ou complexes obtenues par apprentissage ou provenant des connaissances du domaine :

- situation simple : « La situation *A* est définie à l'instant t_0 par les cinq dernières valeurs de toutes les chroniques ». Dans ce cas, les comportements élémentaires sont définis avec une contrainte de localisation par position et il n'y a pas de contraintes temporelles.
- situation complexe (Figure II-16) : « La situation *B* est définie à l'instant t_1 par les valeurs de la variable V_1 dans les 30 dernières secondes (b_1), par une forte augmentation des valeurs de la variable V_2 (b_2), et par les occurrences des événements α (b_3) et β (b_4) sur la variable V_3 ; α précède β (c_1) et β se produit pendant l'augmentation de V_2 (c_2) ». Cette situation montre également comment il est possible de prendre en compte des chroniques échantillonnées (V_2) et à événements (V_1 et V_3) grâce aux comportements élémentaires.

Une situation comportementale est alors utilisée pour déterminer les conditions de réutilisabilité d'un cas. Nous proposons dans notre modèle l'utilisation de deux types de cas permettant de représenter deux types d'enseignements :

- *cas potentiels* : enseignements basés sur des hypothèses préétablies.
- *cas concrets* : enseignements spécifiques, révisés ou atypiques.

Les cas potentiels, que nous introduisons dans nos travaux, n'ont pas d'existence explicite mais sont représentés en compréhension par un *patron*. Un patron définit des règles de construction d'un cas pour un enregistrement et un instant de référence donnés. Il définit en particulier les règles de construction de la situation comportementale. Le résultat de l'instanciation d'un patron est aussi

appelé un cas potentiel. Un cas potentiel peut être mémorisé sous la forme d'un cas concret lors de l'apprentissage.

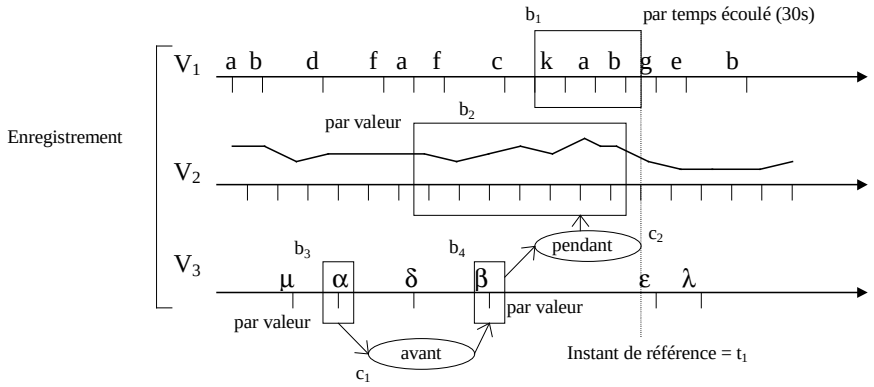


Figure II-16 : exemple de situation complexe

2.1.3 Description détaillée

Nous précisons tout d'abord la granularité des cas choisie dans notre modèle. Puis, nous présentons les deux niveaux de représentation qui séparent les données brutes d'observation stockées dans des *enregistrements*, des connaissances utiles avec deux types de cas : cas potentiels et cas concrets. Chaque cas est alors lié à une *situation comportementale* qui définit l'ensemble de ses indices effectifs par la sélection de certaines données dans un enregistrement.

2.1.3.1 *Granularité des cas*

Un cas est une expérience qu'il faut mémoriser et qui sera utile pour la résolution des problèmes futurs. Cependant, que représente un cas dans des problèmes complexes comme la nutrition des plantes, la navigation sur le Web, la prévision de l'évolution d'un feu de forêt ou encore la commande de robots ? Un cas représente-t-il une longue expérience ou une expérience particulière et assez précise dans le temps ? Ce problème est la généralisation du problème de la *granularité* des cas en prenant en compte la dimension temporelle : un cas est-il monolithique ou décomposable ? Un cas représente-t-il un unique enseignement ou un ensemble d'enseignements utiles ?

Par exemple, le système de raisonnement à partir de cas CELIA (Redmond, 1990) propose la décomposition d'un cas en morceaux (*snippet*). Chaque morceau est indexé séparément, et les morceaux d'un même cas sont liés entre eux. Le cas global est également indexé et contient des informations générales. Cette décomposition permet de retrouver des similarités entre des parties de cas différents et ainsi peut permettre la généralisation d'enseignements (Kolodner, 1993, pages 182–186). A l'inverse, REBECAS (Rougegrez, 1994a) propose une représentation avec un très gros grain : un cas contient tous les événements d'un feu depuis son début jusqu'à sa fin. REBECAS évite ainsi le stockage d'enseignements plus précis, mais l'inconvénient est immédiat car aucune leçon n'est tirée des différents raisonnements comme nous l'avons montré (cf. §1.1.2.2.2). Nous pensons que la prise en compte d'historiques ne fait que rendre plus nécessaire le besoin de représenter des expériences spécifiques pour améliorer notamment les possibilités d'apprentissage et l'efficacité de la recherche. C'est pourquoi nous appelons *cas* un enseignement à un instant précis, issu d'une longue expérience représentée dans notre modèle par un enregistrement. Cet enseignement est alors identifié par une situation comportementale.

Toutefois, la gestion de cas ayant une granularité temporelle fine implique la représentation de nombreux cas. Notre modèle d'indexation propose alors :

- une représentation à deux niveaux (enregistrement et cas) permettant de partager les données brutes d'observation nécessaires à l'expression des cas, pour minimiser les ressources nécessaires,
- deux types de cas (potentiels et concrets) permettant de ne pas représenter de manière explicite et exhaustive tous les cas.

2.1.3.2 *Représentation des données d'observation*

Les données d'observation sont représentées par des historiques de variables, appelés *chroniques*. Un *enregistrement* regroupe alors un ensemble de chroniques :

- pour une période de temps définie et
- pour un contexte donné.

Il est en effet courant de pouvoir identifier des périodes d'observation ou des sessions. L'observation du processus est alors segmentée dans le temps vis-à-vis d'un contexte donnant les caractéristiques constantes d'un enregistrement : la localisation de l'expérience, les modes opératoires ou encore les objectifs. Le contexte peut également comporter des informations de synthèse issues des chroniques (par exemple des statistiques sur les valeurs des variables).

Les données d'un enregistrement sont *brutes* par rapport à la représentation de haut niveau nécessaire pour les indices. Cependant, ces données qui sont issues de capteurs et de mesures font en général l'objet d'un prétraitement. De plus, certaines données peuvent être calculées ou déduites à partir des mesures de base. Nous n'abordons pas ici les problèmes de traitement du signal, bien que ces techniques puissent être utilisées en amont (pour traiter le bruit et les valeurs manquantes par exemple). Ainsi, nous supposons que le processus dynamique et son environnement sont observés suivant un ensemble de variables caractéristiques de leurs états qui sont potentiellement pertinentes vis-à-vis de la tâche effectuée par le système de raisonnement. Ces variables d'observation prennent des valeurs à différents instants utilisant une unité de temps propre.

Sur le plan du formalisme de représentation des historiques, nous optons pour la réutilisation de certains éléments proposés dans le système de gestion de bases de données temporelles TEMPOS (cf.§1.2.2.2). TEMPOS offre un cadre de base que nous enrichissons des opérateurs nécessaires à la gestion des indices de notre modèle d'indexation[20]. Cet outil se limite en effet au stockage et à des manipulations restreintes des historiques.

Dans TEMPOS, l'observation au cours du temps d'une variable est représentée par un historique en extension, appelé chronique, qui contient les données réellement saisies. Au-dessus de cette représentation, un historique en compréhension est défini : historique discret, historique en escalier ou historique interpolé. Ce type d'historiques interprète les valeurs manquantes sur le domaine temporel défini. Nous introduisons alors un troisième niveau : *l'historique en indices potentiels*. Ce type d'historiques est constitué des observations pertinentes d'une variable. Par exemple (cf. Figure II-17), un historique en indices potentiels peut ne retenir que les extremums de la fonction interpolée par l'historique en compréhension. De même, un historique en indices potentiels peut ne retenir que les événements dénotant un changement d'état (élimination des mesures successives identiques), nous obtenons alors un *historique concis* (Williams, 1986). Dans le cas le plus simple, les historiques en indices potentiels, en compréhension et en extension sont identiques.

[20] Nous désirions ainsi faciliter la perspective de l'intégration de cet outil comme un élément de base pour le stockage des historiques utilisés dans notre modèle d'indexation.

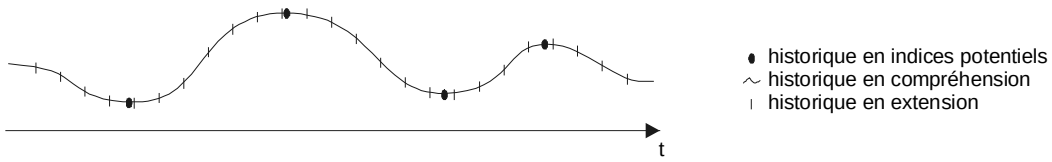


Figure II-17 : exemple d'historique en indices potentiels

Dans la suite, nous utilisons la notation suivante[21], basée sur celle de TEMPOS : <>, [], et → désignent respectivement les types des constructeurs de tuples, de séquences (suite ordonnée de valeurs) et de fonctions. $T_1(T_2)$ dénote une instance du type T_1 paramétrée par le type T_2. L'opérateur ← désigne l'énumération d'une séquence. Dans TEMPOS, le type d'historiques le plus général d'une variable est noté Historique(u,T) pour l'unité temporelle u et le type de valeur structurelle T. Nous étendons cette modélisation en ajoutant la fonction TS qui retourne l'historique en indices potentiels d'un historique, sous la forme d'une chronique d'instants :

TS : Historique(u, T) → I_chronique(u,T)

Pour des raisons de simplification, nous désignerons dans la suite de notre document sous le terme *chronique* le résultat de la fonction TS appliquée à un historique de TEMPOS et nous renommons TSchronique(u,T) son type (qui est I_chronique(u,T)). Nous avons alors les propriétés et les fonctions suivantes dans lesquelles Instant(u) dénote le type des instants de l'unité u :

TSchronique(u,T) : [<Instant(u),T>]
{ soit C= [C_1, ..., C_n] de type TSchronique(u,T), ∀ k ∈ [1..n–1], VT(C_k) < VT(C_{k+1}) }

DomT : TSchronique(u,T) → [Instant(u)]
{ domaine temporel : DomT (K) = [i | <i,v> ← K] }

VS: <Instant(u),T>→ T
{ valeur structurelle : VS(<i,v>) = v }

VT: <Instant(u),T> → Instant(u)
{ valeur temporelle : VT(<i,v>) = i }

Différents types de chroniques peuvent être définis suivant les propriétés de leur domaine temporel. Nous distinguons notamment les *chroniques échantillonnées* et les *chroniques à événements* (cf. Figure II-18). Si la durée séparant deux valeurs temporelles successives du domaine temporel est toujours constante, nous parlerons alors d'une chronique échantillonnée (périodique). Si la chronique n'est pas échantillonnée, nous parlerons de chronique à événements (apériodique), où un événement est une information datée et instantanée extraite du déroulement du processus. Ces événements peuvent avoir différentes sémantiques : alarmes, début ou fin d'une action, déroulement d'une action instantanée pour l'unité de temps considérée, changement d'état.

ChroniqueEch(u,T) : TSchronique(u,T)
{ soit C= [C_1, ..., C_n] de type TSchroniqueEch(u,T),
∃ d tel que ∀ k ∈ [1..n–1], VT(C_k) - VT(C_{k+1}) = d }

ChroniqueEvt(u,T) : TSchronique(u,T)
{ soit C= [C_1, ..., C_n] de type TSchroniqueEvt(u,T),
¬ (∃ d tel que ∀ k ∈ [1..n–1], VT(C_k) - VT(C_{k+1}) = d) }

[21] Cette notation repose sur des notions de programmation fonctionnelle (Bird & Wadler, 1988 ; Scholl *et al.*, 1993).

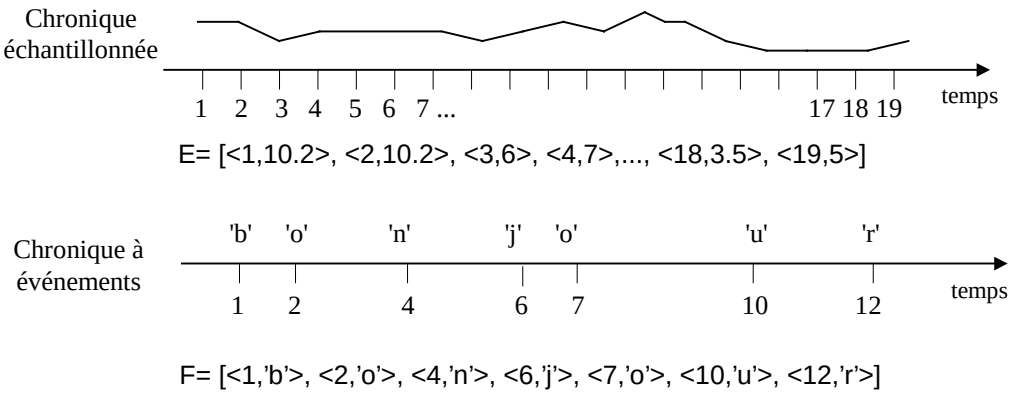


Figure II-18 : exemples de chroniques échantillonnées et à événements[22]

Plusieurs opérateurs peuvent être définis sur les chroniques en se basant, soit sur des opérations temporelles (projection, restriction, produit naturel), soit sur des opérations sur les séquences. Nous donnons uniquement la définition des huit opérateurs de base qui sont nécessaires pour manipuler les données d'observation et définir les indices d'un cas. Seuls les deux premiers opérateurs sont présents dans TEMPOS.

- Restriction temporelle sur les instants :

  Γ : TSchronique(u,T), [Instant(u)] $\rightarrow$ TSChronique(u,T)

  { Γ (K,R) = [<i,v> | <i,v> $\leftarrow$ K; i $\in$ R] }

- Découpage de la séquence suivant un prédicat (l'élément vérifiant le prédicat est exclu de la séquence retournée) :

  Début : TSchronique(u,T), (<Instant(u),T> $\rightarrow$ Booléen) $\rightarrow$ TSchronique(u,T)

  {Début ([],P) = []

  Début (e.S,P) = si P(e) alors [] sinon e.Debut(S,P)}

- Longueur d'une chronique :

  Len : TSchronique(u,T) $\rightarrow$ Entier

  { Len([]) =0 ; Len([C_1,...,C_n]) = n }

- Identification d'un élément par sa position :

  Index : TSchronique(u,T), Entier $\rightarrow$ <instant(u),T>

  { Index([C_1,...,C_k,...,C_n], k) = C_k si 1 $\leq$ k $\leq$n, nil sinon}

- Restriction par position :

  Γ_P : TSchronique(u,T), [Entier] $\rightarrow$ TSchronique(u,T)

  { Γ_P (K,R) = [e | j $\leftarrow$ R ; e = Index(K,j)] }

- Position du dernier élément dont la valeur temporelle est inférieure ou égale à un instant :

  Pos : TSchronique(u,T), Instant(u) $\rightarrow$ Entier

  { Pos(K,t) = Len(Début(K,f_t)) avec f_t(<i,v>) = vrai si i > t sinon faux }

- Instants de début et de fin d'une chronique :

  T_{min} : TSchronique(u,T) $\rightarrow$ Instant(u)

  { T_{min}([])=nil ; T_{min}(K)= VT(Index(K,1)) }

  T_{max} : TSchronique(u,T) $\rightarrow$ Instant(u)

  { T_{max}([]) = nil ; T_{max}(K) = VT(Index(K,Len(K))) }

[22] Dans ces exemples, l'ensemble des instants est considéré comme isomorphe aux entiers.

Par exemple, l'application de ces opérations à la chronique à événements F, décrite dans la Figure II-18, donne les résultats suivants :

Γ(F,[5...11]) = Γ(F,[6...10]) = [<6,'j'>, <7,'o'>, <10,'u'>]

Debut(F,g) = [<1,'b'>, <2,'o'>, <4,'n'>] avec g(<i,v>) = vrai si v='j' faux sinon

Len(F) = 7

Index(F,3) = <4,'n'>

Pos(F,9) = Pos(F,7) = 5

T_{min}(F) = 1 T_{min}(Γ(F,[6...10])) = 6

T_{max}(F) = 12

2.1.3.3 *Représentation des situations comportementales*

Un enregistrement peut regrouper un ensemble considérable de données brutes issues d'observations directes ou de traitements de ces observations pour une période de temps qui peut être assez étendue (plusieurs mois dans le cas de la nutrition des plantes). Nous supposons également que, pour une application donnée, les chroniques d'un enregistrement sont exprimées dans une même unité[23]. Nous proposons un modèle de représentation de *situations comportementales* permettant la sélection des indices effectifs d'un cas en référençant un enregistrement à un instant donné. Nous introduisons tout d'abord la structure générale d'une situation comportementale puis nous détaillons ses trois types d'éléments : d'une part les comportements élémentaires et leurs contraintes de localisation, et d'autre part les contraintes temporelles. Ces éléments peuvent être manipulés durant le raisonnement à travers des interfaces fonctionnelles que nous décrivons enfin.

2.1.3.3.1 Structure générale d'une situation comportementale

Le caractère prédictif d'un indice est très important pour la réutilisation d'un cas. Par exemple pour la nutrition des plantes, une décision est prise chaque jour pour déterminer la concentration de la solution de l'irrigation. Un indice prédictif est une caractéristique qui conditionne le choix de cette concentration. Un indice a donc un caractère explicatif qui peut être réutilisé dans les raisonnements. Si les indices ne caractérisent pas suffisamment un cas, celui-ci sera appliqué dans d'autres conditions et mènera à des échecs. Si les indices caractérisent trop précisément un cas, il ne pourra jamais être réutilisé. Notre modèle d'indexation n'aborde pas les méthodes permettant de déterminer les indices, il se limite à offrir un cadre de représentation et de gestion.

Pour la prise en compte d'historiques dans une situation, nous proposons la notion de *situation comportementale* permettant d'identifier un cas parmi les indices potentiels formés de toutes les données d'un enregistrement. Une situation comportementale est constituée :

- d'une *composante instantanée* qui comprend le contexte de l'enregistrement et toutes les données à caractère instantané qui peuvent être définies pour une application particulière,
- d'une *composante comportementale* qui définit un ensemble de *comportements élémentaires* pour chaque variable, ainsi que des *contraintes temporelles* entre ces comportements. De plus, chaque comportement élémentaire possède une *contrainte de localisation* qui explicite la raison pour laquelle il a été sélectionné.

La composante comportementale est basée sur la donnée d'un instant précis dans un enregistrement, appelé *instant de référence*. Cet instant détermine, pour la situation considérée, la limite entre d'une part le futur (instants strictement supérieurs) et d'autre part le passé et le présent. La composante comportementale décrit, de manière relative par rapport à cet instant, les évolutions

[23] L'étude de l'intégration de la multi-granularité du temps de TEMPOS reste en perspective de notre travail.

pertinentes des variables (comportements élémentaires et contraintes de localisation) et leurs relations (contraintes temporelles).

2.1.3.3.2 Comportements élémentaires et contraintes de localisation

Un comportement élémentaire définit une *sous-chronique* d'une chronique liée à une variable d'observation. Une sous-chronique K(n,m) est une chronique constituée d'une sous-séquence des éléments compris entre les positions n et m d'une chronique K. Une sous-chronique est définie par l'application de l'opérateur de restriction par position :

$$K(n,m) = \Gamma_P(K,[n..m])$$

Associée à un comportement élémentaire, une contrainte de localisation explicite la *raison* pour laquelle ce comportement est sélectionné. Grâce aux contraintes de localisation, il est possible d'identifier, dans les données d'observation d'un nouveau problème et par rapport à l'instant courant, les comportements pertinents *homologues* à une situation comportementale donnée. Cette étape d'identification menée au cours du raisonnement permettra alors de décider si le cas source lié à cette situation peut être réutilisé. Nous avons défini quatre types de fonctions de localisation (cf. Figure II-19) :

- *par position* : la sous-chronique est identifiée par le nombre de mesures depuis l'instant de référence,
- *par temps écoulé* : la sous-chronique est identifiée par le temps écoulé relativement à l'instant de référence,
- *par valeur* : la sous-chronique est identifiée par sa succession de valeurs,
- *par référence* : la sous-chronique est identifiée par référence à une sous-chronique existante.

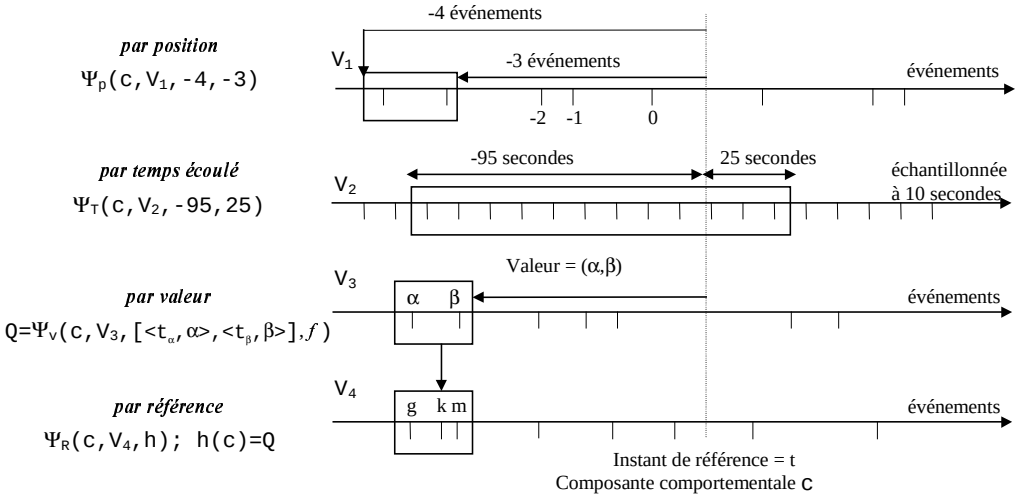


Figure II-19 : exemples de contraintes de localisation

Ces fonctions de localisation peuvent être formalisées en utilisant les opérateurs de base sur les chroniques. Nous définissons tout d'abord deux fonctions permettant de récupérer la chronique d'une composante comportementale pour une variable donnée (Chr) et l'instant de référence (Ref) :

Chr : ComposanteComportementale, Variable → TSchronique(u,T)

Ref : ComposanteComportementale → Instant(u)

Chaque fonction de localisation prend comme paramètre la composante comportementale considérée, notée c, et la variable sur laquelle porte la localisation, notée v. Les fonctions utilisent alors d'autres paramètres suivant l'opération à effectuer. La fonction Ψ_P permet d'identifier une sous-

chronique par position, étant donné les nombres de mesures dp_1 et dp_2 relativement à l'instant de référence.

Ψ_P : ComposanteComportementale, Variable, Entier, Entier $\rightarrow$ TSchronique(u,T)
{ Ψ_p (c,v, dp_1,dp_2) = Γ_P(Chr(c,v), [Pos(Chr(c,v),Ref(c))+dp_1... Pos(Chr(c,v), Ref(c))+dp_2] }

La fonction Ψ_T permet d'identifier une sous-chronique par temps écoulé relativement à l'instant de référence, étant donné les durées dt_1 et dt_2.

Ψ_T : ComposanteComportementale, Variable, Durée(u), Durée(u) $\rightarrow$ TSchronique(u,T)
{ Ψ_p (c,v,dt_1,dt_2) = Γ(Chr(c,v),[Ref(c)+dt_1...Ref(c)+dt_2])

La fonction Ψ_V permet d'identifier une sous-chronique par valeur, étant donné une chronique de référence Q et une fonction de reconnaissance *f*. Cette fonction de localisation permet de parcourir en arrière une chronique jusqu'à ce que la fonction *f* reconnaisse la sous-séquence.

Ψ_V : ComposanteComportementale, Variable, TSchronique(u,T),
(TSchronique(u,T), TSchronique(u,T) $\rightarrow$ Booléen) $\rightarrow$ TSchronique(u,T)
{ Ψ_V (c,v,[],*f*)= []; Ψ_V (c,v,Q,*f*) = g(Γ_P(Chr(c,v),1,Pos(Chr(c,v),ref(c)),Q,*f*)
g(W,Q,*f*) =[] avec Len(W)<Len(Q)
g(S.W.e,Q,*f*) = W.e si *f*(W.e,Q) sinon g(S.W,Q,f) avec Len(W.e)=Len(Q) }

La fonction Ψ_R permet d'identifier une sous-chronique d'une chronique K, étant donné une fonction h donnant la chronique déjà identifiée dans la composante comportementale.

Ψ_R : ComposanteComportementale, Variable,
(ComposanteComportementale $\rightarrow$ TSchronique(u,T))$\rightarrow$ TSchronique(u,T)
{ Ψ_R (c,v, h)= Γ(Chr(c,v),[T_{min}(h(c))...T_{max}(h(c))])}

Chaque comportement élémentaire, noté b_i, est alors défini par une chronique d'origine (ts), par sa localisation dans la situation d'origine (loc), et par sa contrainte de localisation (lc) à utiliser pour la mise en correspondance. Une contrainte de localisation est représentée par une fonction de localisation dont la composante comportementale est inconnue (notée '.'). Ce paramètre sera déterminé lors de l'utilisation de la contrainte. La localisation courante (loc) représente l'utilisation de la contrainte de localisation dans la composante comportementale considérée.

En reprenant l'exemple de situation comportementale de la Figure II-16, nous pouvons exprimer les comportements élémentaires et leurs contraintes de localisation sous la forme suivante, étant donné les chroniques K_1, K_2 et K_3 correspondant respectivement aux variables V_1, V_2 et V_3 :

ComportementsElémentaires = {
b_1={ts=K_1, loc=K_1(8,10), lc=Ψ_T(.,V_1,-30,0)},
b_2={ts=K_2, loc=K_2(7,13), lc=Ψ_v(.,V_2,loc,*f*)},
b_3={ts=K_3, loc=K_3(2,2), lc=Ψ_v(.,V_3,[<t_α,α>],*f*)},
b_4={ts=K_3, loc=K_3(4,4), lc=Ψ_v(.,V_3,[<t_β,β>],*f*)}}

Nous avons ainsi proposé quatre types de contraintes de localisation mais cette liste n'est pas exhaustive, et le formalisme utilisé permet d'exprimer d'autres types de contraintes adaptés à la gestion d'indices dans une application spécifique.

2.1.3.3.3 Contraintes temporelles

La définition des comportements pertinents élémentaires permet d'obtenir un ensemble d'éléments de même nature, bien que les chroniques de l'enregistrement puissent être différentes (échantillonnées ou à événements). Ces comportements peuvent alors être reliés par des contraintes temporelles entre

intervalles (relations d'Allen (Allen, 1984) cf. Figure II-20). Chaque comportement élémentaire $K_{n,m}$ définit en effet un intervalle temporel $[T_{min}(K(n,m))...T_{max}(K(n,m))]$.

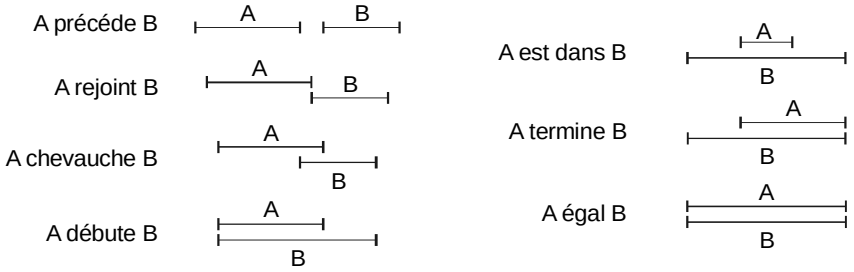


Figure II-20 : relations entre intervalles

Si les comportements élémentaires utilisés sont réduits à un élément, ils représentent des instants. Dans ce cas, les contraintes entre instants doivent être utilisées. Les intervalles représentant les comportements élémentaires sont alors décomposés en un instant de début et un instant de fin. Les contraintes temporelles peuvent porter sur des comportements élémentaires issus de variables différentes. En reprenant l'exemple de la Figure II-16, nous pouvons exprimer les contraintes temporelles pour former la description complète de la composante comportementale :

ComposanteComportementale= {
ComportementsElémentaires = {
b_1={ts=K_1, loc=K_1(8,10), Ic=Ψ_T(.,V_1,-30,0)},
b_2={ts=K_2, loc=K_2(7,13), Ic=Ψ_v(.,V_2,loc,*f*)},
b_3={ts=K_3, loc=K_3(2,2), Ic=Ψ_v(.,V_3,[<t_α,α>],*f*)},
b_4={ts=K_3, loc=K_3(4,4), Ic=Ψ_v(.,V_3,[<t_β,β>],*f*)}},
ContraintesTemporelles= { c_1=avant(b_3,b_4), c_2=pendant(b_4,b_2)}}

2.1.3.3.4 Interfaces fonctionnelles

Nous distinguons quatre interfaces fonctionnelles permettant de définir les manipulations possibles des indices comportementaux d'un cas durant le raisonnement. Pour la phase de recherche, *l'interface de consultation* est utilisée pour accéder aux valeurs des indices des cas sources pour pouvoir les comparer suivant des mécanismes spécifiques aux méthodes de recherche utilisées. Lors de la réutilisation, *l'interface d'analyse* est utilisée pour obtenir les différences entre deux composantes comportementales. *L'interface de sélection* permet principalement à la méthode d'apprentissage de pouvoir sélectionner, parmi les indices potentiels, les indices effectifs du cas cible à mémoriser. Si l'organisation de la mémoire doit être mise à jour, *l'interface de modification* permet d'ajuster l'indexation de chaque cas. Les interfaces permettent de manipuler les *éléments* de la composante comportementale : comportements élémentaires, contraintes de localisation et contraintes temporelles.

- L'interface de consultation permet l'accès aux comportements élémentaires et aux contraintes temporelles d'une composante comportementale. De plus, cette interface permet l'accès à la définition d'un comportement élémentaire : chronique référencée, sous-chronique sélectionnée et contrainte de localisation
- L'interface d'analyse permet de lister les comportements élémentaires et les contraintes temporelles communs entre deux composantes comportementales. Cette interface permet également de connaître les comportements élémentaires et les contraintes temporelles présents dans une première composante comportementale et absents dans une seconde.
- L'interface de sélection permet l'ajout de comportements élémentaires et de contraintes temporelles dans une composante comportementale.
- L'interface de modification permet la suppression des comportements élémentaires et des contraintes temporelles ainsi que la modification des contraintes de localisation : par

exemple extension ou diminution de la sous-séquence sélectionnée. La suppression d'un comportement élémentaire entraîne la suppression des contraintes temporelles associées.

2.1.3.4 *Représentation des cas*

Nous présentons la représentation d'un cas cible qui intègre le problème courant, puis nous détaillons les deux types de cas sources définis.

2.1.3.4.1 Cas cible

Nous rappelons qu'un cas cible est le cas en cours d'étude sur lequel le raisonnement porte. Il est constitué du problème courant à résoudre et d'une solution inconnue que le raisonnement doit déterminer. Un cas cible référence un instant précis, appelé *instant courant*, à l'intérieur de l'enregistrement courant. La composante comportementale du cas cible est vide : les indices comportementaux effectifs du cas cible sont inconnus. Toutes les données disponibles (observations et estimations) sont considérées comme des indices potentiels. Le raisonnement a donc pour but de chercher des cas sources ayant des indices effectifs pouvant être mis en correspondance parmi les indices potentiels du cas cible.

2.1.3.4.2 Deux types de cas sources

Nous proposons l'utilisation de deux types de cas : *cas potentiels* et *cas concrets.* Un cas potentiel est une notion que nous introduisons pour prendre en compte les situations non explicites, c'est-à-dire que la situation d'un cas potentiel devra être explicitée lors d'un raisonnement. Un cas potentiel n'a pas d'existence propre, mais il est défini à travers un *patron de cas potentiels*. Un patron est considéré comme un ensemble de cas potentiels en compréhension. Les patrons sont utilisés lors de la phase de recherche pour expliciter des connaissances jusqu'alors cachées dans l'ensemble des données des enregistrements. Un patron peut être en effet instancié pour un instant de référence et, par abus, nous appelons également « cas potentiel » le résultat de cette instanciation qui reste éphémère. Si le résultat de cette instanciation est ajouté à la mémoire du système lors de l'apprentissage, nous parlerons alors de cas concrets (cf. §2.2.3). Un patron P définit un prédicat (applicable_P) indiquant s'il est applicable à un instant donné dans un enregistrement et une fonction d'instanciation (nouveau_P) :

applicable_P : Enregistrement, Intant(u) $\rightarrow$ Booléen

nouveau_P : Enregistrement, Instant(u) $\rightarrow$ CasPotentiel

Les cas concrets sont identifiés par des situations explicites et dynamiques. Ainsi, la représentation d'un cas concret peut être modifiée (notamment avec l'interface de modification des indices) au cours des échecs et des succès des raisonnements. Ces modifications permettront de retrouver les cas concrets pour des raisonnements dans lesquels ils seront plus utiles. Un cas concret porte des informations quant à son origine : définition manuelle ou patron utilisé pour son instanciation.

2.2 Guide d'utilisation de notre modèle de représentation

Pour l'utilisation de notre modèle de représentation des indices et des cas, nous avons identifié quatre étapes importantes :

1. Identification des variables d'observation.
2. Détermination de la sémantique d'un enregistrement et de son contexte.
3. Définition de la représentation des cas et des situations comportementales.
4. Conception des phases de raisonnement.

Pour la dernière étape qui est fortement liée au processus de raisonnement à partir de cas, nous présentons des principes de gestion des indices permettant :

- de définir un *cadre unifié* pour la description des phases de raisonnement des applications réalisées afin de simplifier leur compréhension (recherche des cas pour la nutrition des plantes cf. §2.3, et assistance à la navigation cf. chapitre V, §2),
- de fournir une *base méthodologique* facilitant l'utilisation du modèle dans la conception des phases de raisonnement pour d'autres applications.

Nous proposons ainsi des principes de gestion portant sur les phases de recherche, de réutilisation et d'apprentissage. Ces principes présentent des typologies de traitements et de données propres à notre modèle de représentation. Ils pourront être mis en œuvre suivant les connaissances et les caractéristiques d'un domaine d'application considéré.

2.2.1 Principes de recherche

Les principes de recherche reposent premièrement sur cinq types de filtrage. Ces étapes doivent alors être assemblées suivant les contraintes spécifiques d'une application. Notre modèle d'indexation laisse ainsi ouvert le contrôle de la recherche, tout en définissant des étapes de base. De plus, nous proposons une *stratégie séquentielle de recherche* qui donne un exemple typique d'assemblage de ces étapes.

2.2.1.1 *Types de filtrage pour la recherche*

Nous rappelons qu'un filtrage est une étape de la phase de recherche : étant donné un ensemble de cas sources et un cas cible, l'exécution d'un filtrage retourne un sous-ensemble des cas sources jugés les plus utiles pour le raisonnement au stade considéré de la recherche. Au niveau connaissance, nous avons défini cinq types principaux de filtrage :

- *Filtrage sur la composante instantanée*. Tout comme dans PADIM (Fuchs *et al.,* 1995) et RADIX (Corvaisier *et al.,* 1998), un filtrage suivant la composante instantanée peut être effectué. La composante instantanée comprend le contexte de l'enregistrement référencé, et il est souvent utile d'éliminer de la recherche les cas ayant un contexte complètement différent, car les connaissances d'adaptation peuvent être inexistantes ou difficiles à mettre en œuvre.
- *Filtrage sur les comportements élémentaires*. L'ensemble des comportements élémentaires est pris en compte pour la mise en correspondance des cas sources avec le cas cible. Avant chaque comparaison, une étape d'identification des comportements élémentaires doit être effectuée. En effet, à partir des contraintes de localisation de chaque comportement, les comportements homologues dans le cas cible sont identifiés (par position, par temps écoulé, par valeur ou par référence). Ensuite la mise en correspondance entre les comportements élémentaires eux-mêmes est effectuée puis agrégée. La sélection et le tri des cas sources peuvent prendre en compte le nombre de comportements élémentaires constituant chaque situation comme un critère supplémentaire. En effet, il peut être nécessaire de privilégier la sélection d'un cas formé d'une situation complexe (donc plus spécifique) à un cas formé de peu de comportements élémentaires.
- *Filtrage sur une restriction*. Une restriction rassemble les notions de point de vue de REBECAS (Rougegrez, 1994a) (restriction sur les variables) et de tendance de ICONS (Schmidt et al., 1996) (restriction sur des horizons temporels). Une restriction est un ensemble de contraintes de localisation permettant d'identifier des comportements élémentaires jugés pertinents dans tous les cas. Une restriction permet donc d'éviter la définition de ces comportements dans chaque situation comportementale. De plus, la restriction considérée est modifiable directement sans nécessiter le changement exhaustif des comportements élémentaires des cas sources. Ce filtrage permet de ne retenir que les

cas ayant un minimum de similarité sur certaines variables et sur un horizon temporel restreint.

- *Filtrage basé sur les contraintes temporelles*. Suite à l'identification des comportements élémentaires dans le cas cible pour chaque cas source pris en compte, les contraintes temporelles sont instanciées dans le contexte du problème courant. Ce filtrage doit retourner les cas sources qui respectent le mieux les contraintes temporelles (suivant le nombre de contraintes satisfaites par exemple).
- *Filtrage des patrons*. Uniquement pour la recherche des cas potentiels, cette étape permet d'obtenir les patrons qui peuvent être appliqués à l'instant courant dans l'enregistrement du cas cible (utilisation du prédicat applicable_P).

Lors de la réalisation d'un système de RàPC utilisant notre modèle d'indexation, ces étapes pourront être assemblées pour former une séquence, ou regroupées dans une même opération. Enfin, ces étapes sont alors réalisées grâce à des index, en tirant profit des propriétés des indices manipulés (domaines de valeurs, structuration). Différents index sont en effet utilisables, comme par exemple (cf. chapitre III) : index par plus proches voisins (linéaire ou optimisé), index par hiérarchie discriminante, index par table de hachage. La plate-forme à objet CBR*Tools que nous proposons (cf. chapitres III et IV) permet de réaliser et d'intégrer ces index dans un système de RàPC.

2.2.1.2 *Exemple d'une stratégie séquentielle de recherche*

La stratégie séquentielle de recherche que nous proposons est basée sur les deux heuristiques suivantes :

- il est préférable de réutiliser des cas concrets plutôt que des cas potentiels, car les cas concrets peuvent avoir été révisés, reliés à des cas d'échec et/ou de succès, ou porter des informations spécifiques quant à leur réutilisation ;
- il n'est pas nécessaire de chercher les cas sources les plus utiles, mais il suffit d'identifier les cas satisfaisant un critère minimum d'utilité (seuil de similarité par exemple).

Suivant ces deux heuristiques, notre stratégie se décompose en deux phases séquentielles : recherche des cas concrets et recherche des cas potentiels. Le résultat global de la recherche correspond à celui de la première phase dont le résultat est non vide ou plus généralement dépassant un seuil de qualité (similarité minimale par exemple). Cette recherche séquentielle amène une meilleure efficacité puisque la recherche des cas potentiels est effectuée *au besoin*, lorsque les cas concrets sont de qualité suffisante (au sens du filtrage) pour être retenus. La recherche des cas potentiels est en effet la plus coûteuse des phases de recherche puisqu'elle nécessite l'instanciation des patrons sur les enregistrements supposés très nombreux et volumineux.

Nous proposons de combiner les types de filtrage identifiés de la façon suivante pour la recherche des cas concrets :

1. *Filtrage sur la composante instantanée*. La composante instantanée contient généralement des données représentant un niveau de description moins détaillé que la composante comportementale. Il s'agit donc d'effectuer en premier lieu un filtrage sur cette composante. De plus, sa structure peut être plus simple, menant à un filtrage peu coûteux et d'une incidence importante sur le nombre de cas à considérer par la suite.
2. *Filtrage sur une restriction.* Une restriction définit un ensemble d'indices pertinents dans tous les cas. Nous utilisons cette notion pour ne retenir que les cas ayant une correspondance minimum avec le cas cible. Cette étape nécessite une mise en correspondance moins complexe que le filtrage sur l'ensemble des comportements élémentaires (étape suivante).
3. *Filtrage sur les comportements élémentaires*. L'identification des comportements élémentaires et leur comparaison constituent une étape coûteuse dans le cas de situations

complexes. Toutefois, le nombre de cas sources à considérer est contrôlé par les deux étapes précédentes.

4. *Filtrage sur les contraintes temporelles*. Les contraintes temporelles permettent de comparer un cas cible et un cas source au niveau de détail le plus fin. Cette étape est donc la dernière et peut être appelée *sélection*.

Pour la recherche des cas potentiels, des étapes similaires sont appliquées avec une première étape supplémentaire permettant de filtrer les patrons à considérer :

1. *Filtrage des patrons*. Les étapes suivantes devront au besoin instancier les patrons sur les enregistrements à des instants différents afin d'effectuer la comparaison avec le cas cible. L'identification des patrons de cas potentiels applicables est donc un prérequis qui est réalisé par cette étape.
2. *Filtrage sur la composante instantanée*. A ce stade de la recherche, il est judicieux de ne prendre en compte que les indices du contexte des enregistrements. Ainsi, la comparaison des cas sources potentiels avec le cas cible ne nécessite pas l'instanciation des patrons. Cette étape réalise alors le filtrage des enregistrements.
3. *Filtrage sur une restriction*. Comme pour les cas concrets, cette étape est importante d'une part pour assurer un minimum de correspondance avec le cas cible, et d'autre part pour réduire le nombre de cas à examiner avec les étapes suivantes. De plus, cette étape ne nécessite pas l'instanciation des patrons, qui est ainsi retardée. A la fin de cette étape, les cas potentiels identifiés sont complètement instanciés (fonction nouveau_P) et peuvent être manipulés par la suite comme des cas concrets.
4. *Filtrage sur les comportements élémentaires pertinents*. Idem aux cas concrets.
5. *Filtrage sur les contraintes temporelles*. Idem aux cas concrets.

2.2.2 Principes de réutilisation

A la suite de la phase de recherche, le système récupère :

- *le cas cible*,
- le ou *les cas sources retenus* dont les indices sont suffisamment vérifiés dans le cas cible pour estimer que chaque cas source donne une *interprétation valide* du cas cible, étant donné les connaissances du système au moment du raisonnement,
- *les traces de recherche* donnant des indications utiles pour le raisonnement et particulièrement pour la phase de réutilisation. Ces informations peuvent principalement décrire : les degrés d'utilité de chaque cas source (similarité), les interprétations du cas cible par ces cas sources (identification des comportements élémentaires et des contraintes temporelles), les résultats intermédiaires de recherche.

Avec ces informations, la phase de réutilisation doit déterminer la solution suggérée pour le cas cible courant. Si les traces de recherche intègrent les interprétations du cas cible, cette phase peut alors se baser sur la mise en correspondance pour effectuer des opérations correctrices sur les solutions proposées par les cas sources. Ces opérations sont fortement dépendantes des connaissances du domaine. Les différences entre la situation comportementale d'un cas source et la situation interprétée du cas cible sont identifiées au travers de l'interface fonctionnelle d'analyse des indices. La gestion de conflits éventuels entre les solutions proposées par les cas sources est une tâche typique de cette phase.

2.2.3 Principes d'apprentissage

L'apprentissage est la phase du raisonnement qui doit mettre à jour l'indexation de la mémoire pour améliorer le système. Quatre types d'opérations sont possibles :

- *Ajout du cas cible*. De manière classique en RàPC, le cas cible peut être mémorisé et indexé dans la mémoire comme un cas concret en définissant ses indices effectifs grâce à l'interface de sélection. Nous prenons ainsi l'hypothèse qu'à la fin du raisonnement il est possible de déterminer ses indices suivant une sélection par défaut ou suivant la solution obtenue après révision (retour d'information après application de la solution dans un environnement réel par exemple). Cet ajout permet de ne pas exécuter un raisonnement identique (qui a pu être coûteux avec une résolution de conflit par exemple) dans un futur problème similaire. L'ajout d'un cas cible nécessite l'ajout de l'enregistrement sous-jacent.
- *Ajout des cas potentiels*. Les cas potentiels identifiés lors de la phase de recherche doivent être étudiés par la phase d'apprentissage pour déterminer s'ils peuvent être utiles dans les raisonnements futurs. Dans ce cas, les cas potentiels sont mémorisés sous la forme de cas concrets. Leurs indices sont alors pris en compte dans les structures d'indexation pour des accès plus rapides et pourront être mis à jour dans les raisonnements suivants.
- *Ajout de l'enregistrement courant*. Lorsqu'un enregistrement est fermé (changement de contexte), il peut être ajouté à la mémoire si cette opération n'a pas déjà été effectuée lors de l'ajout d'un cas cible. Grâce à l'instanciation des patrons de cas potentiels effectuée au besoin avec la stratégie de recherche séquentielle, ce type d'apprentissage est dit *paresseux*. Toutefois, l'enregistrement, pour lequel les patrons ne sont pas applicables ou donneront des situations comportementales vides, ne doit pas être retenu.
- *Autre réorganisation de la mémoire et mise jour des cas*. De multiples réorganisations peuvent être déclenchées à la suite d'un raisonnement : création, modification ou élimination des cas concrets. L'objectif de cette réorganisation est d'améliorer la description des cas et de maintenir l'efficacité de la recherche.

2.3 Un exemple : recherche des cas pour la nutrition des plantes

La problématique générale de cette application a été présentée dans le premier chapitre (cf. chapitre I, §1.1). Notre objectif est d'étudier la conception d'un atelier permettant d'assister un utilisateur dans la découverte et l'analyse des connaissances liées au processus de régulation de la nutrition des plantes. Il s'agit donc d'un système permettant de manière interactive :

- de proposer des concentrations de la solution d'irrigation pour réguler le processus tout en expliquant le raisonnement effectué en se basant sur les cas utilisés,
- de modifier les situations des cas dans la phase d'apprentissage au regard des différences entre l'évolution attendue et effective du processus.

Dans cette section, nous nous plaçons dans le cadre formé par le guide d'utilisation pour présenter la conception d'un système de recherche des cas similaires pour la nutrition des plantes. A travers cette application, nous illustrons ainsi notre modèle d'indexation et nous montrons comment les cas du processus de nutrition peuvent être représentés. Cette spécification a donné lieu à la réalisation d'un prototype pour valider les représentations choisies et les mécanismes de recherche de cas similaires. L'utilisation de notre modèle d'indexation pour la réalisation d'un système complet de raisonnement est donnée au chapitre V avec notre prototype d'assistance à la navigation sur le Web.

2.3.1 Identification des variables d'observation

Dans le problème de la régulation de la nutrition des plantes, le processus à observer comprend un ensemble de plantes, l'environnement et le système de régulation. Le processus est observé par différentes variables donnant pour chaque jour (cf. Figure II-21) : la température, le rayonnement solaire, l'humidité, la concentration de la solution d'irrigation ainsi que l'écart de la concentration du drainage avec la consigne à maintenir. Les évolutions de ces variables sont représentées par des

chroniques échantillonnées numériques où l'unité de temps est le jour. Ces variables ont été choisies par les experts du domaine (après des analyses statistiques) pour leur pertinence potentielle vis-à-vis de l'objectif de la régulation qui est de maintenir stable la concentration du drainage. Le système de raisonnement à partir de cas doit alors permettre de déterminer la concentration de la solution de l'irrigation à appliquer pour le jour suivant, étant donné toutes les observations disponibles à l'instant courant.

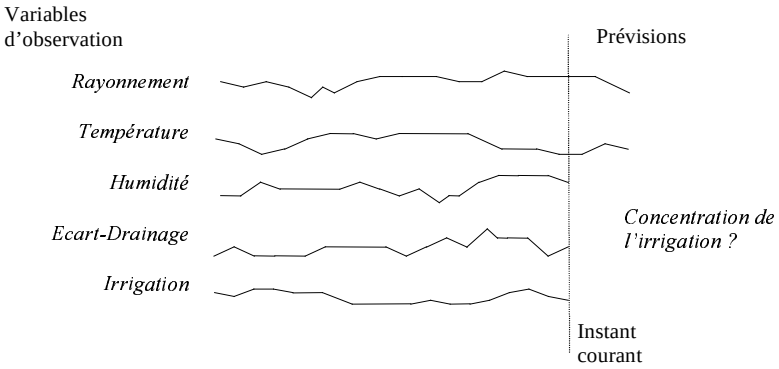


Figure II-21 : données d'observation pour la nutrition des plantes

2.3.2 Détermination de la sémantique d'un enregistrement et de son contexte

Plusieurs expérimentations de régulation sont menées et peuvent même se dérouler simultanément. Chaque expérimentation a pour objectif d'évaluer le choix pris pour la consigne de concentration du drainage à maintenir. Cette consigne est alors constante pour toute la durée de l'expérimentation. C'est pourquoi, nous regroupons les données d'observations dans des enregistrements qui représentent les données collectées pour une unique expérimentation. La consigne à maintenir définit alors le contexte de l'enregistrement (cf. Figure II-22).

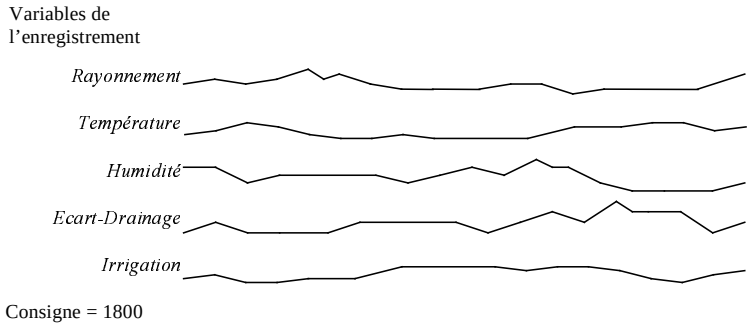


Figure II-22 : exemple d'enregistrement pour la nutrition des plantes

2.3.3 Définition de la représentation des cas

Les cas (sources et cible) sont structurés en deux composantes :

- une composante *problème* qui comprend les données d'observation et la situation comportementale (indices effectifs du cas),
- une composante *solution* qui donne la concentration de la solution d'irrigation correspondante et l'évaluation de la régulation. Cette évaluation est calculée à partir de l'évolution de la concentration de la solution de drainage dans les cinq jours suivant l'instant courant.

2.3.3.1 *Cas cible*

Un cas cible est constitué d'une situation comportementale vide (les indices effectifs sont indéterminés) et d'une solution que le raisonnement doit calculer :

```
CasCible {
    IdentificateurCas= "exemple-cas-cible",
    Origine = cas-cible,
    Problème= {
        Enregistrement= R, Instant= t,
        SituationComportementale= {
            ComposanteInstantanée= {ContexteEnregistrement= {Consigne= 1800}},
            ComposanteComportementale= {
                ComportementsElémentaires = {}
                ContraintesTemporelles= {}}}},
    Solution= {     Commande= ?,
                    Evaluation= ?}}
```

2.3.3.2 *Cas potentiels*

Nous définissons également deux patrons de cas potentiels : P_1 qui est applicable lorsque la moyenne du rayonnement solaire des 5 derniers jours dépasse un seuil fixé, et P_2 qui est applicable dans le cas contraire. Nous utilisons les expérimentations existantes qui montrent que pour un fort rayonnement solaire l'inertie est de 2 jours, et respectivement de 5 jours pour un rayonnement faible. C'est pourquoi le patron P_1 définit les règles de sélection des évolutions passées sur les 2 derniers jours et respectivement P_2 sur les 5 derniers jours. La fonction d'instanciation du patron P_2 s'exprime alors de la façon suivante :

```
nouveauP2 (R : Enregistrement,t : Instant(u)) {
    retourne CasPotentiel {
        IdentificateurCas= générerIdentificateur(),
        Origine= P2,
        Problème= {
            Enregistrement= R, Instant= t,
            SituationComportementale= {
                ComposanteInstantanée= {ContextEnregistrement= {Consigne= Consigne(R)}},
                c=ComposanteComportementale= {
                    ComportementsElémentaires = {
                        b1={ts=K1, loc=ΨP(c,Rayonnement,-4,5), lc=ΨP(.,Rayonnement,-4,5)},
                        b2={ts=K2, loc=ΨP(c,Température,-4,5), lc=ΨP(.,Température,-4,5)},
                        b3={ts=K3, loc=ΨP(c,Humidité,-4,0), lc=ΨP(.,Humidité,-4,0)},
                        b4={ts=K4, loc=ΨP(c,Ecart-Drainage,-4,0), lc=ΨP(.,Ecart-Drainage,-4,0)},
                        b5={ts=K5, loc=ΨP(c,Irrigation,-4,0), lc=ΨP(.,Irrigation,-4,0)}},
                    ContraintesTemporelles= {}}},
        Solution= {     Commande= ΨP(c,Irrigation,1,1),
                        Evaluation=  ΨP(c,Ect-Drainage,1,5)}}
    }
```

Dans cet exemple, l'expression de la fonction d'instanciation utilise les fonctions de localisation définies. Toutefois, les règles d'instanciation d'un patron peuvent être plus complexes, que ce soit sur la partie problème ou sur la partie solution (cf. règles de sélection pour Broadway, chapitre V, §2.1.3.2). Un algorithme d'instanciation est alors nécessaire, mais il est pratique, dans un cas simple comme celui-ci, de pouvoir représenter un patron dans le même formalisme qu'un cas potentiel instancié.

2.3.3.3 *Cas concrets*

Un cas concret peut être issu de l'instanciation d'un cas potentiel et, grâce à sa représentation explicite, il peut être manipulé et révisé suivant les raisonnements successifs d'échec ou de succès. Il peut également être issu d'une création manuelle par les experts du domaine qui annotent ainsi une

expérience concrète de régulation. Nous donnons comme exemple un cas créé avec le patron précédent et ayant été modifié par l'utilisateur avec l'ajout d'un comportement pertinent (b_6) et d'une contrainte temporelle (c_1) (cf. Figure II-23).

CasConcret {
  IdentificateurCas= "exemple-cas-concret",
  Origine= P_2,
  Problème= {
    Enregistrement= R, Instant= t,
    SituationComportementale= {
      ComposanteInstantanée= {ContextEnregistrement= {Consigne= 1800}},
      ComposanteComportementale= {
        ComportementsElémentaires = {
          b_1={ts=K_1, loc=K_1(46,55), Ic=Ψ_P(.,Rayonnement,-4,5)},
          b_2={ts=K_2, loc=K_2(46,55), Ic=Ψ_P(.,Température,-4,5)},
          b_3={ts=K_3, loc=K_3(46,50), Ic=Ψ_P(.,Humidité ,-4,0)},
          b_4={ts=K_4, loc=K_4(46,50), Ic=Ψ_P(.,Ecart-Drainage,-4,0)},
          b_5={ts=K_5, loc=K_5(46,50), Ic=Ψ_P(.,Irrigation,-4,0)},
          b_6={ts=K_1, loc=K_1(16,26), Ic=Ψ_V(.,Rayonnement,loc,f)}}},
        ContraintesTemporelles= { c_1= avant(b_6,b_1)}}}},
  Solution= { Commande = 1200,
    Evaluation = K_4(51,56)}}

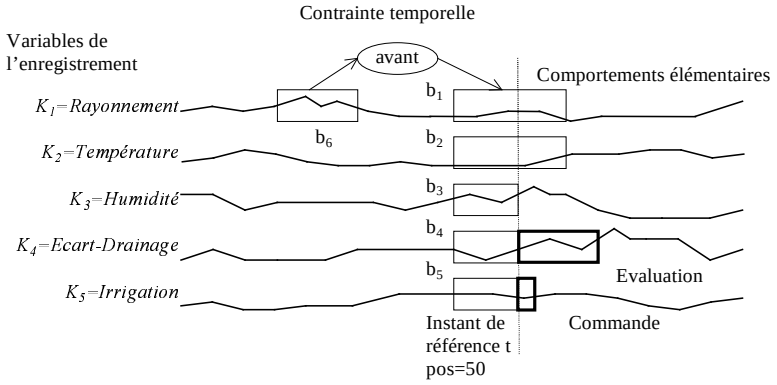


Figure II-23 : exemple de cas concret pour la nutrition des plantes

2.3.4 Conception des phases de raisonnement

Dans cet exemple, nous nous limitons à l'étude de la phase de recherche. Pour cette phase, nous utilisons notre stratégie séquentielle de recherche. La recherche des cas concrets utilise les quatre étapes que nous avons définies :

1. *Filtrage sur la composante instantanée* : la recherche est restreinte aux cas provenant d'enregistrements ayant la même consigne pour la concentration du drainage. En effet, nous n'avons pas encore de connaissances pour réutiliser les cas issus de contraintes de régulation différentes. Un index par table de hachage, permettant de ne retenir que les cas correspondant à la même consigne, est utilisé.
2. *Filtrage sur une restriction* : la restriction correspond à la sélection des évolutions passées sur les 2 derniers jours, sachant que ces données sont pertinentes dans tous les cas. Ce filtrage peut être effectué avec un index par plus proches voisins optimisé (*Kd-tree*, Wess *et al.,* 1994) utilisant une similarité numérique basée sur les écarts de valeurs. Cet index est applicable car les valeurs sont toutes numériques et leur nombre est figé.
3. *Filtrage sur les comportements élémentaires* : chaque comportement est mis en correspondance avec son comportement homologue grâce à une mesure de similarité basée sur la moyenne des écarts de chaque mesure de la chronique. Ensuite, une similarité

globale est calculée en faisant la moyenne des similarités de chaque comportement. Cette similarité est combinée avec le ratio entre le nombre de comportements d'une situation et le nombre maximum de comportements. Un index de type plus proches voisins avec un seuil de similarité minimal permet d'identifier un sous-ensemble de cas sources dont la similarité est la plus grande tout en présentant le plus de comportements élémentaires.

4. *Sélection sur les contraintes temporelles* : un index de type plus proches voisins permet d'identifier les meilleurs cas en utilisant une mesure de similarité calculant le ratio de contraintes satisfaites.

Finalement, si aucun cas concret n'est retrouvé, la recherche des cas potentiels est effectuée. Le filtrage des patrons de situations comportementales est basé sur le rayonnement solaire courant suivant les prédicats d'application des patrons. Par conception des prédicats d'applicabilité, un seul patron est sélectionné pour le raisonnement courant. Puis, les mêmes étapes que précédemment sont utilisées.

3 Conclusion

Dans ce chapitre, nous avons étudié et proposé des solutions pour la gestion des indices basés sur des historiques. Nos apports se situent à deux niveaux :

- Proposition d'une approche globale et générique de l'indexation à travers le *modèle d'indexation par situations comportementales* comprenant le modèle de représentation proprement dit des indices et des cas ainsi qu'un guide d'utilisation donnant un cadre de description et de conception des phases de raisonnement. Notre approche d'indexation pour le RàPC est la seule abordant de manière générale la représentation et la gestion d'historiques.
- Proposition d'un modèle de représentation des indices dépassant les limites des approches existantes de RàPC : représentation d'historiques hybrides, meilleure représentation pour la découverte de connaissances.

Premièrement, notre démarche est en effet originale par rapport aux travaux existants en RàPC qui restent liés à des applications particulières. Le modèle de représentation des indices et des cas est en effet générique : données d'observation regroupées dans des enregistrements et situation comportementale formée de comportements élémentaires et de contraintes temporelles. De plus, le guide d'utilisation associé à notre modèle de représentation est chargé de réunir les connaissances et les méthodes pour l'intégration effective des représentations dans les phases de raisonnement. Nous proposons un ensemble de types généraux de traitements faisant la synthèse des problématiques d'indexation étudiées (nutrition des plantes et assistance à la navigation sur le Web). Ce guide est destiné à être suivi et étendu par l'analyse de nouvelles applications ou par l'étude générale de traitements particuliers (généralisation de chroniques par exemple). La problématique du modèle d'indexation est également différente de celle abordée par les approches de reconnaissance d'historiques et les systèmes de gestion de bases de données temporelles. Il s'agit en effet de proposer des représentations et des mécanismes de gestion des indices dans un cycle de raisonnement visant la réutilisation d'expériences passées : possibilité d'analyser la mise en correspondance pour effectuer la réutilisation, possibilité de mettre à jour des cas existants.

Deuxièmement, sur le plan du modèle de représentation des indices, nous permettons tout d'abord, contrairement aux approches de RàPC étudiées, l'expression d'indices sur des chroniques échantillonnées et à événements. En effet, la sélection de comportements élémentaires crée une structure de plus haut niveau qui masque ainsi l'hétérogénéité des chroniques. Puis, nous proposons

une représentation permettant une meilleure gestion des trois étapes du processus de découverte des connaissances (cf. § 1.1.2.2) :

1. *Extraction structurée de cas potentiels à partir des données brutes*. L'extraction de cas potentiels est effectuée grâce à l'instanciation des *patrons de cas potentiels*. Les cas potentiels permettent de représenter des situations non explicites comme dans la plupart des approches existantes. Grâce à l'utilisation originale de patrons, nous permettons en plus l'expression d'une *structuration* (comportements élémentaires et contraintes temporelles) par l'utilisation d'hypothèses existantes.
2. *Sauvegarde et mise à jour des cas potentiels sous forme de cas concrets.* Une fois identifiés, ces cas potentiels peuvent être enregistrés sous la forme de cas concrets. La description de ces cas peut alors être modifiées suivant les raisonnements. Nous pallions ainsi au manque des approches existantes (sauf SINS) qui extraient des cas ou des parties de cas, sans pouvoir retenir les conséquences de leur utilisation. La structure des situations permet alors de sélectionner et de modifier les indices effectifs parmi les indices potentiels : définition de comportements élémentaires et expression de contraintes temporelles. Ainsi, depuis les situations simples issues de l'instanciation des patrons, les raisonnements peuvent construire des connaissances plus précises et faire émerger des comportements plus complexes, proches de scénarios temporels. De plus, il est possible d'associer à chaque cas concrets des informations complémentaires (par exemple l'évaluation de la qualité du cas suivant un critère lié à l'application) qui seront utilisées lors des raisonnements.
3. *Interprétation facilitée des cas concrets*. Contrairement à SINS qui effectue un apprentissage en créant des cas virtuels, nous permettons une démarche visant à déterminer, sur une expérience concrète, les indices effectifs tout en éliminant les descriptions qui sont jugées non pertinentes (indices potentiels). Ainsi, les cas concrets référencent toujours les données brutes réelles pour faciliter leurs interprétations par des experts du domaine.

Notre modèle de représentation permet une expression riche des situations mais peut nécessiter une identification coûteuse : mise en correspondance de multiples comportements élémentaires et de contraintes temporelles, instanciation des patrons de cas potentiels. Nous donnons toutefois l'exemple d'une stratégie de recherche séquentielle combinant différents types de filtrage pour réduire progressivement les cas à examiner et retarder les étapes coûteuses. De plus, la transformation des cas potentiels en cas concrets, combinée avec la stratégie séquentielle de recherche proposée, amène une meilleure efficacité, puisque l'extraction de comportements n'est pas effectuée à chaque raisonnement.

Notre modèle d'indexation a été utilisé pour la spécification et la réalisation d'un prototype de recherche des cas pour la nutrition des plantes. Dans ce domaine, nous pensons que l'utilisateur est capable d'effectuer lui-même, lors de la phase d'apprentissage, la modification de l'indexation d'un cas source suite à son utilisation. Par exemple, si le comportement prévu par un cas source (concret ou potentiel) ne s'est pas reproduit, il peut analyser les indices et ajouter un comportement élémentaire qui pourrait expliquer cette différence. Chaque cas représente ainsi une hypothèse de l'explication d'un comportement, et le système entier se présente comme un gestionnaire d'hypothèses. La mise en place d'un tel système dans un environnement réel permettrait d'effectuer l'acquisition des connaissances pour concevoir à terme un système automatique.

Notre modèle a été également utilisé dans la réalisation complète et opérationnelle du système Broadway pour l'assistance à la navigation sur le Web (cf. chapitre V). Ce système permet d'évaluer les principales caractéristiques du modèle d'indexation : représentation des cas concrets et potentiels,

et guide d'utilisation pour les phases du raisonnement. La création de cas abstraits par généralisation de chroniques, ainsi que la modification automatique des indices des cas, constituent des perspectives intéressantes qui permettraient d'étendre nos travaux.

Enfin, pour l'utilisation concrète de notre modèle, nous avons conçu une plate-forme à objets facilitant son utilisation par la spécialisation et la composition d'un ensemble d'objets. Cette plate-forme facilite également l'intégration du modèle et son utilisation dans un système complet de raisonnement à partir de cas (cf. chapitres III et IV).

Chapitre III : plate-forme à objets pour la réalisation de systèmes de RàPC

Résumé

Pour faciliter l'utilisation de notre modèle d'indexation par situations comportementales, il est nécessaire de concevoir un outil ouvert permettant principalement la spécialisation du modèle puis son intégration cohérente dans un système complet de RàPC. Les outils existants, bien que réutilisables dans les domaines ciblés, ne reposent pas sur une modélisation ouverte du RàPC. Plus précisément, ils ne facilitent ni la modification des composants proposés, ni l'ajout de nouveaux composants. Pour dépasser ces limites, nous proposons un *nouveau type d'outils* pour le RàPC à travers la conception d'une plate-forme à objets. Notre approche repose sur la définition d'une architecture abstraite modélisant les concepts du RàPC et de l'indexation par situations comportementales. Cette architecture intègre des points d'ouverture qui peuvent être configurés par spécialisation ou par instanciation pour répondre aux besoins spécifiques d'une application. Nous proposons ainsi un ensemble de points d'ouverture structuré en deux niveaux de spécificité (RàPC et indexation par situations comportementales) et en trois axes de variabilité (gestion du raisonnement, représentation des cas et organisation de la mémoire). La représentation graphique de cette structuration sous la forme de cas d'utilisation définit alors un guide d'utilisation des points d'ouverture au sein de notre plate-forme CBR*Tools.

Dans le chapitre précédent, nous avons présenté le modèle d'indexation des cas par situations comportementales. Ce modèle repose sur une représentation spécifique des indices pour prendre en compte des historiques et comprend également un guide d'utilisation. Ce modèle présente un cadre général pour la gestion des situations comportementales et nous voulons faciliter son utilisation par la conception et la réalisation d'un outil. Dans ce but, cet outil doit permettre la spécialisation et l'intégration du modèle ainsi que la capitalisation de composants. En définitive, cet outil doit être basé sur une architecture *ouverte* (cf. chapitre I, §3.3.2) et doit aborder la problématique générale du RàPC pour fournir un cadre cohérent d'intégration du modèle d'indexation au sein d'un système de RàPC.

Nous commençons, dans la première partie, par mener l'état de l'art des outils logiciels existants en RàPC et nous montrons leurs limites en terme d'ouverture. Nous proposons alors, dans la deuxième partie, notre approche basée sur la conception d'une *plate-forme à objets*. Nous introduisons tout d'abord le concept de plate-forme à objets puis nous argumentons les qualités de la plate-forme réalisée en présentant ses *points d'ouverture*. Le chapitre suivant décrit la réalisation de ces points d'ouverture au sein de modèles à objets ouverts. Nous donnons également, dans le chapitre V, la configuration de ces points d'ouverture pour la réalisation de Broadway, et l'annexe C décrit leur utilisation dans un exemple classique de RàPC.

1 Outils logiciels existants pour le RàPC

De nombreux générateurs d'applications ou « *shells* » ont été développés pour le raisonnement à partir de cas, dont les premiers datent de 1991. La revue des caractéristiques de ces outils[24] et leurs comparaisons sont régulièrement effectuées depuis 1992 (Harmon, 1992 ; Althoff *et al.,* 1995 ; Watson, 1997). La synthèse la plus remarquable est sans aucun doute présentée dans (Althoff *et al.,* 1995), où de nombreux tests comparatifs sont menés tant sur le plan technique qu'ergonomique. Toutefois, ces analyses ne prennent généralement en compte que les outils commerciaux.

Dans cette section, nous faisons un état de l'art des principaux outils logiciels existants. Nous écartons de cette présentation les approches récentes qui tentent de définir des méthodes pour la gestion d'un projet de RàPC (Bergmann *et al.,* 1997 ; Bergmann *et al.*, 1998), mais qui n'abordent pas les problèmes de conception et de réalisation en termes d'outils logiciels. Nous nous limitons tout d'abord à la description de ces outils (méthodes proposées, interfaces de programmation). Cette présentation a également pour but d'introduire les techniques de base du raisonnement à partir de cas qui seront utilisées dans la suite du document. Puis, nous menons une analyse critique pour identifier leurs limites en terme d'ouverture.

1.1 Présentation des outils

Dans un premier temps, nous ne décrivons que les outils présentant des approches intéressantes en terme d'ouverture qui seront analysées par la suite (cf. §1.2) : REMIND, REPRO, KATE et le projet FABEL. REMIND est un outil intégrant de nombreux index qui peuvent être combinés. REPRO est construit au-dessus de REMIND et montre comment enrichir un outil existant pour le rendre encore plus flexible. KATE offre différentes techniques d'indexation, dont l'induction dynamique, permettant

[24] Le terme *outil* est employé au regard de la tâche de réalisation d'une application RàPC et non de la tâche que le système réalisé permettra d'accomplir.

un meilleur traitement des valeurs manquantes. Enfin, l'approche prise dans le projet FABEL montre comment construire une application RàPC par assemblage de composants hétérogènes et distribués. Pour finir, nous décrivons brièvement les autres outils disponibles.

1.1.1 REMIND

REMIND (REMIND, 1992 ; Barletta, 1994) est le résultat d'un vaste effort de recherche mené dans différentes universités américaines durant la fin des années 1980 et soutenu par l'agence américaine de recherche pour la défense (DARPA). Cet outil fut alors commercialisé par la société Cognitive Systems à partir de 1992. REMIND est basé sur une représentation non structurée des cas[25], c'est-à-dire qu'un cas est défini par une liste de couples (attribut, valeur). Le stockage des cas s'effectue dans une base de données locale et propriétaire. Plusieurs éditeurs graphiques sont alors proposés pour la définition des attributs des cas et la description de hiérarchies de symboles.

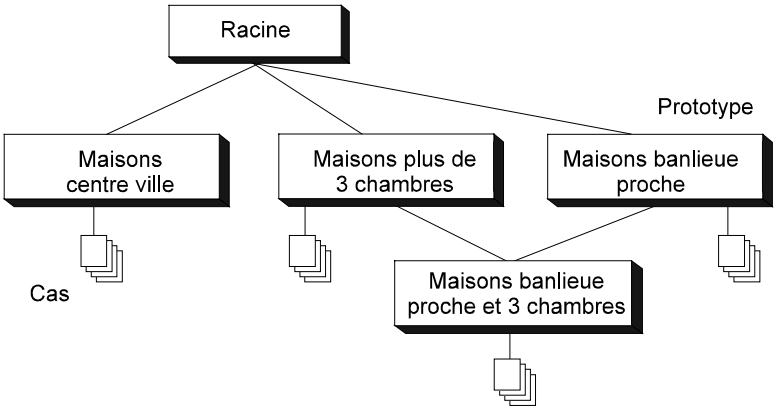


Figure III-1 : exemple d'une hiérarchie de prototypes

REMIND offre quatre index de base : hiérarchie de prototypes, arbre d'induction, plus proches voisins et requête. Un prototype définit un ensemble de contraintes sur les indices d'un cas, et tous les cas vérifiant ces conditions lui sont associés. Les prototypes sont organisés dans une hiérarchie d'héritage multiple qui permet de spécifier des prototypes généraux et d'autres plus spécifiques. Les prototypes sont construits manuellement et permettent de segmenter l'espace des cas en utilisant des connaissances du domaine d'application. Lors de la recherche, le graphe de prototypes est parcouru depuis la racine jusqu'aux prototypes les plus spécifiques dont les contraintes sont vérifiées. Toute contrainte faisant intervenir une valeur inconnue est supposée vraie. Les cas associés aux prototypes les plus spécifiques sont alors retournés. Par exemple, dans une application chargée d'évaluer la valeur d'une maison en réutilisant des évaluations passées, une hiérarchie identifiant les grandes classes de maisons peut être définie (cf. Figure III-1) pour regrouper les cas suivant des contraintes sur leurs indices.

L'indexation par arbre d'induction de REMIND permet de construire automatiquement un arbre discriminant binaire à partir de l'analyse de la base de cas (cf. Figure III-2). Un arbre discriminant, aussi appelé arbre de décision ou encore arbre de régression, définit un arbre dont chaque nœud est associé à un indice. Chaque fils d'un nœud correspond à une valeur ou un domaine de valeurs possibles de cet indice. Les feuilles de l'arbre contiennent tous les cas qui répondent aux contraintes de valeurs sur les indices suivant le chemin qui mène de la racine de l'arbre à la feuille. Le but de cet arbre est d'arriver le plus vite possible à classer un cas cible dans une des feuilles. L'arbre doit donc

[25] Toutefois, REMIND permet d'imbriquer des cas, mais les attributs utilisés dans le raisonnement et notamment les indices, doivent être définis au plus haut niveau (en définissant par exemple des attributs de type formule qui vont recopier ou utiliser des attributs de cas imbriqués).

avoir certaines qualités. Tout d'abord, il doit être le plus pertinent possible, c'est-à-dire que les indices doivent être analysés dans un bon ordre. De plus, la hauteur de l'arbre doit être la plus petite possible. Dans REMIND, cet arbre est construit de manière automatique en utilisant un *algorithme d'induction*, dont des exemples connus sont les algorithmes ID3 (*Iterative Dichotomizer Three*) (Quinlan, 1979) et CART (Breiman *et al.,* 1984). Les algorithmes de ce type étudient à chaque nœud quel est l'indice à examiner pour optimiser le gain d'information par rapport à l'attribut du cas qui représente la solution, en prenant en compte les valeurs des indices de la base de cas existante ainsi que les indices déjà examinés dans les niveaux supérieurs de l'arbre. REMIND inclut, dans ce calcul, des poids d'importance sur les indices et des modèles qualitatifs du domaine reliant les différents indices. Lors de la phase de recherche, l'arbre d'induction est parcouru depuis la racine. Les indices du cas cible sont examinés suivant l'ordre établi dans l'arbre et, à chaque nœud, la branche dont la contrainte de valeurs n'est pas satisfaite est éliminée du parcours. Si la valeur d'un indice est inconnue, l'algorithme est obligé d'examiner les différents fils possibles. Dans REMIND, ce parcours peut être interactif ou se faire à partir d'une description préalable du cas cible. Dans ce dernier cas, si une valeur est indéterminée, REMIND peut la demander à l'opérateur. Enfin, les cas associés aux feuilles ainsi identifiées sont retournés.

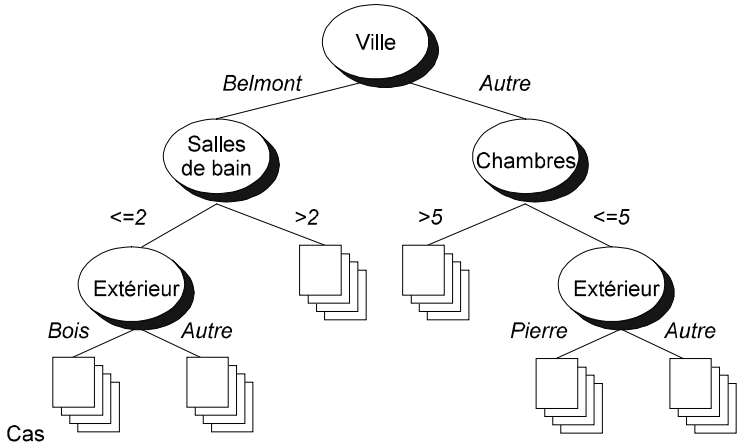


Figure III-2 : arbre d'induction

Dans une recherche par plus proches voisins, une structure linéaire d'indexation est parcourue par un algorithme qui calcule la similarité entre le cas cible et chaque cas source, de manière à ne retenir que les *k* meilleurs. Cette méthode repose sur la définition de fonctions élémentaires de similarité entre chaque indice. Dans REMIND, ces fonctions sont liées au type de l'indice et, à titre d'exemple, nous donnons la fonction utilisée pour les types date, entier et réel :

$$sim(x, y) = 100 - \left| f(x) - f(y) \right|$$

Dans cette fonction de similarité, *x* et *y* sont deux valeurs d'un même indice, et *f* une fonction qui ramène le domaine de valeurs de l'indice entre 0 et 100, en utilisant la moyenne et l'écart type des valeurs de l'indice pour toute la base de cas. La similarité globale entre un cas cible *c* et un cas source *s* est alors calculée en faisant une moyenne pondérée pour agréger les similarités élémentaires :

$$S(c, s) = \frac{\sum_{i=1}^{m} w_i . sim(c_i, s_i)}{\sum_{i=1}^{n} w_i}$$

REMIND permet de paramétrer le nombre de cas recherchés ainsi que le vecteur de poids à utiliser (plusieurs vecteurs sont nommés et stockés).

Enfin, REMIND offre un quatrième index grâce aux requêtes. Une requête est semblable à un prototype et définit un ensemble de contraintes sur les valeurs des indices des cas. Les requêtes sont nommées et enregistrées, ainsi l'utilisateur peut les exécuter à tout moment. Le fonctionnement de cet index est similaire à une requête dans une base de données.

Les possibilités d'indexation de REMIND ne s'arrêtent pas à ces quatre index de base et certaines de leurs combinaisons sont également proposées :

- prototypes et arbres d'induction,
- arbre d'induction et plus proches voisins,
- prototypes, arbres d'induction et plus proches voisins,
- requêtes et plus proches voisins.

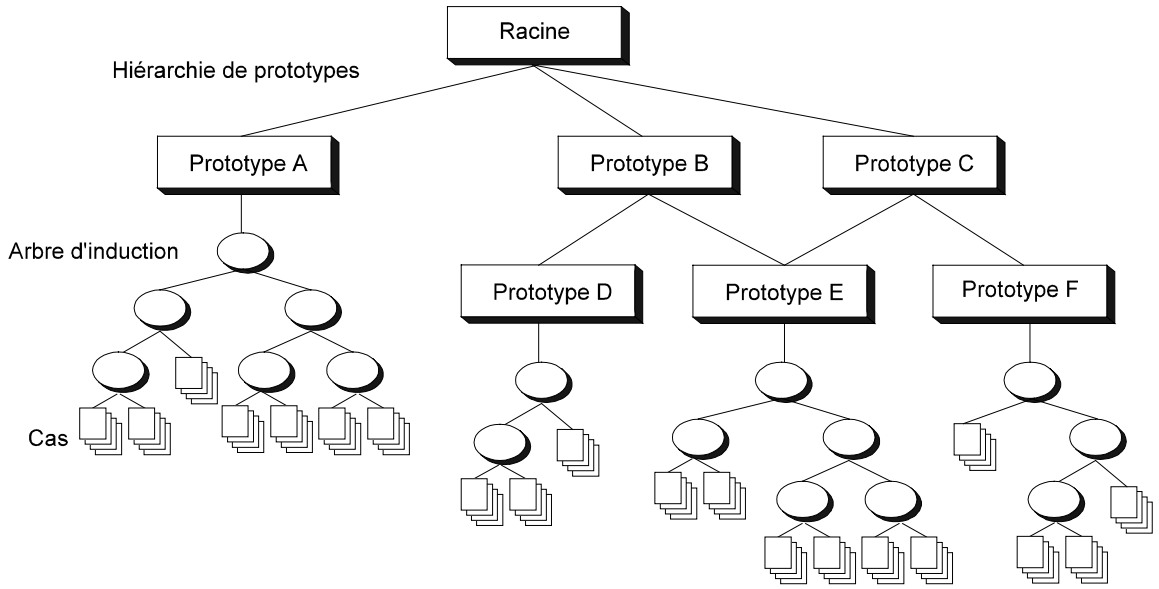


Figure III-3 : index hybride dans REMIND

Par exemple, la combinaison des prototypes, des arbres d'induction et des plus proches voisins repose alors sur une structure complexe (cf. Figure III-3) dans laquelle un arbre d'induction peut être associé à chaque prototype. Cette organisation présente deux avantages importants : elle permet d'améliorer l'organisation d'un arbre d'induction grâce à une connaissance *a priori* du domaine, qui est modélisée par cette hiérarchie statique de prototypes ; de plus, la construction de chaque arbre d'induction est plus efficace car elle ne prend en compte que les cas associés au prototype et non l'ensemble des cas tout entier. Enfin, l'union des cas identifiés par les arbres d'induction est analysée avec une recherche par plus proches voisins pour sélectionner les meilleurs cas.

Ainsi, REMIND offre un ensemble d'index qui peuvent être combinés. Pour la phase de réutilisation, REMIND propose l'utilisation de formules qui vont analyser les différences entre les indices du cas cible et d'un cas source identifié par la phase de recherche. Par exemple, pour l'évaluation de la valeur d'une maison, la différence du nombre de chambres intervient dans le calcul de la valeur de la maison cible c_{valeur} en adaptant la valeur de la maison source s_{valeur} :

$$c_{valeur} = s_{valeur} + 10000(c_{chambres} - s_{chambres})$$

De plus, il est possible de définir des *vues* pour constituer différentes stratégies de recherche et d'adaptation tout en utilisant les données d'une même base de cas. Enfin, REMIND propose une librairie de plus de 50 fonctions en langage C permettant de l'utiliser dans un environnement spécifique :

- manipulation des bases de cas (création, ouverture, fermeture),
- manipulation des cas (création et édition des attributs des cas, modification et stockage des cas),
- lancement de recherches simples ou composées en combinant les méthodes,
- édition des requêtes et des vecteurs de poids pour la recherche par plus proches voisins,
- lancement de la phase d'adaptation.

1.1.2 REPRO

REPRO (Mark *et al.,* 1996) est un outil de raisonnement à partir de cas conçu chez Lockheed. Il fait suite au développement d'un système, nommé CABER, pour le diagnostic et la résolution de pannes. Dans cette application, les index ont dû être souvent modifiés durant son développement et sa mise en place. Ainsi, REPRO est un outil partant de deux hypothèses issues de l'expérience pratique du système CABER (Mark *et al.,* 1996, page 284) :

- « les caractéristiques d'un domaine d'application peuvent nécessiter l'utilisation de différents algorithmes de recherche et d'apprentissage durant le cycle de vie d'un système de RàPC »,
- « la sélection des algorithmes appropriés doit être basée sur une méthodologie qui prend en compte les caractéristiques des connaissances du domaine et les caractéristiques des algorithmes ».

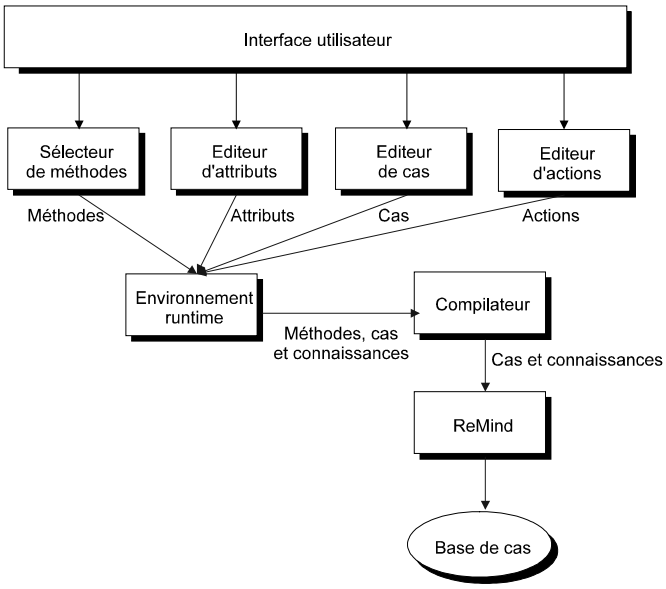


Figure III-4 : architecture de REPRO (Mark *et al.,* 1996)

REPRO est un outil permettant de sélectionner, de combiner et de faire évoluer l'indexation utilisée dans un système de raisonnement à partir de cas appliqué au diagnostic et à la résolution de pannes. REPRO met l'accent sur les phases de recherche et d'apprentissage (REPRO est l'acronyme de « *recherche et propose* »). Cet outil fournit alors un ensemble d'éditeurs (cf. Figure III-4) permettant de définir une base de cas (éditeurs de cas et d'attributs) et de sélectionner des types d'index (sélecteur de méthodes). L'éditeur d'actions permet également de définir les connaissances du domaine nécessaires pour les méthodes sélectionnées. Dans cet environnement, l'utilisateur peut construire et évaluer différentes stratégies et lorsque les résultats sont satisfaisants, REPRO génère une application indépendante. REPRO est une couche au-dessus de REMIND, et il utilise les fonctionnalités existantes de gestion des cas ainsi que les méthodes et structures de recherche.

1.1.3 KATE

KATE[26] représente un ensemble d'outils (dont notamment KATE-DataMining et KATE-CBR) qui peuvent être intégrés dans un processus de raisonnement à partir de cas. Ces outils sont commercialisés par la société Acknosoft et ont profité d'une collaboration européenne au sein du projet INRECA (projet ESPRIT). Ces outils proposent trois index de base : plus proches voisins, arbre d'induction, et induction dynamique. Alors que les deux premiers index sont classiques (cf. REMIND), l'induction dynamique permet au moment de la résolution du problème de construire l'arbre de décision. Cette technique offre l'avantage d'un meilleur traitement des valeurs inconnues et permet d'adapter les indices à examiner en fonction de ceux déjà exprimés. Les index par plus proches voisins et par induction peuvent également être combinés (Auriol *et al.*, 1994).

```
defcase 5
    objects
        Hyalonema Hyalonema5
            identification : identification1, contexte : contexte1, description : description1;
        identification identification1
            classe : Oonema, reference : "henshawi Lendenfeld 1915";
        contexte contexte1
            localite : "Eastern trop Pacific", latitude : "5 deg 17'S", longitude : "19 deg 5W",
            profondeur : 4086 ;
        description description1
            corps : corps1;
        corps corps1
            micro-elements : micro-elements1, forme : en-cloche, taille : 124 , couleur : blanchatre;
        micro-elements micro-elements1
            tignules : nothing, microxyhexactines : microxyhexactines1,
            pinules-dermaux : pinules-dermaux1,
            amphidisques : amphidisques1 amphidisques2 amphidisques3;
```

Figure III-5 : extrait de la représentation d'un cas en CASUEL

Dans l'ensemble de ces méthodes et contrairement à REMIND par exemple, une représentation structurée à objets peut être utilisée. L'importation et l'exportation des cas et des connaissances du domaine sont effectuées en respectant un format nommé CASUEL (Manago *et al.*, 1994). Ce format permet de gérer des cas structurés avec un formalisme à objets. La Figure III-5 donne une partie de la description d'un cas dans ce format pour une application de classification d'éponges marines : chaque cas possède un identificateur et définit un graphe d'objets. Chaque composant est l'instance d'une classe et possède un nom qui est alors suivi d'une liste de valeurs d'attributs. Les classes de ces objets ainsi que les types des attributs sont définis dans d'autres parties conformément au format.

1.1.4 FABEL

De 1992 à 1996, le projet FABEL (Gebhardt *et al.,* 1997) a réuni différents centres de recherche allemands pour l'étude de l'assistance à la conception en architecture, avec des approches basées sur des modèles et sur le raisonnement à partir de cas. Le résultat de cette étude s'est concrétisé par le développement d'un prototype qui porte le même nom, comportant un ensemble de composants distribués. Les composants développés ont ainsi été évalués dans le domaine de l'architecture par leur intégration dans un logiciel d'assistance à la conception, appelé M5PSEDIT. Pourtant, la plupart de ces composants ont une portée plus générale et peuvent être utilisés dans d'autres contextes. Plus précisément, un ensemble de composants pour le RàPC sont proposés, dont notamment : RABIT, ASM et ASPECT pour la phase de recherche, TOPO pour les phases de recherche et de réutilisation, AAAO pour la phase de réutilisation, CHECKUP pour la phase de révision, et enfin CBASE et CSET pour la

[26] http://www.acknosoft.com/

gestion des bases de cas et des ensembles de cas manipulés. Nous décrivons brièvement ces différents composants pour montrer leur diversité puis nous reviendrons sur l'architecture logicielle utilisée.

RABIT est un composant pour le calcul de similarités entre indices structurés. RABIT utilise un formalisme à objets pour représenter les indices sous forme d'attributs simples et composites. Pour le calcul d'une similarité, un contexte de mise en correspondance (*matching context*) doit être construit pour permettre d'associer à chaque attribut une fonction de similarité, ainsi que des poids d'importance. Pour les attributs composites, une fonction d'agrégation est spécifiée. Une bibliothèque de fonctions de similarité pour les attributs simples et composites est également définie. La Figure III-6 donne l'exemple d'un contexte de mise en correspondance pour un indice composite représentant une adresse (comprenant un pays, un code postal, un lieu et une rue).

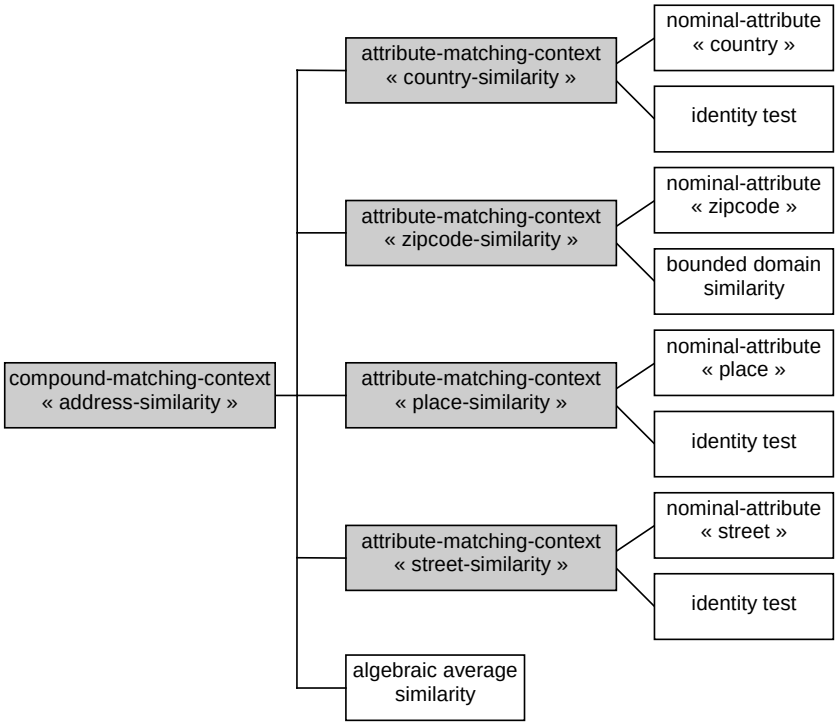


Figure III-6 : contexte du calcul d'une similarité dans Rabit (Gebhardt *et al.*, 1997, page 54)

ASM est un composant de recherche de cas similaires basé sur l'utilisation de réseaux de neurones. Suivant cette méthode, les indices d'un cas sont décrits par un ensemble de descripteurs ayant tous la même importance. ASM permet deux types de recherches : une recherche dite *exacte*, dans laquelle les cas sources comportant au moins les mêmes descripteurs que le cas cible sont retournés, et une recherche dite *partielle*, qui identifie les cas sources partageant le plus grand nombre de descripteurs avec le cas cible. ASPECT permet de combiner différentes mesures de similarités (qui sont dans ce cas des distances) : chaque cas possède plusieurs ensembles d'indices appelés *aspects* sur lesquels des distances de comparaison sont définies. La comparaison entre un cas cible et un cas source se fait suivant chaque aspect, puis les résultats sont agrégés avec un vecteur de poids d'importance. Un algorithme de recherche permet alors de retrouver les cas similaires en utilisant une structure précalculée liant les différents cas.

Le composant TOPO est utilisé à la fois pour la recherche et l'adaptation. TOPO permet de comparer des graphes d'objets et de relations pour la phase de recherche ; il permet ensuite d'adapter le graphe retenu. De manière plus spécifique au domaine de l'architecture, AAAO permet de réaliser l'adaptation de la disposition de colonnes dans une architecture initiale proposée par un cas. Dans cette approche, chaque colonne est représentée par un agent qui négocie et se déplace pour arriver à

une disposition globale acceptable. CHECKUP permet de vérifier les agencements des éléments d'un bâtiment en émettant des avertissements lorsque des contraintes sont violées. Enfin, le composant CBASE permet de créer, d'ajouter, d'effacer ou d'éditer les cas ; et le composant CSET permet de sauvegarder des ensembles de cas issus de recherches avec la possibilité d'effectuer des intersections, des unions et des différences entre les ensembles.

Ces différents composants sont réunis dans une application suivant une architecture répartie. Chaque composant peut s'exécuter sur une machine distante et possède une interface graphique simple qui est intégrée à l'outil d'assistance à la conception M5PSEDIT, et une interface graphique d'experts pour manipuler en détail les différents paramètres du composant. Les composants ne communiquent pas directement entre eux, mais au travers du logiciel M5PSEDIT. Cette architecture a été choisie pour trois raisons importantes. Tout d'abord, les composants ont pu être implantés dans différents langages de programmation : Lisp, C, C++, Oz. Puis, les composants sont exécutés suivant les besoins de l'utilisateur et sur des machines potentiellement différentes pour une meilleure répartition et utilisation des ressources. Enfin, de nouveaux composants peuvent être rajoutés en suivant les procédures de développement et de connexion.

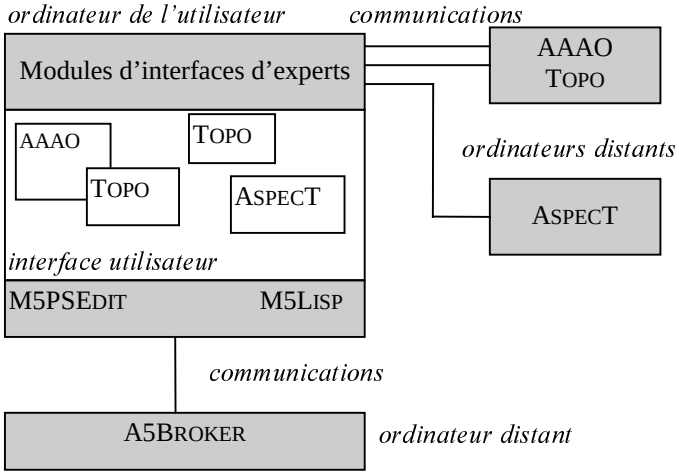


Figure III-7 : exemple de configuration distribuée du prototype FABEL (d'après Gebhardt *et al.*, 1997, page 205)

La Figure III-7 donne l'exemple d'une configuration dans laquelle les composants AAAO, TOPO et ASPECT sont utilisés à partir d'ordinateurs distants. Cet exemple montre également l'utilisation du module M5LISP qui permet de gérer les communications avec A5BROKER, la base de données à objets utilisée dans FABEL.

Le projet FABEL montre ainsi une réussite dans l'application du raisonnement à partir de cas à des problèmes complexes, avec une approche très modulaire et distribuée basée sur l'utilisation de méthodes variées durant le raisonnement.

1.1.5 Autres outils

CBR 3[27] est un ensemble d'outils pour construire des applications d'assistance technique et de diagnostic. L'objectif de ces applications est de déterminer les actions à effectuer pour résoudre un problème technique, par exemple : réparation d'une imprimante, utilisation d'un logiciel ou d'un environnement comme Windows 95. Dans cet outil, un cas est constitué d'un titre, de la description

[27] CBR 3 est commercialisé par la société Inference Corp., http://www.inference.com.

d'un problème, d'un ensemble de questions avec leurs réponses pour caractériser le contexte, ainsi que d'un ensemble d'actions à mener pour résoudre le problème. Au début d'un raisonnement, le cas cible est initialisé avec la description textuelle du problème courant. Ensuite, l'ensemble des cas sources ayant une description similaire est identifié. Puis, suivant les réponses aux questions que peut fournir l'utilisateur, les cas sources sont sélectionnés et ordonnés. La Figure III-8 donne un exemple d'utilisation du système pour la résolution d'un problème dans lequel le curseur de la souris d'un ordinateur ne bouge plus à l'écran. Cette figure montre le dernier panneau après une succession de cinq questions, et le cas source le plus similaire est alors indiqué sur la gauche (mention « *try it* ») : il s'agit de bien visser et serrer le connecteur de la souris au boîtier de l'unité centrale.

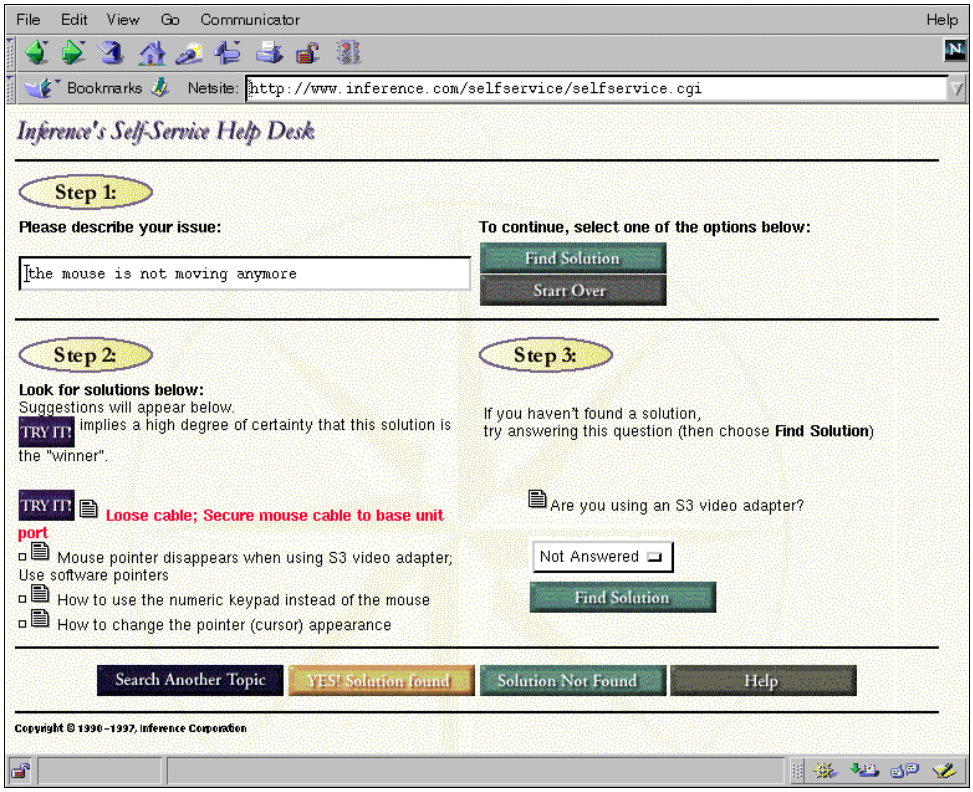


Figure III-8 : résolution d'un problème technique avec CasePoint WebServer

CBR 3 comprend CBR Express un outil pour le développement des bases de cas (création et maintenance). De plus, un générateur de cas est également fourni pour construire les cas automatiquement à partir d'une documentation textuelle existante. CasePoint est le nom du module permettant d'effectuer la recherche. L'utilisation de ce produit peut se faire au travers d'un browser Web classique pour un déploiement sur Internet ou Intranet grâce au module CasePoint WebServer. Un kit de développement permet de personnaliser la phase de recherche avec l'outil CasePoint Search SDK. Par exemple, la recherche peut être effectuée sans aucune interaction avec l'utilisateur, la recherche et les réponses aux questions sont alors guidées par des événements et des modules extérieurs.

The Easy Reasoner[28] est une bibliothèque C++ de raisonnement à partir de cas reposant sur trois modules importants : TIM (*The Intelligent Memory*) pour une indexation par plus proches voisins, ClassIE (*Class Induction Engine*) pour la construction d'arbres d'induction et Eclipse un moteur d'inférence. Ainsi, *The Easy Reasoner* propose différent index et permet l'utilisation de règles pour l'adaptation. Cet outil ne dispose pas d'un environnement graphique pour le développement, mais

[28] *The Easy reasoner*, CPR et Help!CPR sont commercialisés par la société *The Haley Entreprise*, http://www.haley.com.

offre un ensemble de fonctions qui peuvent être directement embarquées dans une application spécifique. De plus, la connexion avec des systèmes de bases de données relationnelles est possible pour stocker les cas. Associée à cet outil, une autre bibliothèque nommée CPR[28] (*Case-based Problem Resolution*) aborde plus particulièrement l'application du RàPC à des problèmes d'assistance technique, de manière similaire à CBR 3. Dans CPR, uniquement le module TIM est utilisé. Un cas[29] est composé d'une description, d'un ensemble de questions et d'un ensemble d'actions à mener pour résoudre un problème. L'application Help!CPR[28], développée à partir de CPR, montre l'utilisation concrète de cette bibliothèque pour construire un système d'aide au diagnostic de pannes pouvant opérer sur plusieurs bases de cas. La Figure III-9 donne l'exemple d'un cas dans Help!CPR pour la résolution de pannes liées à du matériel informatique.

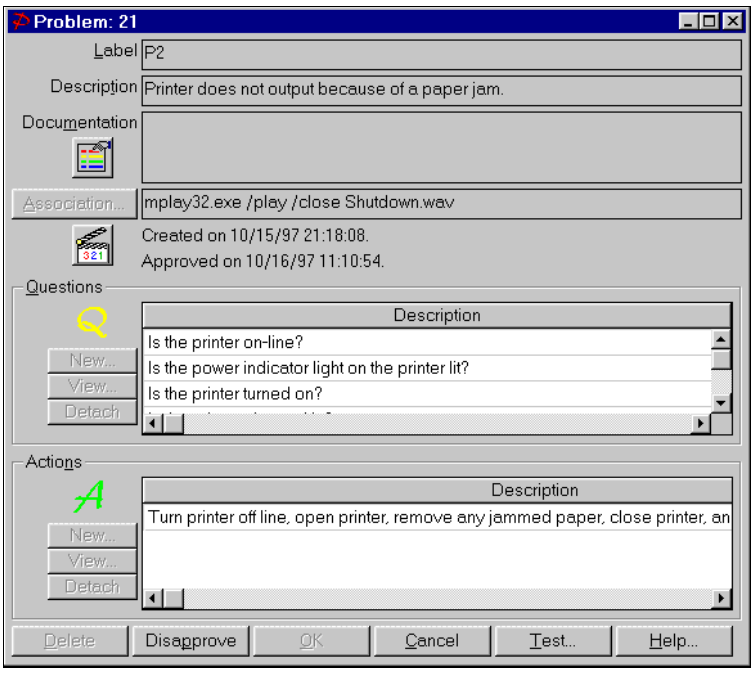


Figure III-9 : représentation d'un cas dans Help!CPR

Pour finir, nous citons brièvement d'autres outils qui sont plus amplement décrits dans (Watson, 1997) : ART*Entreprise, Esteem, Recall, CBRWorks et Caspian. ART*Entreprise[30] est un ensemble d'outils permettant le développement des systèmes experts et intègre des fonctionnalités similaires à CBR 3 pour le raisonnement à partir de cas. L'accès à un langage de programmation à objets permet d'étendre et de personnaliser les applications. Esteem[31] est également un outil reposant sur un moteur d'inférence permettant d'utiliser des règles pour l'adaptation et propose différentes méthodes de recherche. Recall[32] est un outil offrant des index par plus proches voisins et par arbre d'induction ainsi que des mécanismes d'adaptation par formules. La représentation des cas suit un modèle à objets, ce qui permet de structurer les données. Cet outil offre également une bibliothèque C++ pour son utilisation et son extension dans des applications spécifiques. CBRWorks[33] est un outil qui permet une représentation structurée des cas et dont les mesures de similarité peuvent être étendues. Enfin, Caspian[34] permet d'effectuer une recherche par plus proches voisins et utilise des règles pour

[29] Dans CPR, un cas est appelé un problème.
[30] http://www.brightware.com
[31] Esteem Software, 302 E. Main street, Cambridge City, IN 47327, USA.
[32] http://www.alice.fr/
[33] http://www.tecinno.de/tecinno_e/ecbrwork.htm
[34] http://www.aber.ac.uk/~dcswww/Research/arg/cbrprojects/getting_caspian.html

l'adaptation. Cet outil peut être configuré par la définition d'une base de cas et des règles d'adaptation dans un fichier dont le format est spécifique.

1.2 Limites des outils existants

Les outils existants permettent de réaliser différentes applications et sont donc réutilisables au regard d'un type ciblé d'applications. Cependant, sur le plan de l'ouverture, ils présentent deux difficultés majeures portant sur la modification les composants existants et sur l'intégration de nouveaux composants.

1.2.1 Difficulté de modifier les composants proposés

Chaque outil propose une ou plusieurs techniques d'indexation, d'adaptation et de manipulation des cas. Chaque méthode repose sur de multiples choix de conception qui ont été adoptés par les concepteurs de l'outil mais qui ne sont plus modifiables par l'utilisateur. Par exemple, dans REMIND, les fonctions élémentaires de similarité et la fonction d'agrégation, utilisées dans l'index par plus proches voisins, ne sont pas modifiables. Pourtant, ces fonctions ne dépendent pas que du domaine de valeurs de l'indice, mais aussi de sa sémantique et des connaissances du domaine. Ainsi, différentes fonctions devraient pouvoir être utilisées. De même, pour un index par arbre d'induction, de nombreuses fonctions de calcul du gain d'information ou du traitement des valeurs manquantes sont possibles, mais ne sont pas modifiables par l'utilisateur.

Par exemple, nous donnons la description d'un extrait de l'interface de programmation de la bibliothèque CPR, dans lequel nous voyons qu'il est possible de spécifier la méthode qui sera utilisée pour ordonner les actions proposées par le système (cf. Figure III-10). Bien que trois choix soient proposés, l'utilisation d'une énumération ne permet pas de définir sa propre méthode de tri. En revanche, la définition d'une classe d'objets responsables de ce calcul aurait permis la réalisation d'une classe spécifique implantant la méthode désirée.

```
enum CPRscoringMethod {
        cprsmAverage,
        cprsmCumulative,
        cprsmMaximum
};

// Sets the scoring method being used. If the scoring method changed,
// then it returns TRUE and removes all of the ranked objects, but it
// retains any answers. Otherwise, it returns FALSE.
DLLEXP THE_BOOLEAN SetActionScoring(CPRscoringMethod _actionScoring);
```

Figure III-10 : extrait de l'interface de programmation de CPR

Cet exemple est révélateur de l'objectif de ces outils et de leur interface de programmation. Chaque outil (sauf *The Easy Reasoner*) dispose d'une interface graphique évoluée par laquelle les cas et les méthodes de raisonnement sont paramétrés et utilisés. Ces outils sont généralement pourvus d'une interface de programmation et fournissent ainsi le moyen de gérer plus directement ces paramètres et parfois de manière plus étendue (par exemple, Recall permet la définition de formules d'adaptation spécifique par l'utilisation du langage C++). Cependant l'objectif de ces interfaces de programmation n'est pas de fournir un ensemble de composants ouverts, mais seulement de fournir des points d'entrée pour directement utiliser l'outil, tout en développant une interface graphique

spécifique. Ainsi, les composants fournis dans ces interfaces de programmation restent des *boîtes noires* qu'il n'est possible de modifier que suivant des possibilités préétablies.

1.2.2 Difficulté d'intégrer de nouveaux composants

L'intégration de nouveaux composants n'est abordée que dans le projet FABEL et le système REPRO. En effet, l'approche prise dans le projet FABEL est très intéressante et montre comment il est possible de construire un système de raisonnement à partir de cas en assemblant différents composants répartis. Une méthodologie en plusieurs étapes est d'ailleurs proposée pour ajouter un composant dans la construction d'une application (Gebhardt *et al.,* 1997, page 208). Cependant, l'utilisation de ces composants reste lourde et complexe : ils sont obligatoirement répartis en plusieurs processus, et ils sont hétérogènes (différents langages sont utilisés). De son côté, REPRO aborde uniquement la phase de recherche et offre la possibilité de définir différents algorithmes puis de les sélectionner suivant les caractéristiques de l'application. Les autres outils proposent une ou plusieurs méthodes, mais il est impossible d'intégrer un nouvel algorithme, à moins de reprendre l'approche de REPRO qui utilise un outil existant (REMIND) à travers son interface de programmation, et qui enrichit ces possibilités. Dans ce cas, l'intégration est possible mais entièrement opérée par le programmeur sans aide de l'outil utilisé.

L'intégration de nouveaux algorithmes peut permettre de remplacer les algorithmes existants ou de les combiner. Pour la phase de recherche, REMIND donne l'exemple le plus complet de combinaisons d'algorithmes et de structures nécessaires. Cependant, aucune approche ne propose un cadre général et unifié pour la combinaison d'algorithmes existants et nouveaux pour l'ensemble des étapes du raisonnement.

2 Proposition d'une plate-forme à objets pour le RàPC

Pour dépasser les deux principales limites des outils existants énoncées précédemment, nous proposons une approche originale dans le domaine du RàPC tirant profit du concept de plate-forme à objets (Fayad & Schmidt, 1997 ; Johnson, 1997 ; Johnson & Foote, 1988). Une plate-forme à objets est souvent présentée comme l'approche à objets la plus flexible pour gérer la réutilisation de composants et d'architectures. Dans un premier temps, nous introduisons les principes fondamentaux d'une plate-forme, en mettant en avant, les avantages et les limites d'une telle approche ainsi que les techniques couramment utilisées. Nous montrons alors l'adéquation de cette approche pour la modélisation ouverte et réutilisable du raisonnement à partir de cas. Enfin, nous argumentons notre approche par la présentation des points d'ouverture de la plate-forme que nous avons réalisée.

2.1 Principes d'une plate-forme à objets

Nous mettons tout d'abord en évidence les principales caractéristiques des plates-formes à objets, puis nous précisons la problématique de conception d'une plate-forme ainsi que le rôle des patrons de conception (*design pattern*). Enfin, nous donnons les principes d'utilisation d'une plate-forme.

2.1.1 Définition et propriétés d'une plate-forme

Le concept de plate-forme à objets remonte aux années 1980, et la définition initiale décrit une plate-forme comme « un ensemble de classes qui représente une conception abstraite d'une solution pour une famille de problèmes, et qui permet la réutilisation à un grain plus large que celui de la

classe » (Johnson & Foote, 1988). L'objectif d'une plate-forme est donc très ambitieux quant à ses possibilités de réutilisation et doit notamment permettre :

- la *réutilisation d'une architecture complexe* sous la forme d'un ensemble de classes abstraites définissant l'interface et le flux de contrôle de la plate-forme,
- la *réutilisation de composants de base* réalisant certaines parties de la plate-forme représentées par des classes concrètes d'objets.

Une plate-forme est similaire à un générateur d'applications (Johnson, 1997). Dans les deux cas, il s'agit d'outils permettant de construire plusieurs applications concrètes partageant un même modèle. Un générateur utilise une représentation interne spécifique de ce modèle servant à générer l'application. Cependant, la particularité d'une plate-forme est de représenter ce modèle dans le même langage que celui utilisé pour l'implantation, en exploitant les possibilités offertes par le paradigme objet. Une plate-forme peut alors être considérée comme une application incomplète, définissant une architecture et proposant certaines variations. Lors de la réalisation d'une application finale, il s'agit d'analyser les exigences vis-à-vis des fonctionnalités offertes dans la plate-forme pour choisir ou modifier les composants existants afin de respecter les exigences de réalisation.

Une plate-forme est bien plus qu'une bibliothèque de composants logiciels. Une bibliothèque définit un ensemble de classes qui peuvent être réutilisées indépendamment ou en petits groupes. En effet, il faut choisir parmi l'ensemble des composants ceux qui sont utiles, et c'est au programmeur d'écrire les algorithmes qui permettront leur utilisation dans le cadre de l'application visée. De son côté, une plate-forme définit une architecture de collaboration entre objets, en plus d'un ensemble de composants de base comme dans une bibliothèque. Le *flux de contrôle est alors inversé* : le code spécifique lié à la réalisation d'une application est appelé par la plate-forme, et ce n'est pas au programmeur de concevoir ni de réaliser la logique de l'application. Toutefois, le flux de contrôle proposé dans la plate-forme peut parfois être modifié. Il est alors souvent fait référence au principe d'Hollywood : « Ne nous appelez pas, nous vous appellerons »[35].

Bien que le concept de plate-forme ne soit pas nouveau, il présente des atouts essentiels pour les développements actuels d'outils. En effet, la tendance de l'industrie du logiciel est d'offrir des outils interopérables sachant qu'il devient inévitable, au vu de la complexité des systèmes, de réunir des composants provenant de fournisseurs divers. Ainsi, le succès du langage Java de Sun Microsystems repose en partie sur la conception et l'intégration de différentes plates-formes dont notamment RMI (*Remote Method Invocation*) pour les communications entre processus et l'AWT (*Abstract Window Toolkit*) pour l'interface graphique. Dans le domaine de l'intelligence artificielle, nous pouvons également citer comme exemple la plate-forme pour la réalisation de systèmes interactifs pour la résolution de problèmes par satisfaction de contraintes (JCL)[36].

2.1.2 Conception d'une plate-forme

La conception d'une plate-forme à objets est reconnue comme une des activités les plus délicates dans le domaine de la conception de logiciels à objets (Johnson & Foote, 1988 ; Gamma *et al.,* 1995). Trois types de logiciels à objets peuvent être identifiés : une application, une bibliothèque et enfin une plate-forme à objets. Alors qu'une application répond à des exigences limitées à un cadre bien précis, une bibliothèque doit fournir un ensemble de composants réutilisables pour un domaine ou un type de traitement. La conception d'une bibliothèque est donc sensiblement plus complexe qu'une application. Une plate-forme doit non seulement offrir un certain nombre de composants de base

[35] « *Don't call us, we'll call you* »
[36] Ecole Polytechnique Fédéral de Lausanne, http://liawww.epfl.ch/~torrens/Project/JCL/jclhome.html

comme dans une bibliothèque mais doit en plus définir une architecture réutilisable dans laquelle ces différents composants collaborent.

Dans ce cadre, il est alors plus difficile de garantir cette réutilisabilité *a priori*, c'est pourquoi le processus de conception d'une plate-forme est essentiellement itératif. Ces itérations mettent en jeu d'une part une phase de conception et de réalisation, et d'autre part une phase de réutilisation. Les échecs et les difficultés rencontrés lors de la réutilisation sont alors utilisés dans une nouvelle phase de conception pour améliorer la plate-forme. Selon la règle, en effet, un logiciel n'est réutilisable que s'il a été concrètement réutilisé. Une certaine stabilité doit en définitive être obtenue par la disparition des difficultés de réutilisation. Cependant, cette stabilité peut ne jamais être atteinte à cause des évolutions des exigences fonctionnelles des différentes applications à réaliser, bien que les changements soient de moins en moins fréquents (Codenie *et al.,* 1997). Une analyse du domaine visé et l'expérience issue de développements antérieurs d'applications sont toutefois très utiles pour commencer ce processus (cf. Figure III-11).

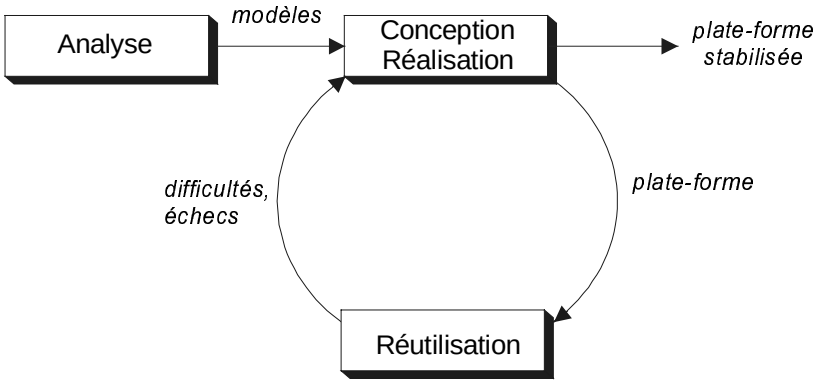


Figure III-11 : cycle de développement d'une plate-forme

La phase de conception repose sur l'identification de points d'ouverture ou « *hot spots* » (Schmid, 1997). Un point d'ouverture représente un comportement susceptible d'être différent dans les applications visées. Les différents comportements peuvent alors être choisis dans une liste de comportements prédéfinis (instanciation), nous parlons alors d'un point d'ouverture à boîte noire (cf. Figure III-12) car il n'est nul besoin de connaître précisément comment ce comportement est implanté et comment les collaborations à l'intérieur de la plate-forme sont effectuées. Toutefois, ce comportement peut être défini pour chaque application par le programmeur (spécialisation), il s'agit alors d'un point d'ouverture à boîte transparente puisque, dans ce cas, il est nécessaire de connaître exactement la façon de définir ce comportement. En règle générale, un point d'ouverture n'est ni purement à boîte noire ni purement à boîte transparente et offre à la fois des comportements prédéfinis ainsi que la possibilité de fournir son propre comportement. Toutefois, un point d'ouverture qui atteint une certaine maturité aura tendance à devenir une boîte noire, dans la mesure où différents comportements auront été identifiés au cours des développements successifs. Cette différenciation peut également être remontée au niveau global d'une plate-forme, et nous pouvons parler de *plate-forme à boîte noire* ou de *plate-forme à boîte transparente* suivant les caractéristiques de l'ensemble de ses points d'ouverture.

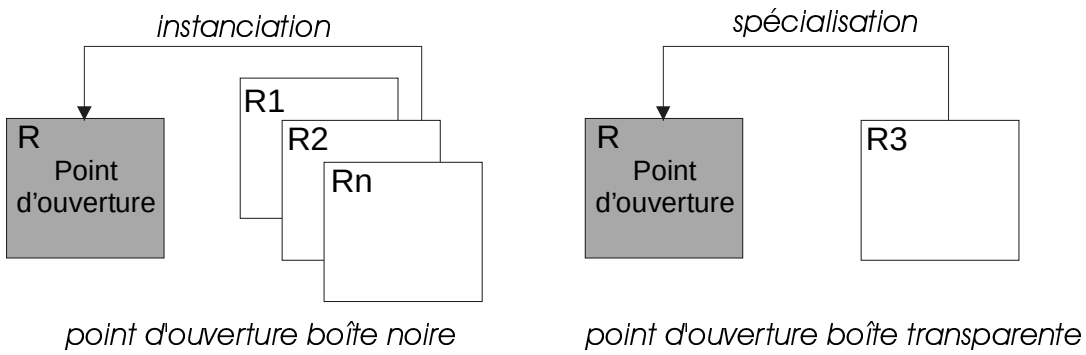


Figure III-12 : types de points d'ouverture (d'après Schmid, 1997)

Pratiquement, un point d'ouverture est représenté par une classe *abstraite* (cf. Figure III-13). Une classe abstraite définit un ensemble de méthodes dont l'implantation est déléguée à des sous-classes concrètes. Une classe abstraite ne peut alors être instanciée mais définit l'interface du point d'ouverture. Les différentes classes concrètes doivent respecter cette interface et peuvent être instanciées pour être utilisées dans la constitution d'une application.

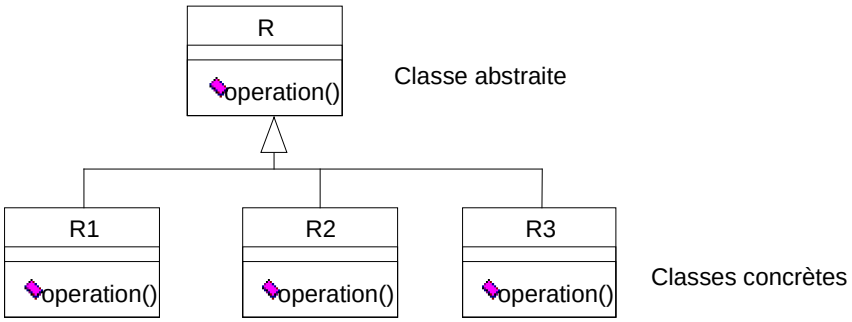


Figure III-13 : représentation d'un point d'ouverture

L'identification de ces points d'ouverture peut être basée sur la définition *d'axes de variabilité* et de la prise en compte d'exigences liées à la conception d'un système ouvert : interopérabilité, distribution et extensibilité (Demeyer *et al.,* 1997). Par exemple, dans le cas d'un système ouvert de gestion d'un hypermédia, il est proposé trois axes : stockage (protocoles d'accès aux nœuds de l'hypermédia), présentation (techniques et outils de visualisation) et navigation (manières de lier les nœuds). La prise en compte de ces axes et des contraintes d'ouverture mène à la création de classes de base, représentant des points d'ouverture pour, par exemple, séparer les responsabilités du chargement, de la visualisation et de la résolution des liens entre nœuds de l'hypermédia.

2.1.3 Rôles des patrons de conception dans une plate-forme à objets

La conception d'une plate-forme et des points d'ouverture étant une activité complexe, l'utilisation de *patrons de conception* permet d'anticiper des problèmes d'ouverture et facilite la compréhension des décisions de conception. Un patron de conception (Gamma *et al.,* 1995) définit une conception typique répondant à un problème récurrent dans la conception de logiciels à objets. Un patron expose ses avantages et ses inconvénients suivant son contexte d'application. Les patrons abordent généralement des problèmes élémentaires de conception (création d'objets, structuration des objets ou comportement entre objets) bien que des patrons composés soient également proposés (Riehle, 1997). Alors qu'une plate-forme vise des modèles réutilisables dans un domaine d'application, les patrons sont plus abstraits et sont réutilisables à travers les domaines. Par exemple, le patron de conception *Composite* permet d'organiser les objets en une structure arborescente pour manipuler des objets simples et des objets composites de la même façon. Ce patron peut être utilisé pour concevoir une

hiérarchie de composants graphiques qui seront alors manipulés suivant la même interface qu'ils soient unitaires ou composites (cf. Figure III-14).

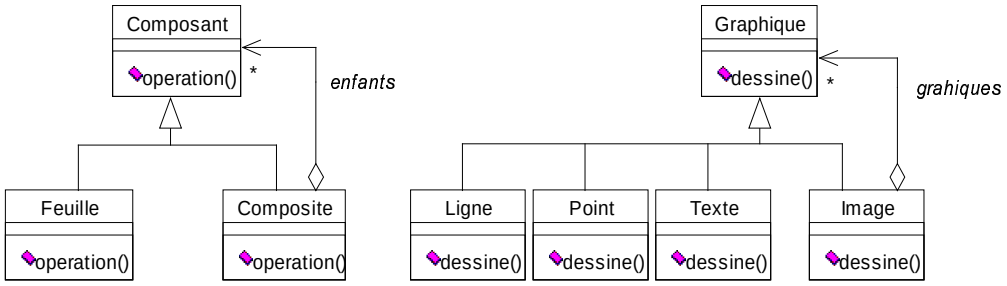


Figure III-14 : patron de conception *Composite* et son application pour des objets graphiques

L'apport des patrons pour une plate-forme à objets se situe tout d'abord au niveau de la conception. L'utilisation de patrons dans une plate-forme permet en effet de réutiliser l'expérience d'experts ayant conçu ces patrons : des problèmes de réutilisation et de flexibilité sont ainsi anticipés. Les patrons constituent également un *langage* qui permet aux équipes de développement de mieux communiquer (Schmidt, 1995). De même, l'intégration des patrons dans la documentation d'utilisation facilite la compréhension de la plate-forme (Meusel *et al.*, 1997 ; Odenthal & Quibeldey-Cirkel, 1997). De plus, l'utilisation de ce langage abstrait n'est pas liée à un domaine d'application et permet ainsi l'évaluation des choix de conception par des spécialistes d'autres domaines. Les patrons évitent donc le cloisonnement d'une communauté (dans notre cas, celle du raisonnement à partir de cas) pour profiter de l'expérience d'autres communautés.

Les patrons sont également importants pour l'utilisation d'une plate-forme. Tout d'abord, les décisions de conception peuvent être commentées en termes de patrons pour permettre une rapide compréhension de sa conception. De plus, cette description abstraite amène un niveau de réutilisation supplémentaire. En effet, en plus de la réutilisation directe de la plate-forme, le modèle ainsi décrit peut être utilisé en partie ou totalement dans la réalisation d'applications lorsque l'utilisation directe de la plate-forme initiale ne peut être effectuée (Schmidt, 1995).

Toutefois, les patrons et leur utilisation présentent deux limites importantes. Tout d'abord, ils n'apportent aucune garantie dans les propriétés de la modélisation. C'est pourquoi, l'analyse des besoins doit motiver l'utilisation d'un patron : l'objectif n'est pas d'appliquer un patron pour introduire une flexibilité inutile qui ne servirait qu'à rendre plus complexe ou inefficace les applications construites. Ensuite, les patrons permettent de réaliser de nombreux points d'ouverture dont il faut ensuite assurer la cohérence dans une application concrète. Cette cohérence peut parfois être vérifiée par des patrons eux-mêmes comme la *Fabrique Abstraite* (Rüping, 1996).

2.1.4 Utilisation d'une plate-forme

L'utilisation (ou la réutilisation) d'une plate-forme se fait par analyse des différences entre les exigences d'une application et les fonctionnalités offertes. Cette analyse permet de déterminer comment les points d'ouverture doivent être configurés. Cette configuration peut se faire soit par instanciation d'un composant concret, soit par spécialisation d'un composant existant ou d'une classe abstraite de base. L'objectif est de minimiser l'effort de conception et de réalisation en se basant sur l'architecture de la plate-forme et ses composants. Cet objectif s'inscrit également dans le temps et l'espace des applications. En effet, dans le long terme, l'utilisation d'une plate-forme réduit les coûts de maintenance puisque cette maintenance est alors identique à une phase de réutilisation. De plus, de par sa conception itérative, la plate-forme peut être enrichie par de nouveaux composants. Enfin,

différentes applications peuvent être développées et maintenues plus facilement puisqu'elles reposent sur des architectures uniformes (Johnson, 1997).

L'enjeu d'une plate-forme est donc important, cependant l'utilisation effective d'une plate-forme nécessite une attention particulière sur certaines caractéristiques du développement, notamment : la documentation et la gestion de la cohérence. En effet, la réutilisation ne concerne jamais une seule classe et un ensemble de classes doit être assimilé pour la réutilisation : le programmeur doit alors avoir une connaissance globale de l'architecture et du fonctionnement de la plate-forme, même si les classes à modifier sont généralement petites. Cette connaissance est donc plus difficile et plus longue à acquérir que pour la réutilisation d'une classe isolée d'une librairie. De plus, si des points d'ouverture à boîte transparente doivent être spécialisés, une compréhension approfondie est alors nécessaire. Enfin, la gestion de la cohérence de l'utilisation d'une plate-forme reste un problème important et ouvert. Il s'agit en effet d'assurer que les modifications apportées dans une sous-classe respectent les hypothèses faites dans la conception des superclasses. Cette cohérence doit être assurée au niveau de chaque classe mais aussi lors des modifications d'un ensemble de points d'ouverture, voire même lors de la combinaison de plusieurs plates-formes pour une même application (Mattsson & Bosch, 1997).

C'est pourquoi la documentation d'une plate-forme est très importante pour faciliter son utilisation. Trois niveaux de documentation sont d'ailleurs identifiés (Johnson, 1992) :

- *documentation d'introduction* décrivant l'objet de la plate-forme qui doit permettre d'évaluer si la plate-forme répond aux besoins et aux contraintes d'une application,
- *documentation d'utilisation directe* permettant de construire des applications typiques de la plate-forme,
- *documentation détaillée de conception* permettant de construire des applications qui ont besoin d'étendre les composants de base offerts par la plate-forme.

Différentes techniques sont alors utilisées pour rédiger pratiquement ces documentations : schémas d'utilisation (Johnson, 1992 ; Meusel *et al.*, 1997) présentant des cas typiques d'utilisation avec les détails de conception associés, présentation structurée des actions à effectuer pour utiliser la plate-forme (*hook*, Froedlish *et al.*, 1997), patrons de conception (Gamma *et al.,* 1995 ; Meusel *et al.*, 1997 ; Odenthal & Quibeldey-Cirkel, 1997) et contrats de réutilisation (Lajoie & Keller, 1994) qui décrivent les contraintes de collaboration entre classes.

2.2 Choix d'une plate-forme comme outil réutilisable et ouvert

La conception d'une plate-forme pour le raisonnement à partir de cas présente les caractéristiques de l'outil recherché en termes de réutilisabilité et d'ouverture pour représenter, d'une part les éléments de base du raisonnement (objets-métier), et d'autre part le raisonnement lui-même. En effet, le raisonnement à partir de cas s'appuie premièrement sur des données représentées par des cas, sur des structures d'indexation, et sur des algorithmes d'adaptation et d'apprentissage. Ces éléments sont généralement différents dans chaque application. Une plate-forme fournit un ensemble de classes concrètes directement réutilisables par assemblage qui peuvent représenter ces éléments. L'ajout de nouveaux composants est facilité par la définition d'interfaces abstraites qui peuvent être étendues par la définition de nouvelles classes concrètes. Les comportements existants peuvent également être modifiés par spécialisation ou par l'utilisation de paramètres existants.

De plus, un raisonnement doit permettre le contrôle des algorithmes et la gestion de la cohérence globale. La représentation d'un raisonnement nécessite donc la gestion de plusieurs concepts et d'algorithmes qui interviennent en coordination. Une simple librairie de composants est incapable de représenter ce niveau du raisonnement et l'utilisateur devrait concevoir et réaliser à chaque fois le

module principal qui coordonne les composants utilisés. Une plate-forme à objets offre une architecture abstraite de collaboration entre objets permettant de lier un ensemble de concepts de manière homogène. Cette architecture est capable de refléter le fonctionnement du raisonnement et d'intégrer les différents composants nécessaires. Enfin, elle offre les possibilités de réutilisation et de modification des mécanismes du contrôle du raisonnement.

Ainsi, une plate-forme présente non seulement l'atout de permettre la réutilisation de composants utiles pour le raisonnement, mais peut aussi faciliter la réutilisation et l'extension du contrôle.

2.3 Analyse des points d'ouverture de notre plate-forme à objets

Nous argumentons notre approche par l'analyse des points d'ouverture de notre plate-forme. Ces points d'ouverture sont principalement structurés suivant deux critères (cf. Tableau III-1) :

- *Spécificité*. Deux niveaux de spécificité sont identifiés : le premier regroupe les points d'ouverture généraux liés au raisonnement à partir de cas et le second regroupe les points d'ouverture supplémentaires ou spécialisés, nécessaires pour l'indexation par situations comportementales.
- *Axes de variabilité*. Un axe de variabilité définit une dimension dans laquelle les différentes applications nécessitent de la flexibilité et sont susceptibles de varier (Demeyer *et al.,* 1997). Nous avons identifié trois axes de variabilité regroupant chacun un ensemble de points d'ouverture : gestion du raisonnement, représentation des cas et organisation de la mémoire.

	Axe « gestion du raisonnement »	***Axe « représentation des cas »***	***Axe « organisation de la mémoire »***	***Total***
Niveau « RàPC »	11	4	11	*26*
Niveau « indexation par situations comportementales »	(1)	10 (1)	2 (1)	*12*
Total	*11*	*14*	*13*	*38*

Tableau III-1 : structuration et dénombrement des points d'ouverture[37]

Pour représenter cette structuration, nous utilisons le formalisme des *cas d'utilisation* (cf. annexe A). Un cas d'utilisation représente un type d'interaction entre un utilisateur (le concepteur d'une application) et un système (formé par la plate-forme complétée de son environnement : éditeur graphique ou textuel par exemple). Dans ce contexte, un cas d'utilisation représente une *tâche* que doit réaliser le concepteur pour construire une application (cf. Figure III-15). Ce formalisme a été choisi car il permet de représenter de manière simple non seulement les liens entre les deux niveaux de points d'ouverture mais aussi la décomposition en axes de variabilité.

[37] Les points d'ouverture comptabilisés entre parenthèses correspondent à des points d'ouverture qui *remplacent* des points d'ouverture du niveau « RàPC » (c'est la cas pour TimeExtendedReasonerFactory, TimeExtendedCase et TimeExtendedMemory qui remplacent respectivement ReasonerFactory, CbrCase et Memory). Il ne s'agit donc pas de points d'ouverture supplémentaires et ne sont pas comptabilisés dans le total des points d'ouverture de la plate-forme.

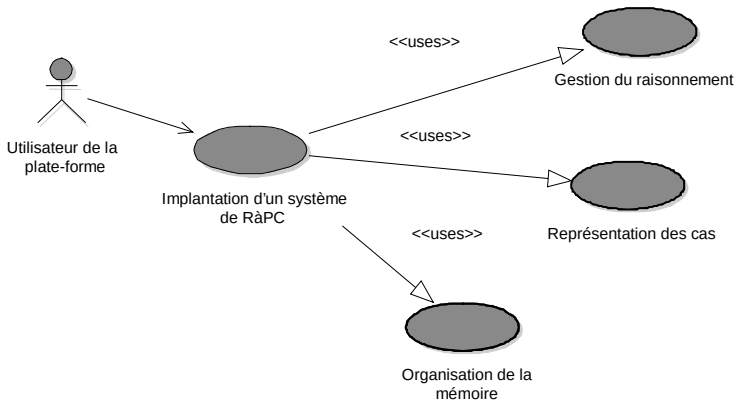


Figure III-15 : décomposition des points d'ouverture en trois axes de variabilité

2.3.1 Points d'ouverture du RàPC

Nous présentons, dans cette section, l'analyse des besoins de flexibilité d'un moteur de raisonnement à partir de cas que prend en compte notre plate-forme. Notre analyse est basée sur les travaux de synthèse existants (Aamodt & Plaza, 1994 ; Gebhardt *et al.,* 1997), sur l'analyse des outils existants (cf. §1), et sur trois applications que nous avons réalisées avec des versions préliminaires de notre plate-forme ou dans d'autres environnements (Jaczynski & Trousse, 1994). Cette analyse suit les trois axes de variabilité identifiés.

2.3.1.1 *Gestion du raisonnement*

Le premier axe de variabilité représente la *gestion du raisonnement* : il recouvre le contrôle global du raisonnement ainsi que la définition des différentes phases (cf. Figure III-16). Tout d'abord, bien que les différentes phases du raisonnement soient souvent exécutées de manière séquentielle (recherche, réutilisation, révision et enfin apprentissage), d'autres gestions du raisonnement peuvent être nécessaires (point d'ouverture Reasoner). Par exemple, l'échec de la phase de réutilisation peut mener à exécuter une nouvelle phase de recherche. Ainsi, des interactions plus complexes entre les phases peuvent être observées (Gebhardt *et al.,* 1997, page 185). De plus, le raisonnement peut ne pas se dérouler d'une traite. En effet, les phases de révision ou d'apprentissage peuvent avoir besoin de données issues du monde réel (exécution de la solution, explication d'un échec par des experts) qui ne sont pas disponibles immédiatement. Cependant, le système doit être capable entre-temps de continuer à raisonner sur d'autres problèmes. Par exemple, dans la nutrition des plantes, la phase d'apprentissage ne peut être effectuée que lorsque l'évolution de la concentration du drainage est connue sur les cinq jours qui suivent la décision de commande ; entre-temps d'autres raisonnements doivent pouvoir être menés. Le raisonnement peut alors être suspendu puis repris grâce à la sauvegarde de son état (point d'ouverture Reasoning).

Enfin, le raisonnement à partir de cas ne propose pas une unique méthode pour la recherche, la réutilisation, la révision et l'apprentissage. Différentes techniques doivent être mises en place suivant notamment les connaissances du domaine et le degré d'automatisation souhaité du raisonnement (points d'ouverture Retrieve, Reuse, Revise et Retain). Les outils existants de RàPC montrent bien cette diversité. Par exemple, le projet FABEL repose sur un ensemble de techniques pour la recherche et l'adaptation. Pour fonctionner, ces phases utilisent les résultats des phases exécutées précédemment dans un même raisonnement (points d'ouverture RetrieveResult, ReuseResult, ReviseResult et RetainResult). Enfin, la gestion du raisonnement demande la création d'un ensemble d'objets (cas cible, objet de sauvegarde de l'état du raisonnement) qu'il est nécessaire de paramétrer (point d'ouverture ReasonerFactory).

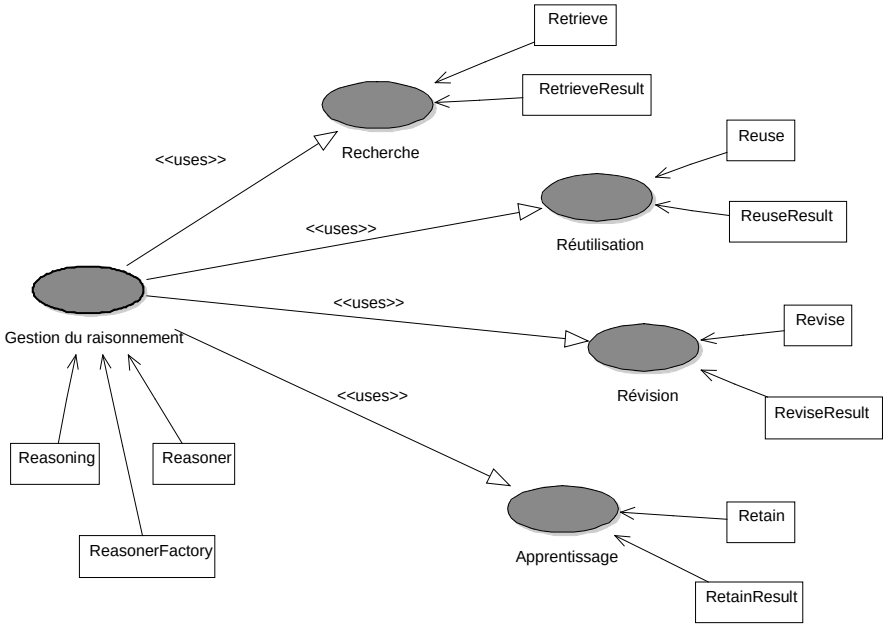


Figure III-16 : points d'ouverture pour la gestion du raisonnement

2.3.1.2 *Représentation des cas*

Le deuxième axe de variabilité repose sur la *représentation des cas* et de leurs indices. La tâche de représentation des cas comporte en effet une sous-tâche importante correspondant à la représentation des indices (cf. Figure III-17). La représentation générale d'un cas (point d'ouverture CbrCase) comprend la description de la solution, des données de gestion ainsi que l'accès aux indices. Dans la plupart des outils existants, la représentation des indices est linéaire, cependant une structuration est souvent nécessaire dans les domaines complexes (KATE, FABEL, CBRWorks) et dépend de la modélisation du domaine et de l'utilisation des indices durant le raisonnement (point d'ouverture CompoundIndice[38]). Les indices peuvent être définis sur de nombreux domaines de valeurs (point d'ouverture IndiceType) par exemple les entiers, les réels, ou des symboles et avec différentes propriétés (domaine ordonné et/ou dénombrable par exemple). Enfin, chaque indice peut avoir un poids d'importance donnant son degré de pertinence au regard des autres indices. L'affectation et le calcul de ces poids peuvent se faire suivant différentes méthodes (point d'ouverture WeightVector) : utilisation de poids par défaut, affectation manuelle par l'utilisateur ou calcul dynamique des poids.

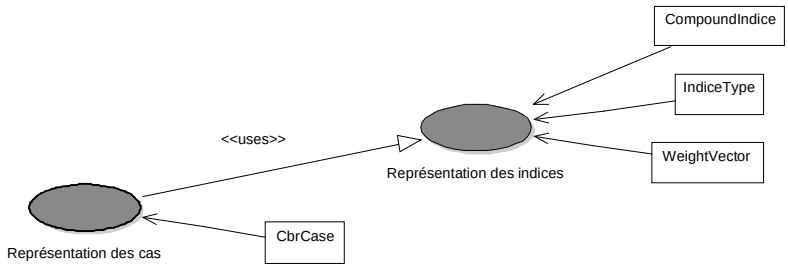


Figure III-17 : points d'ouverture pour la représentation des cas

[38] Ce point d'ouverture est appelé CompoundIndice et non Indice car nous utilisons une représentation dans laquelle l'élément de base est un *indice composé* issu de la définition d'une classe d'indices caractérisant les propriétés des sous-indices (cf. chapitre IV, §1.2.2).

2.3.1.3 *Organisation de la mémoire*

Le troisième axe représente *l'organisation de la mémoire* qui contient les cas sources et les organise suivant des structures de données. Nous remarquons tout d'abord que l'organisation de la mémoire doit servir d'une part au stockage et à la consultation des cas, et d'autre part à la recherche des cas utiles pour le raisonnement. Nous proposons donc de décomposer la réalisation de l'organisation de la mémoire en deux sous-tâches principales (cf. Figure III-18) : *l'organisation des cas* et *l'organisation de l'indexation*. La mémoire (point d'ouverture Memory) gère la cohérence de ces deux organisations. L'organisation des cas recouvre l'étude du stockage physique des cas dans différentes bases issues des systèmes d'information utilisés (Brown *et al.,* 1995) et du cycle de vie dans lequel ils peuvent être créés, enregistrés, validés par des experts ou finalement effacés de la mémoire (point d'ouverture CaseBase).

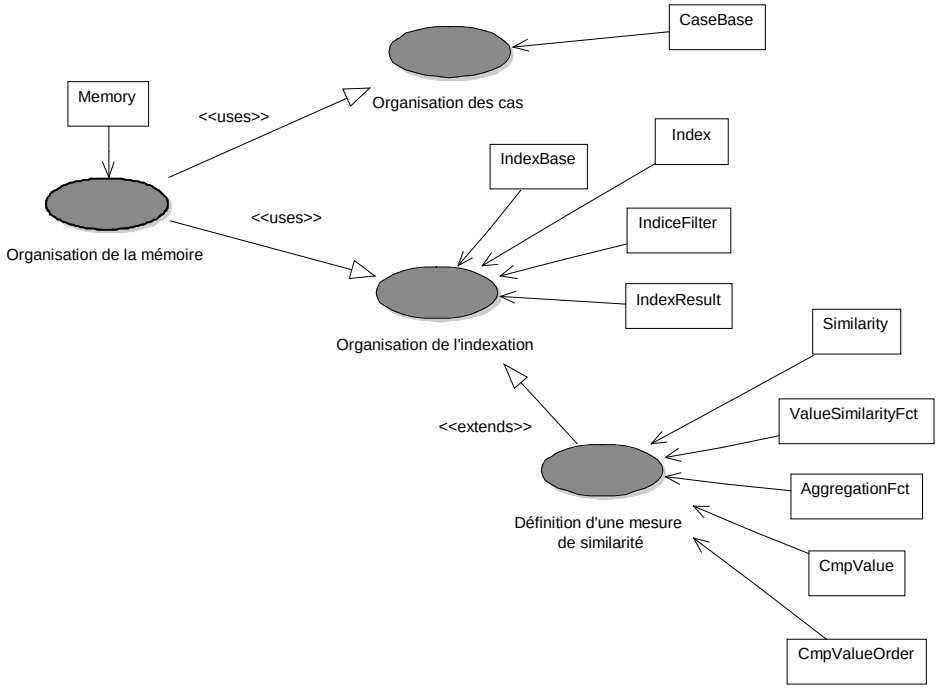


Figure III-18 : points d'ouverture pour l'organisation de la mémoire

L'organisation de l'indexation vise à définir des index qui permettront de retrouver les cas lors de la phase de recherche du raisonnement et qui peuvent également être mises à jour lors de l'apprentissage. Comme nous l'avons montré dans l'analyse des outils existants, différents index sont couramment utilisées (point d'ouverture Index) : organisation linéaire, arbre discriminant construit *a priori* ou par induction. De plus, il est nécessaire de pouvoir faire évoluer ces index durant le cycle de vie du système (cf. le système REPRO) afin d'inclure de nouvelles connaissances ou de faire face au nombre croissant des cas ajoutés en mémoire. Il est donc nécessaire de définir simultanément plusieurs stratégies d'indexation pour les évaluer et les faire évoluer (point d'ouverture IndexBase). Enfin, les index peuvent ne prendre en compte qu'une partie des indices des cas (point d'ouverture IndiceFilter) et retournent différents types d'informations (point d'ouverture IndexResult) : ensemble de cas, ensemble de prototypes, analyse détaillée des indices.

Ces différents index utilisent notamment des mesures de similarité (point d'ouverture Similarity) permettant de comparer les cas sources au cas cible. Ces mesures retournent des évaluations (points d'ouverture CmpValue) qui peuvent être de différents types : généralement un facteur entre 0 et 1, ou entre 0 et 100 comme dans REMIND ou bien un couple possibilité/nécessité (Jaczynski & Trousse,

1994). Ces résultats sont alors ordonnés suivant une relation d'ordre associée (point d'ouverture CmpValueOrder). Enfin, ces mesures s'appuient sur différentes fonctions élémentaires de similarité et sur des fonctions d'agrégation (points d'ouverture ValueSimilarityFct et AggregationFct) qui doivent être définies suivant les connaissances du domaine, la sémantique des indices et leur domaine de valeurs.

2.3.2 Points d'ouverture pour l'indexation par situations comportementales

L'indexation par situations comportementales permet de construire un type particulier de systèmes de RàPC. Ainsi, les points d'ouverture précédemment identifiés sont, soit spécialisés, soit utilisés directement, soit enrichis par de nouveaux points d'ouverture spécifiques. Nous reprenons la structuration en axes de variabilité pour présenter ces points d'ouverture.

2.3.2.1 *Gestion du raisonnement*

Pour l'indexation par situations comportementales, la gestion du raisonnement repose sur les points d'ouverture déjà présentés. En effet, l'utilisation des principes de gestion concernant le contrôle du raisonnement s'appuie sur les points d'ouverture généraux (notamment Retrieve, Reuse, et Retain). La configuration et le lancement de la stratégie de recherche sont effectués avec le point d'ouverture Retrieve. Le résultat de la mise en correspondance entre les cas sources et le cas cible est récupéré en utilisant les points d'ouverture RetrieveResult et Reasoning. La phase de réutilisation peut alors opérer des actions d'adaptation des solutions à travers le point d'ouverture Reuse. Enfin, la phase d'apprentissage exécute une ou plusieurs actions parmi les quatre types d'actions proposés, avec le point d'ouverture Retain. Toutefois, la gestion raisonnement doit permettre la sélection et la création des nouveaux éléments introduits dans les autres axes (base d'enregistrements et base patrons de cas potentiels pour l'organisation de la mémoire, cf. §2.3.2.3). C'est pourquoi le point d'ouverture ReasonerFactory est remplacé par le point d'ouverture TimeExtendedReasonerFactory (cf. Figure III-19).

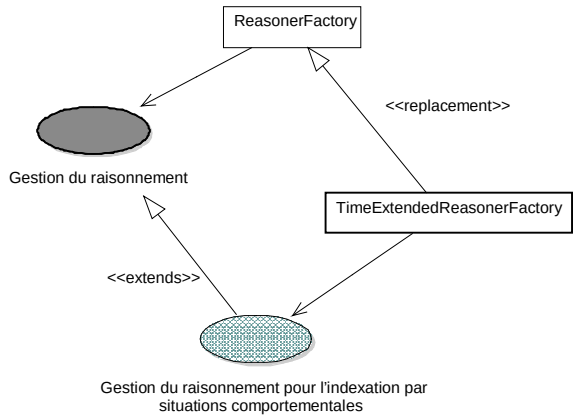


Figure III-19 : remplacement du point d'ouverture ReasonerFactory

2.3.2.2 *Représentation des cas*

Dans le cadre de notre modèle d'indexation, l'axe de variabilité de représentation des cas recouvre la représentation des enregistrements (données d'observation) et la représentation des cas proprement dite (cf. Figure III-20). Les enregistrements (point d'ouverture Record) sont composés d'un ensemble de chroniques et d'un contexte à définir suivant l'application considérée. Les chroniques utilisées peuvent être échantillonnées ou à événements, et leurs domaines structurels (entier, réel, chaîne de caractères ou autres) et temporels (jour, mois, année par exemple) sont également variables (point d'ouverture TimeSeries). Par implication, la gestion d'un instant de référence au sein d'un

enregistrement, qui sera utilisé pour exprimer une situation comportementale, doit être adaptée à cette représentation (point d'ouverture Cursor).

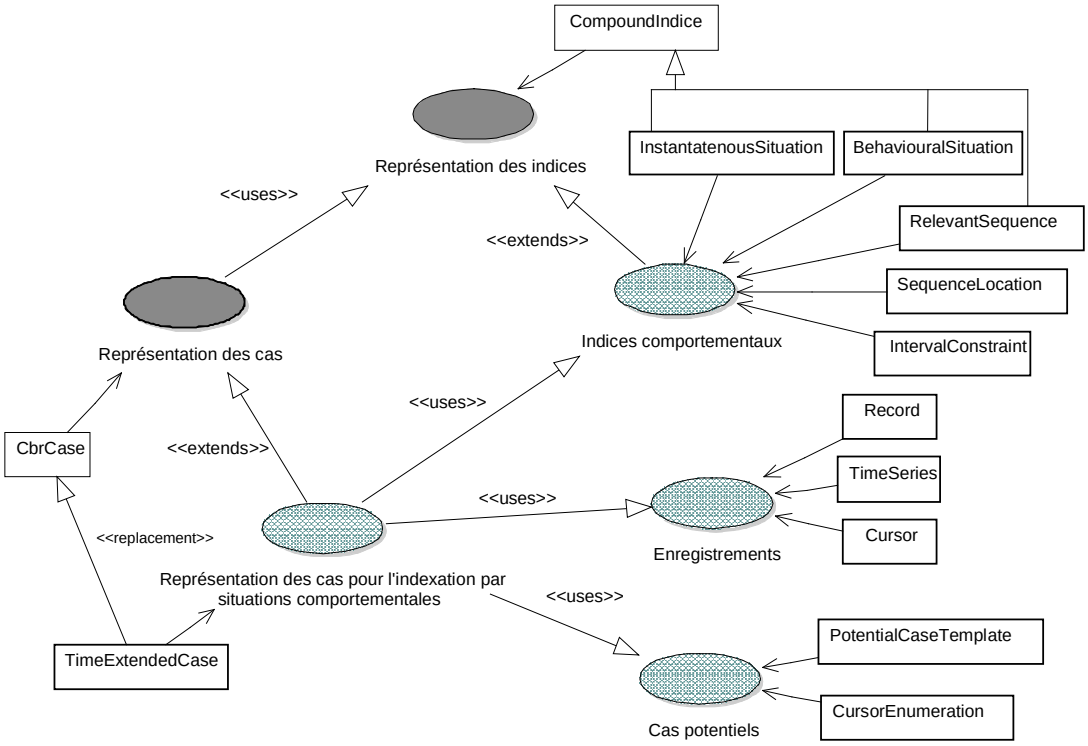


Figure III-20 : points d'ouverture pour la représentation des cas avec des indices comportementaux

Nous rappelons que la représentation des cas comprend la représentation des patrons de cas potentiels, et des cas concrets. Un patron de cas potentiels doit implanter deux opérations (point d'ouverture PotentialCaseTemplate) : le prédicat d'applicabilité ($applicable_P$) et la fonction d'instanciation ($nouveau_P$). De plus, pour une meilleure efficacité, l'algorithme de parcours des enregistrements, utilisé pour instancier un patron, doit être conçu en fonction du prédicat d'applicabilité et des propriétés spécifiques des chroniques (point d'ouverture CursorEnumeration).

Pour les patrons de cas potentiels ainsi que pour les cas concrets, il est nécessaire de représenter les parties « solution » et « problème » d'un cas. La représentation de la solution est spécifique à la tâche du système (point d'ouverture TimeExtendedCase). Le point d'ouverture général (CbrCase) a été remplacé pour permettre l'expression de cette solution, tout en intégrant par défaut des informations particulières : type de cas (concrets ou potentiels) et utilisation d'une situation comportementale.

La partie problème des cas s'appuie sur la représentation d'une situation comportementale qui comprend des indices instantanés (point d'ouverture InstantaneousSituation) et des indices comportementaux. L'expression des indices comportementaux spécialise la tâche générale de représentation des indices. La composante comportementale doit gérer les différentes interfaces fonctionnelles (consultation, analyse, sélection et modification). Ces interfaces doivent prendre en compte la structure particulière considérée pour fournir des opérations efficaces et adaptées (point d'ouverture BehaviouralSituation). La composante comportementale s'exprime en termes de comportements élémentaires et de contraintes temporelles. Les comportements élémentaires doivent être adaptés aux types de chroniques sous-jacentes (point d'ouverture RelevantSequence). Nous avons introduit un certain type de contraintes temporelles de base (relations d'Allen) et d'autres types peuvent êtres définis (contraintes numériques par exemple, d'où le point d'ouverture

IntervalConstraint). Les comportements élémentaires reposent sur des contraintes de localisation (point d'ouverture SequenceLocation) dont nous avons défini quatre types de base : localisation par position, par temps écoulé, par valeur et par référence.

2.3.2.3 *Organisation de la mémoire*

L'organisation de la mémoire repose en grande partie sur les points d'ouverture déjà spécifiés. La mise en œuvre des principes de gestion pour la phase de recherche nécessite une organisation particulière de l'indexation qui est réalisée avec les points d'ouverture existants (notamment Index et IndexResult). L'organisation des cas n'est pas non plus spécifique à l'indexation par situations comportementales. Par contre, l'organisation générale de la mémoire intègre deux nouvelles composantes (cf. Figure III-21) pour gérer et organiser l'ensemble des patrons (point d'ouverture TemplateBase) et des enregistrements (point d'ouverture RecordBase). Il en résulte que le point d'ouverture existant (Memory) gérant globalement la mémoire est remplacé (point d'ouverture TimeExtendedMemory).

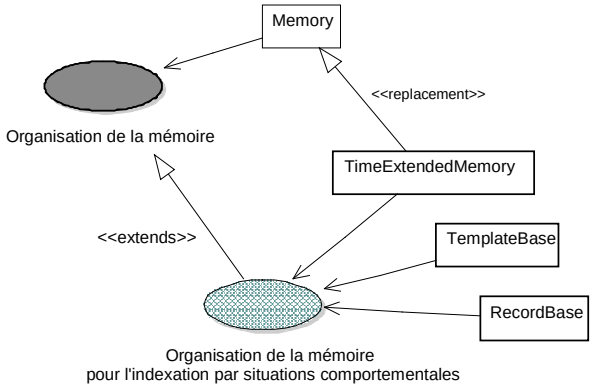


Figure III-21 : points d'ouverture pour l'organisation de la mémoire dans l'indexation par situations comportementales

3 Conclusion

Dans ce chapitre, nous avons mené l'état de l'art des outils logiciels existants pour le RàPC. Nous avons montré qu'ils ne présentaient pas une démarche d'ouverture systématique, ce qui les rend difficiles à enrichir et/ou à modifier. Pour dépasser ces limites, nous proposons un *nouveau type d'outils* pour le RàPC à travers la conception d'une plate-forme à objets. Cette approche repose sur la définition d'une architecture abstraite modélisant les concepts du RàPC et de l'indexation par situations comportementales. Cette architecture intègre des points d'ouverture qui peuvent être configurés par spécialisation ou par instanciation.

Nous concrétisons alors cette approche par la proposition d'un ensemble de 38 points d'ouverture qui est structuré en deux niveaux de spécificité (RàPC général et indexation par situations comportementales) et en trois axes de variabilité (cf. Tableau III-1) : gestion du raisonnement, représentation des cas et organisation de la mémoire. Au sein de ces axes, nous avons structuré ces points d'ouverture en fonction des tâches de réalisation d'un système de RàPC. Les points d'ouverture nécessaires à l'indexation par situations comportementales apparaissent alors comme une extension des points d'ouverture de base. Ainsi, nous définissons un cadre cohérent pour la gestion de la spécialisation et de l'intégration du modèle d'indexation dans un système de RàPC. Enfin, l'intérêt de

cette analyse, nécessaire pour la conception d'une plate-forme, dépasse ce cadre précis et peut permettre la mise en place de points d'ouverture dans d'autres réalisations de systèmes de RàPC.

Dans le chapitre suivant, nous montrons comment ces points d'ouverture sont intégrés à la conception des modèles à objets ouverts de la plate-forme CBR*Tools. La structuration proposée des points d'ouverture et sa représentation graphique sous forme de cas d'utilisation permettent alors une meilleure compréhension de la plate-forme et fournissent un guide au concepteur d'un système de RàPC désirant configurer certains points d'ouverture.

Chapitre IV : patrons de conception et modèles à objets ouverts pour le RàPC

Résumé

En suivant les deux niveaux de spécificité et les trois axes de variabilité retenus, nous détaillons les modèles à objets permettant de lier les points d'ouverture de notre plate-forme au sein d'une architecture abstraite. Notre démarche originale dans le cadre du RàPC tient principalement dans l'explication systématique de notre modélisation en termes de *patrons de conception*. Un patron de conception définit une modélisation typique répondant à un problème récurrent dans la conception de logiciels à objets. Les patrons permettent d'anticiper des problèmes d'ouverture et facilitent la compréhension des décisions de conception. Pour la gestion du raisonnement, nous proposons l'assemblage de composants gérant chacune des phases (recherche, réutilisation, révision et apprentissage). Chaque phase est configurée par la spécialisation d'un schéma générique à sélectionner. Pour la représentation des cas, nous permettons une structuration des indices. Enfin, pour l'organisation de la mémoire, nous permettons d'une part la structuration de la base de cas, et d'autre part la structuration des index. De plus, nous avons défini un cadre général et extensible pour la représentation des mesures de similarité. La représentation de l'indexation par situations comportementales vient enrichir la modélisation de base avec des objets spécifiques. La modélisation obtenue, justifiée en termes de patrons, est ainsi réutilisable au niveau de la conception de systèmes de RàPC. Cette modélisation a également mené à la réalisation concrète de la plate-forme CBR*Tools programmée dans le langage à objets Java. Cet outil permet alors la spécialisation et la capitalisation d'objets-métier notamment sous les formes d'index, de fonctions de similarité et de schémas de phase de raisonnement. Nous donnons enfin une estimation *a priori* de la complexité d'utilisation de CBR*Tools (effort minimum et dépendances systématiques).

Dans le chapitre précèdent, nous avons introduit notre approche originale proposant la conception d'une plate-forme à objets pour le RàPC et plus particulièrement pour l'indexation par situations comportementales. Nous avons également introduit les points d'ouverture nécessaires à une telle approche. Nous présentons alors dans ce chapitre les modèles à objets qui explicitent l'architecture abstraite de la plate-forme et la conception de chaque point d'ouverture. Notre démarche de conception est centrée sur l'utilisation de patrons de conception (*design pattern*) qui permettent de réutiliser des décisions typiques de conception pour obtenir l'ouverture désirée.

Dans les première et deuxième parties, nous détaillons la modélisation proposée respectivement, pour le cycle du RàPC de manière générale, et pour l'indexation par situations comportementales. Nous expliquons et argumentons l'utilisation des patrons en suivant les trois axes de variabilité retenus : gestion du raisonnement, représentation des cas et organisation de la mémoire. Chaque axe se décline alors suivant différents modèles à objets pour réaliser les points d'ouverture explicités précédemment. Nous nous concentrons plus particulièrement sur la représentation des cas et l'organisation de la mémoire qui sont essentielles pour notre modèle d'indexation. Pour la gestion du raisonnement, nous nous limitons à définir un cadre cohérent permettant d'intégrer les composants de notre modèle d'indexation dans un système complet de RàPC. Les diagrammes présentant notre modélisation respectent la notation UML (UML, 1997 ; Muller, 1997) dont les éléments de base sont précisés dans l'annexe A. Les patrons utilisés sont plus amplement décrits dans l'annexe B. Enfin, dans la troisième partie, nous décrivons la réalisation de cette plate-forme qui est alors nommée CBR*Tools. Un exemple simple de l'utilisation de cet outil est donné dans l'annexe C. Le chapitre suivant présente notre système Broadway dont la réalisation repose sur CBR*Tools.

1 Modèles à objets ouverts pour le RàPC

Nous présentons tout d'abord l'utilisation des patrons de conception pour la modélisation du cycle du raisonnement et des concepts généraux au RàPC suivant les trois axes de variabilité.

1.1 Gestion du raisonnement

Le raisonnement à partir de cas est généralement décomposé en quatre phases : recherche, réutilisation, révision et apprentissage. Nous montrons, dans cette section, comment sont organisés les 11 points d'ouverture liés à ce cycle (cf. chapitre III, §2.3.1.1) : contrôle du raisonnement, phases de raisonnement, résultat des phases, état du raisonnement et création des objets nécessaires au raisonnement. Nous introduisons tout d'abord notre modélisation par délégation des phases de raisonnement avec le patron *Stratégie*. Un raisonnement est alors construit par assemblage. Puis, pour la réalisation de chaque phase, nous proposons la définition d'une bibliothèque de *schémas de phase.* Chaque schéma peut être spécialisé pour réaliser une phase particulière. L'ouverture de cette modélisation repose notamment sur les patrons de conception *Stratégie*, *Façade*, *Fabrique abstraite*, *Patron de méthode* et *Chaîne de responsabilité* (Gamma *et al.,* 1995).

1.1.1 Patrons de conception et représentation du cycle du raisonnement

Pour la gestion du cycle du raisonnement, nous détaillons tout d'abord le mécanisme de contrôle qui permet la coordination des phases. Nous donnons alors l'exemple du protocole standard de

raisonnement. Puis, nous proposons des mécanismes permettant d'expliciter et de vérifier les contraintes syntaxiques et sémantiques garantissant la cohérence du raisonnement.

1.1.1.1 *Contrôle du raisonnement*

Nous proposons de déléguer l'exécution et le contrôle de chaque phase du raisonnement à un objet différent en appliquant quatre fois le patron de conception *Stratégie*. Ce patron permet la définition d'une famille d'algorithmes pour chaque phase du raisonnement, parmi lesquels il est possible de sélectionner celui à utiliser. Cette modélisation permet la réalisation des quatre points d'ouverture importants pour le raisonnement dont les classes de base sont Retrieve, Reuse, Revise et Retain. Chacune de ces classes définit une interface abstraite pour les phases du raisonnement et différentes implantations peuvent être réalisées. Le raisonnement est alors construit par assemblage des différents algorithmes pour chaque phase. Le contrôleur du raisonnement (classe Reasoner) est alors la façade du système de raisonnement à partir de cas (patron de conception *Façade*) et définit donc les points d'entrée (cf. Figure IV-1). Il permet de lancer un raisonnement étant donné un cas cible et donne l'accès à la mémoire. De plus, il constitue le point d'ouverture par lequel la gestion globale du raisonnement peut être spécialisée.

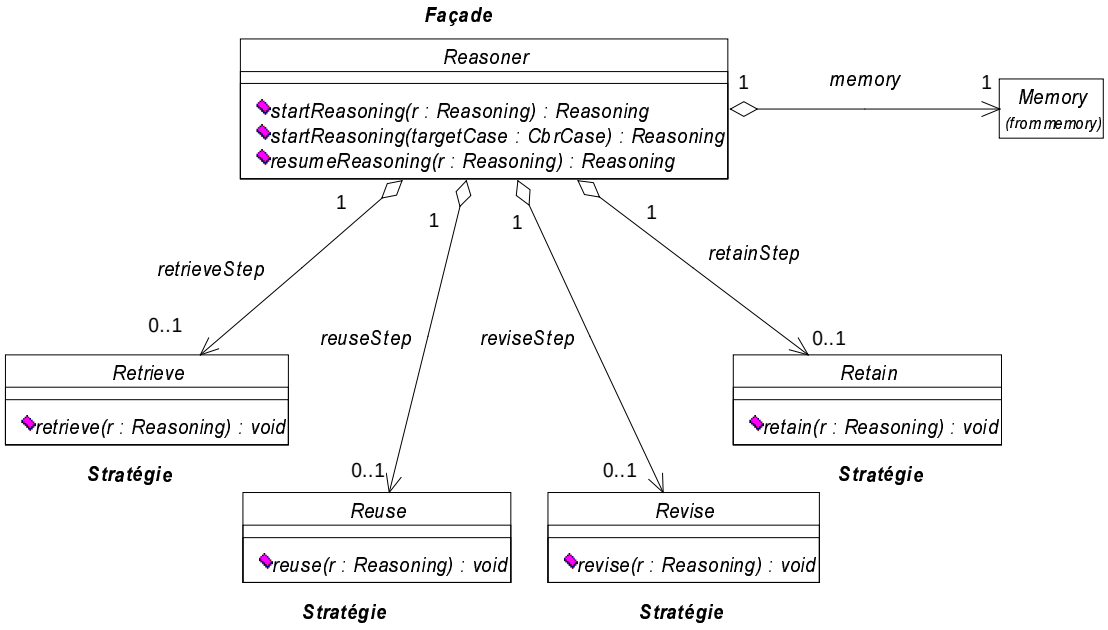


Figure IV-1 : représentation du contrôleur de raisonnement

Dans le raisonnement à partir de cas, chaque phase possède des paramètres spécifiques (nombre maximum de cas à rechercher par exemple) et opère sur les résultats issus des phases de raisonnement déjà exécutées. Par exemple, la phase de réutilisation utilise les résultats de la recherche, et la phase d'apprentissage peut utiliser les résultats des autres phases pour mettre à jour l'indexation de cas. Nous proposons alors de réunir dans un objet (point d'ouverture de classe Reasoning) l'ensemble des données liées au déroulement du raisonnement. En ce sens, nous suivons une des règles de conception donnée dans (Johnson & Foote, 1988) pour réduire le nombre de paramètres d'une méthode en créant un objet regroupant ces paramètres. Cet objet est alors modifié par chaque étape pour refléter l'avancement du raisonnement (traces du raisonnement), c'est pourquoi il est composé d'objets donnant les résultats de chaque phase (points d'ouverture de classes respectives RetrieveResult, ReuseResult, ReviseResult et RetainResult, cf. Figure IV-2). L'utilisation de cet objet présente

également l'avantage de pouvoir sauvegarder l'état d'un raisonnement qui peut alors être suspendu puis repris. De plus, cette modélisation permet d'analyser le raisonnement après son exécution.

Cette modélisation est apparentée à une architecture qui est souvent utilisée dans les systèmes multi-agents (Haton *et al.*, 1991) sous la forme d'un tableau noir (*blackboard*). Le modèle de tableau noir a été également présenté sous la forme d'un patron de conception (Bushmann *et al.*, 1996, pages 71–95). L'architecture typique à base de tableau noir est composée d'un ensemble d'agents appelés *sources de connaissances*, du tableau noir proprement dit et d'un module de contrôle. Les sources de connaissances possèdent une certaine compétence et ont une vue réduite sur l'état de la solution en cours d'élaboration dans le tableau noir. Le tableau noir est une sorte de structure de données qui permet de mémoriser l'état de la solution résultant de l'application des différentes sources de connaissances. Le module de contrôle supervise le déclenchement des différentes sources de connaissances. Son rôle est d'identifier les sources de connaissances qui peuvent s'appliquer, d'en choisir une et de l'exécuter. Toutefois, ce module peut être absent et dans ce cas, les sources de connaissance sont autonomes et réagissent aux modifications des données auxquelles elles ont accès dans le tableau noir. Dans notre modélisation, les sources de connaissances sont les phases du raisonnement, le tableau noir est représenté par l'objet sauvegardant l'état du raisonnement et le module de contrôle (obligatoirement présent) correspond au contrôleur du raisonnement. Ainsi, nous reprenons les avantages reconnus de ce type d'architecture (modularité et évolutivité), tout en faisant apparaître des concepts spécifiques au RàPC.

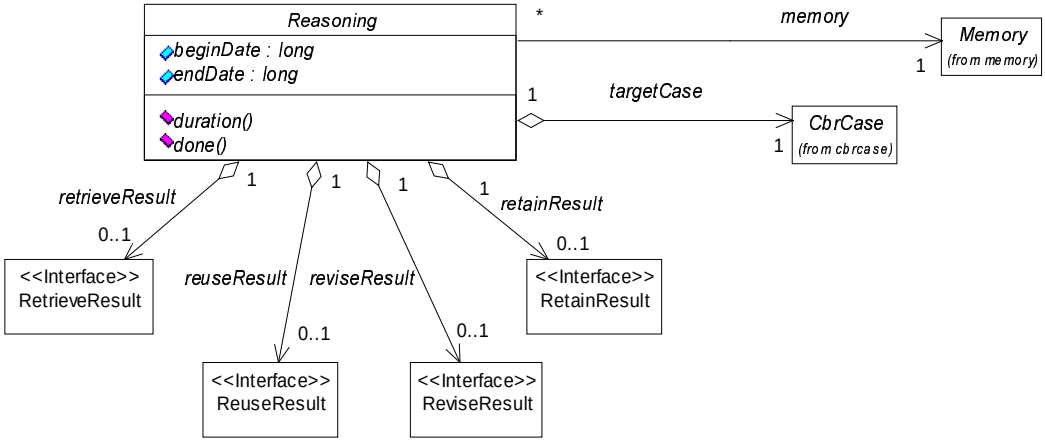


Figure IV-2 : représentation de l'état d'un raisonnement

A titre d'exemple, nous décrivons le protocole standard de raisonnement qui est sélectionné par défaut dans la plate-forme. Ce protocole est implanté dans la classe concrète SequentialReasoner qui spécialise la classe Reasoner (cf. Figure IV-3). Lors de l'appel d'un raisonnement, le contrôleur commence par initialiser le raisonnement (utilisation de la fabrique, cf. §1.1.1.2.2), puis appelle successivement les phases de recherche, de réutilisation, de révision et d'apprentissage. Le raisonnement peut être stoppé, soit sur l'initiative d'une des phases, soit en lançant le raisonnement avec un paramètre indiquant la phase d'arrêt. De plus, aucune des phases n'est obligatoire et les phases non définies ne sont pas exécutées.

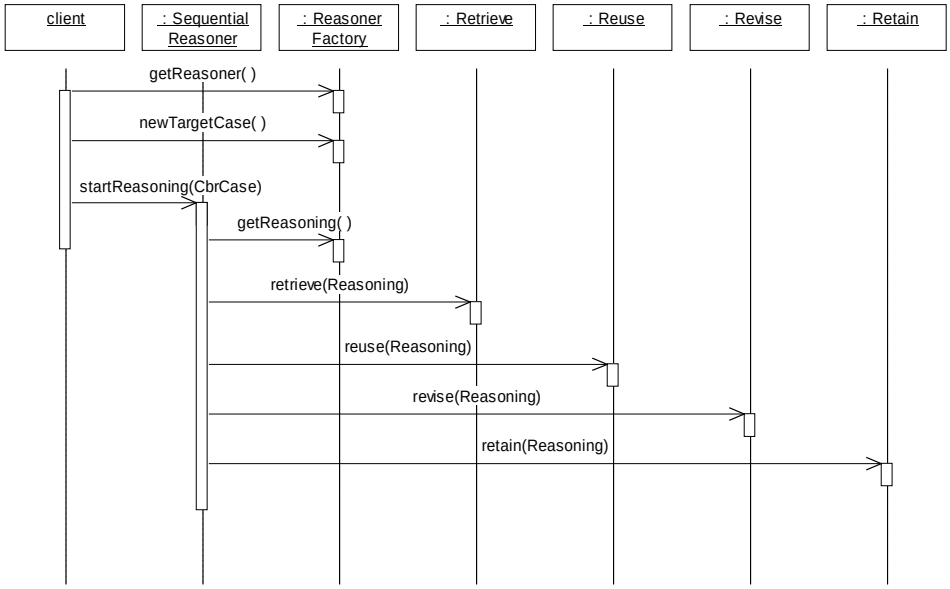


Figure IV-3 : protocole standard de raisonnement

1.1.1.2 *Cohérence de l'assemblage des phases*

Grâce à l'utilisation du patron de conception *Stratégie* pour les phases du raisonnement et la centralisation de leur interaction dans un objet sauvegardant l'état du raisonnement, nous permettons la réutilisation par assemblage de différentes stratégies de recherche, de réutilisation, de révision et d'apprentissage. Toutefois, cet assemblage ne peut être effectué aveuglément. Par exemple, si des hypothèses émises par une phase de réutilisation ne sont pas satisfaites par les résultats effectifs de la phase de recherche, le raisonnement est alors impossible. Il s'agit donc d'assurer la *compatibilité* entre les phases. Dans ce but, nous avons défini les notions de profils d'entrée et de sortie, et nous utilisons le patron *Fabrique abstraite* pour centraliser la création des objets nécessaires au raisonnement.

1.1.1.2.1 Profils d'entrée et de sortie

Nous proposons d'associer à chaque phase un *profil d'entrée* et un *profil de sortie*. Un profil d'entrée pose des contraintes sur le contenu des résultats des phases qui lui sont utiles et un profil de sortie définit les propriétés des résultats produits par une phase. Pratiquement, un profil est constitué par un ensemble d'interfaces de manipulation de données. Ces interfaces sont organisées de manière hiérarchique (cf. Figure IV-4). Nous avons particulièrement étudié les interfaces suivantes, nécessaires pour décrire les types des résultats de la phase de recherche :

- ensemble de cas (interface RetrieveResult),
- singleton (interface SingleCaseRetrieveResult),
- association d'un degré d'utilité à chaque cas retrouvé (interface UsefulnessRetrieveResult),
- association de la similarité sur chaque indice pour les cas retrouvés (interface DetailedSimRetrieveResult).

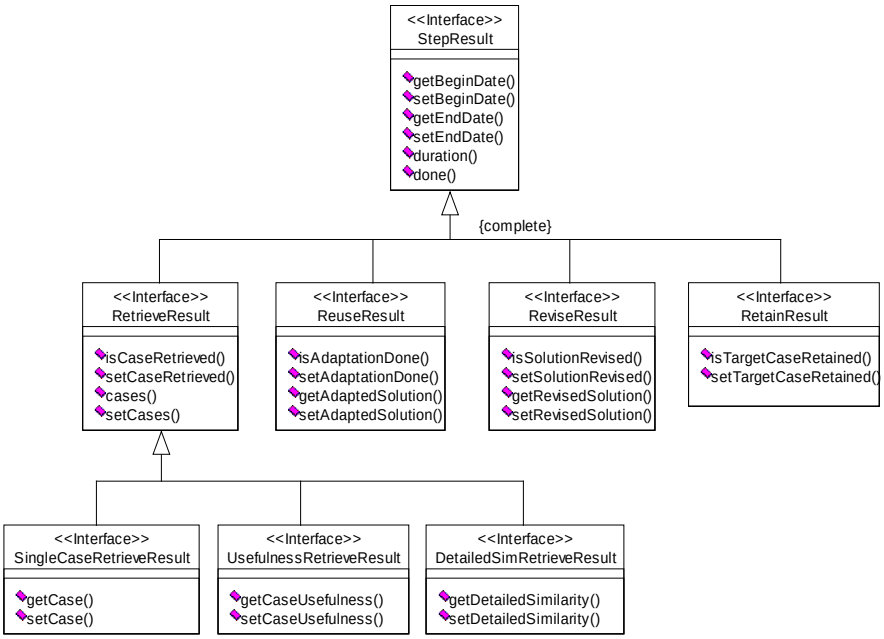


Figure IV-4 : interfaces de base pour les résultats des phases de raisonnement

Ainsi, les phases d'un raisonnement pourront être assemblées si et seulement si, pour chaque phase, l'union des profils de sortie des autres phases vérifie son profil d'entrée. L'union de deux profils est un profil, et un profil P vérifie un profil Q, si et seulement si, il existe dans P pour chaque interface de Q une interface identique ou plus spécifique. Cette relation, notée vérifie, s'écrit alors de la façon suivante :

Profil : {Interface}

vérifie : Profil, Profil $\rightarrow$ Booléen

{ P vérifie Q $\Leftrightarrow \forall$ i $\in$ Q, $\exists$ j $\in$ P tel que i=j ou j plus spécifique que i }

	Assemblage A		*Assemblage B*	
Profils de recherche	entrée : sortie :	{} {SingleCaseRetrieveResult}	entrée : sortie :	{} {DetailedSimRetrieveResult}
Profils de réutilisation	entrée : sortie :	{RetrieveResult} {ReuseResult}	entrée : sortie :	{UsefulnessRetrieveResult} {ReuseResult}
Profils de révision	entrée : sortie :	{ReuseResult} {ReviseResult}	entrée : sortie :	{} {}
Profils d'apprentissage	entrée : sortie :	{ReuseResult, ReviseResult} {RetainResult}	entrée : sortie :	{} {RetainResult}
Contrôle de l'assemblage	*compatible*		*incompatible*	

Tableau IV-1 : exemples de contrôles d'assemblages

Cette modélisation présente trois avantages importants. Premièrement, la hiérarchie d'interfaces est facilement extensible et permet la prise en compte de nouveaux types de résultats sans être liée à l'ordre d'exécution des phases. Deuxièmement, le contrôle de l'assemblage peut être effectué dès la conception, les profils jouant le rôle d'une documentation associée aux phases. Le contrôle peut

également être effectué lors de l'assemblage des objets[39], et nous évitons ainsi de retarder le contrôle jusqu'à l'exécution. Enfin, l'utilisation concrète de ces profils repose sur la définition d'une classe héritant des interfaces nécessaires pour chaque phase. Cette classe d'objets peut permettre d'assembler des phases *a priori* incompatibles en encapsulant des opérations de conversion (patron *Adaptateur*). Le couplage direct entre les phases est ainsi minimisé pour une meilleure réutilisabilité.

Le Tableau IV-1 donne l'exemple de deux assemblages : les phases de l'assemblage *A* sont compatibles contrairement à celles de l'assemblage *B*. En effet, dans l'assemblage *B*, la phase de recherche retourne la similarité détaillée de chaque cas retrouvé, mais ne donne pas de facteur d'utilité nécessaire à la phase de réutilisation. La modélisation proposée consiste à assembler ces deux phases en utilisant un objet de résultats réalisant les deux interfaces mises en cause (UsefulnessRetrieveResult et DetailedSimRetrieveResult). L'héritage multiple d'interfaces est alors une solution élégante pour réaliser le patron de conception *Adaptateur* (cf. Figure IV-5). Cet objet encapsule alors le calcul du facteur d'utilité de chaque cas à partir de la similarité détaillée.

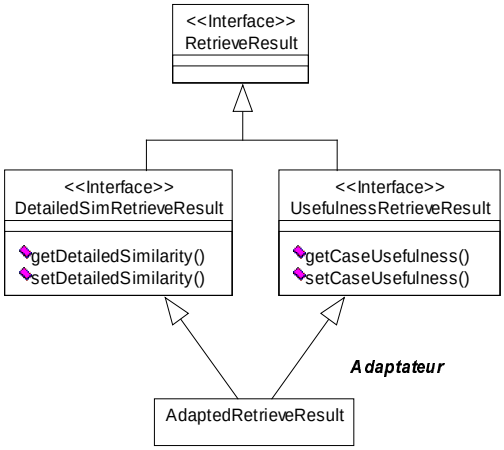


Figure IV-5 : adaptation d'interfaces pour l'assemblage

1.1.1.2.2 Gestion centralisée de la création des objets

L'utilisation des profils garantit la cohérence syntaxique de l'assemblage mais ne garantit pas la compatibilité sémantique de l'ensemble des objets intervenant durant un raisonnement. Cette compatibilité est difficile à vérifier automatiquement dans la mesure où elle repose principalement sur les connaissances d'un domaine : il serait coûteux de représenter ses propriétés pour pouvoir gérer leur cohérence au niveau de l'assemblage. Nous laissons donc la responsabilité du choix des objets à assembler au concepteur, mais nous offrons un modèle créateur permettant la centralisation de la gestion de l'assemblage. Dans ce but, nous utilisons le patron de conception *Fabrique abstraite* et nous réunissons dans une classe (ReasonerFactory) les méthodes de création des objets participant à cet assemblage (cf. Figure IV-6) : contrôleur de raisonnement, phases de recherche, objet de raisonnement, mémoire, cas cible. La centralisation des méthodes de création renforce la notion de famille d'objets dont la cohérence doit être assurée. Plusieurs fabriques concrètes peuvent alors être définies. Cette modélisation facilite également la substitution d'un type d'objet par un autre sachant qu'à chaque fois qu'un objet important doit être créé, il faudra s'adresser à la fabrique. Cette substitution est simplement réalisée par la dérivation d'une fabrique concrète existante et par la redéfinition d'une méthode de création.

[39] Dans CBR*Tools, les profils sont pour l'instant uniquement utilisés comme sources de documentation.

En reprenant l'exemple de la section précédente (cf. Figure IV-5), la réalisation de l'assemblage entraîne la spécialisation de la fabrique pour faire appel à la création des objets gérant la phase de recherche (méthode newRetrieveStep), la phase de réutilisation (méthode newReuseStep) et le résultat de la recherche (méthode newRetrieveResult).

De plus, lors de la création d'un contrôleur de raisonnement (méthode getReasoner, cf. Figure IV-6), l'objet de fabrique est automatiquement affecté comme composant du contrôleur. Le contrôleur joue donc pleinement son rôle de façade en donnant à ses clients l'accès à la fabrique. Ces derniers pourront ainsi créer les cas cibles compatibles avec le système de raisonnement construit. Le contrôleur lui-même utilise la fabrique pour créer, à chaque raisonnement, l'objet d'état du raisonnement dont la classe de base est Reasoning. Le système peut alors assurer que les objets utilisés dans le raisonnement sont issus de la famille définie par le concepteur dans la fabrique.

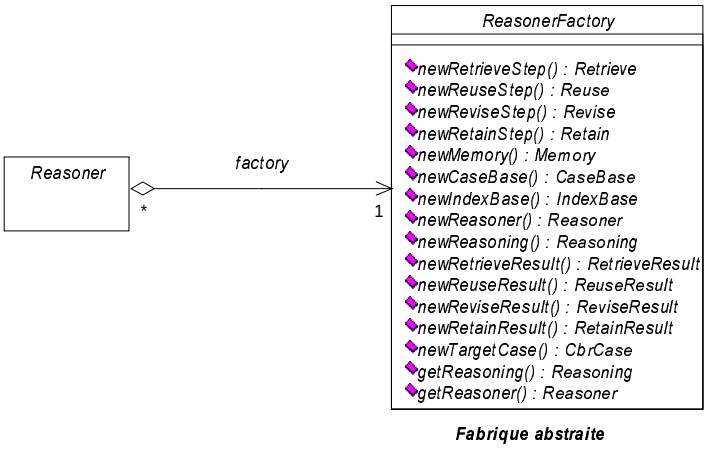


Figure IV-6 : la fabrique comme composant du contrôleur du raisonnement

1.1.2 Schémas de base pour les phases du raisonnement

Suivant notre modèle, le contrôleur du raisonnement délègue l'exécution de chacune des phases à un de ses composants. Ainsi, des objets indépendants contrôlent à leur tour les phases de recherche, de réutilisation, de révision et d'apprentissage. Afin de prendre en compte la diversité des techniques employées pour réaliser ces différentes phases, nous proposons une approche pragmatique basée sur la description de *schémas*. Un schéma est une solution typique pour la réalisation d'une phase du raisonnement adaptée à un cadre bien défini. Chaque schéma peut être spécialisé pour une application concrète, puis assemblé à d'autres pour former un raisonnement complet. Un schéma comprend un ensemble de classes abstraites et de composants concrets ; ainsi un schéma est semblable à une plate-forme à objets de très petite taille.

Nous donnons, à titre d'exemple, les schémas modélisés et implantés dans CBR*Tools pour les phases de recherche, de réutilisation et d'apprentissage. Ces schémas reposent tous sur une classe d'objets définissant un protocole de contrôle du schéma suivant la patron de conception *Patron de méthode*. Ce protocole se décompose en quatre étapes au minimum : initialisation, opérations spécifiques, mise à jour des résultats de la phase en accord avec le profil de sortie et finalisation. L'utilisation de ces schémas se fait principalement par spécialisation du protocole. Les étapes d'initialisation et de finalisation sont utilisées comme des démons pouvant être spécialisés pour déclencher des actions de propagation (pour l'interface utilisateur par exemple). L'étape de mise à jour des résultats peut également être spécialisée pour formuler les résultats suivant un profil de sortie requis. Cette modélisation permet ainsi d'adapter finement les étapes d'un schéma à une utilisation spécifique.

1.1.2.1 *Schéma de configuration pour la recherche*

La phase de recherche s'appuie sur les index qui organisent les cas suivant leur indices pour retrouver les cas les plus utiles étant donné un cas cible. Plusieurs index de haut niveau, formant des stratégies de recherche différentes, peuvent coexister dans CBR*Tools (cf. §1.3.3). Lors de la phase de recherche, il s'agit principalement de sélectionner et de configurer la stratégie à adopter. Le schéma que nous proposons pour la recherche permet de contrôler et de lancer la stratégie retenue. Le protocole de ce schéma est défini dans la classe BasicRetrievePattern et les paramètres de configuration d'une stratégie sont donnés dans un objet de classe GlobalIndexParams (cf. Figure IV-7). Ces paramètres peuvent être clonés suivant le patron *Prototype*. Cette modélisation présente donc l'avantage de permettre la modification dynamique des paramètres par défaut et la constitution de plusieurs configurations types.

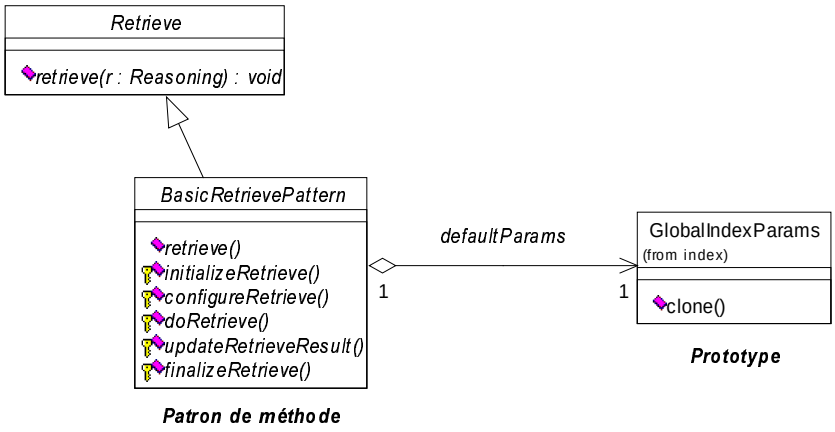


Figure IV-7 : classes d'objets du schéma de base pour la phase de recherche

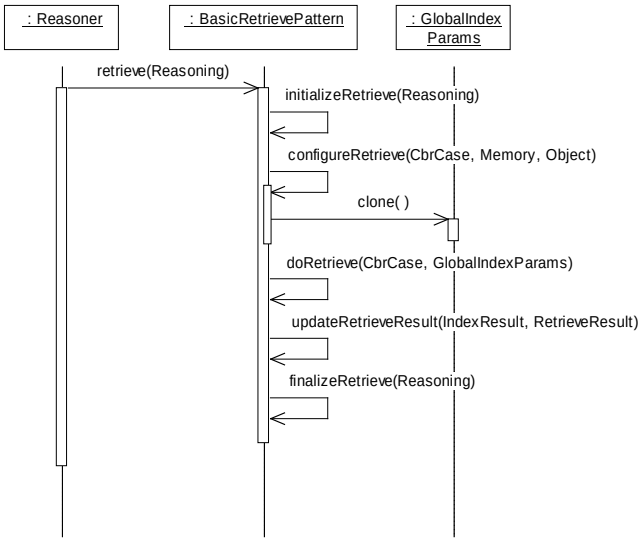


Figure IV-8 : protocole de configuration et d'exécution de la recherche

Le protocole de recherche repose alors sur trois étapes principales de contrôle qui sont exécutées en séquence (cf. Figure IV-8) :

1. *configuration* (méthode configureRetrieve) : clone les paramètres par défaut et les modifie au besoin,
2. *recherche* (méthode doRetrieve) : lance la recherche avec la stratégie retenue,
3. *mise à jour des résultats* (méthode updateRetrieveResult) : transforme les résultats de la stratégie de recherche (de type IndexResult) en résultats de la phase de recherche (de type RetrieveResult).

Le profil d'entrée de ce schéma est vide et celui de sortie est à définir par l'utilisateur qui doit obligatoirement spécialiser l'étape de mise à jour des résultats suivant le profil désiré.

1.1.2.2 *Deux schémas pour la réutilisation*

Pour la phase de réutilisation, nous proposons dans cette section deux schémas : le premier permet une adaptation basée sur l'analyse des indices du meilleur cas retrouvé et le deuxième permet de réaliser une adaptation des solutions proposées par un ensemble de cas. Nous donnerons également un troisième schéma de réutilisation issu de notre application Broadway (cf. chapitre V, §2.2.1).

1.1.2.2.1 Schéma d'adaptation d'un cas par analyse des indices

Le premier schéma permet de réaliser une adaptation similaire à celle du système REMIND (cf. chapitre III, §1.1.1) où la solution du cas cible est calculée en prenant en compte les différences entre les indices du cas cible et ceux du cas source. Par exemple, nous pouvons grâce à ce schéma représenter la formule d'adaptation utilisée dans REMIND qui permet d'évaluer la valeur d'une maison en fonction de la valeur s_{valeur} d'une autre maison (cas source) et de l'indice donnant le nombre de chambres :

$$c_{valeur} = s_{valeur} + 10000(c_{chambres} - s_{chambres})$$

La Figure IV-9 présente le diagramme de classes participant à ce schéma : la classe IndiceAnalysisReusePattern définit le protocole qui délègue l'analyse des indices et le calcul de la solution à un objet de classe IndiceAnalysisAdpatationFormula (patron *Stratégie*). La modélisation proposée est flexible puisqu'un ensemble de formules peut être défini, tout en permettant le changement dynamique de la formule utilisée.

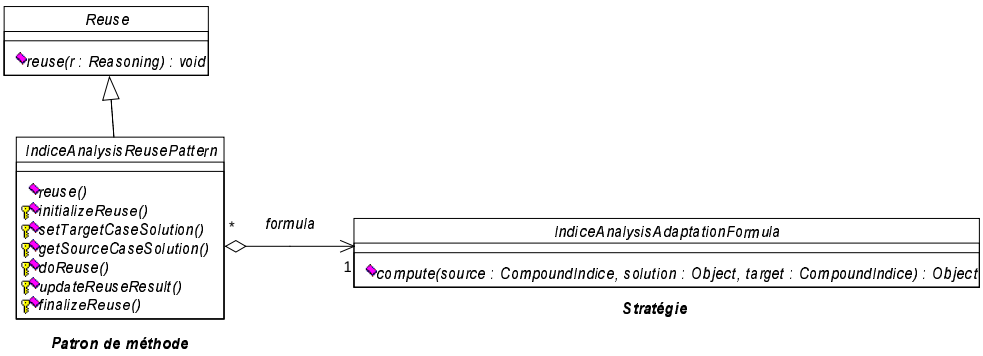


Figure IV-9 : schéma d'adaptation d'un cas par analyse des indices

Le protocole comprend deux étapes importantes (cf. Figure IV-10) :

1. *Calcul de la solution* (méthode doResuse) : la solution du cas source est récupérée (méthodes getSourceCaseSolution) et transmise à la formule d'analyse avec ses indices et les indices du cas cible. Cette formule retourne alors la solution qui est affectée au cas

cible (méthode setTargetCaseSolution). Les méthodes d'accès à la solution du cas doivent obligatoirement être définies par l'utilisateur suivant le type de cas considéré pour une meilleure encapsulation de sa structure.

2. *Mise à jour des résultats* (méthode updateReuseResult).

Les profils par défaut d'entrée et de sortie de ce schéma sont respectivement {SingleCaseRetrieveResult} et {ReuseResult}.

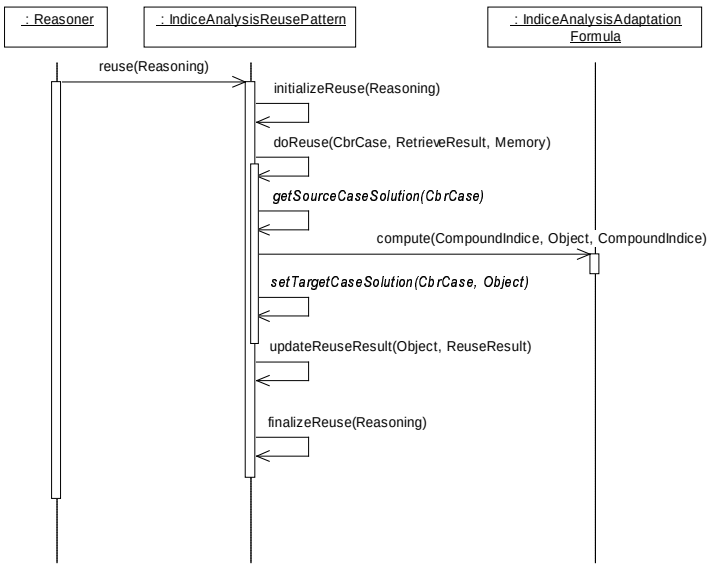


Figure IV-10 : protocole de réutilisation par analyse d'indices

1.1.2.2.2 Schéma d'adaptation de plusieurs cas par formule

L'objectif du deuxième schéma est de calculer la solution du cas cible en fonction des solutions proposées par un ensemble de cas sources retrouvés et de leur degré d'utilité. Ce type d'adaptation peut par exemple être utilisé pour la conduite de la nutrition des plantes. Dans notre approche (cf. chapitre II, §2.3), chaque cas retrouvé donne en effet comme solution la concentration de l'irrigation qui a été utilisée. A la suite de la phase de recherche qui a identifié un ensemble de cas sources, la solution peut être calculée en faisant une moyenne (ou tout autre formule) des solutions passées pondérées par les degrés d'utilité des cas sources associés (similarités des cas sources avec le cas cible courant).

La Figure IV-11 donne le diagramme de classes participant à ce schéma : la classe CaseSetAdaptationFormula définit le protocole qui délègue, en suivant les mêmes étapes que le schéma précédant, le calcul de la formule d'adaptation à un objet de classe CaseSetAdpatationFormula (patron *Stratégie*). Ce schéma permet donc de capitaliser différentes formules concrètes dont deux sont données en exemple : moyenne des solutions passées pondérées par leur degré d'utilité (classe MeanAdaptationFormula) et sélection de la solution présentant la meilleure utilité (classe BestAdaptationFormula).

Le profil par défaut d'entrée de ce schéma est donné par l'ensemble d'interfaces {UsefulnessRetrieveResult} alors que le profil de sortie est le singleton {ReuseResult}.

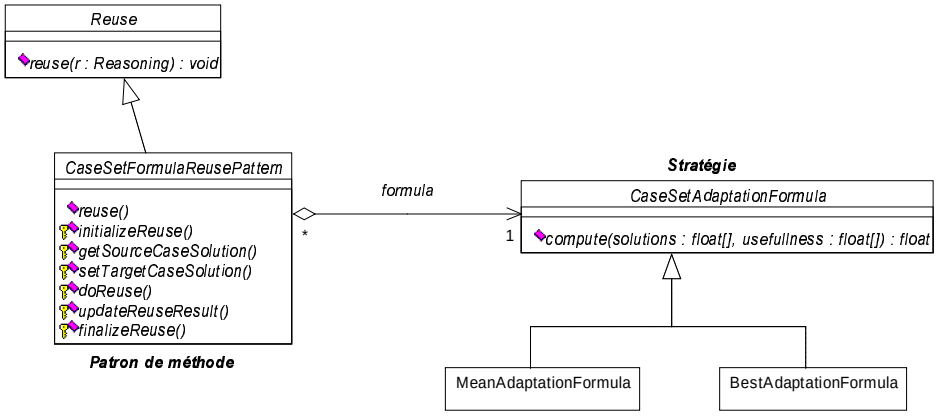


Figure IV-11 : schéma de réutilisation de plusieurs cas par formule

1.1.2.3 *Schéma de décision de l'ajout d'un cas pour l'apprentissage*

Dans le cadre du raisonnement à partir de cas, plusieurs types d'actions peuvent être effectués lors de l'apprentissage dont notamment la réorganisation générale de la mémoire avec modification de l'indexation, ou l'ajout et l'indexation du cas cible courant dans la mémoire. Nous proposons un schéma d'apprentissage permettant de simplifier la gestion de ce dernier type d'actions en appliquant le patron de conception *chaîne de responsabilité*. De manière similaire à la conception des schémas pour les autres phases (cf. Figure IV-12), ce schéma comprend un contrôleur de protocole (classe ResponsibilityChainRetainPattern) qui délègue la prise de décision de l'ajout d'un cas et de son indexation à un ensemble d'analyseurs (classe RetainAnalyser) organisés suivant une chaîne de responsabilité. L'intérêt de cette modélisation réside dans sa capacité à pouvoir prendre en compte de nouveaux types d'analyseurs tout en permettant leur combinaison.

Chaque analyseur étudie une caractéristique du cas cible ou de l'état du système de raisonnement pour décider s'il faut ou non ajouter le cas cible courant. Nous proposons trois analyseurs de base :

- un analyseur qui prend toujours la même décision, quel que soit le cas cible courant (classe ConstantRetainAnalyser),
- un analyseur vérifiant que la taille de la base de cas ne dépasse pas un certain seuil (classe CaseBaseSizeRetainAnalyser),
- un analyseur qui effectue une recherche dans la mémoire pour vérifier si un cas source existant n'est pas déjà trop similaire au cas cible courant (classe SimilarityRetainAnalyser).

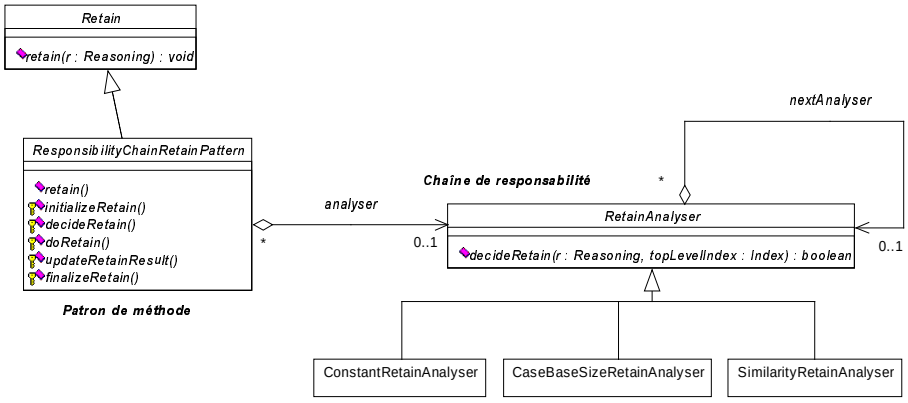


Figure IV-12 : schéma d'apprentissage par chaîne de responsabilité

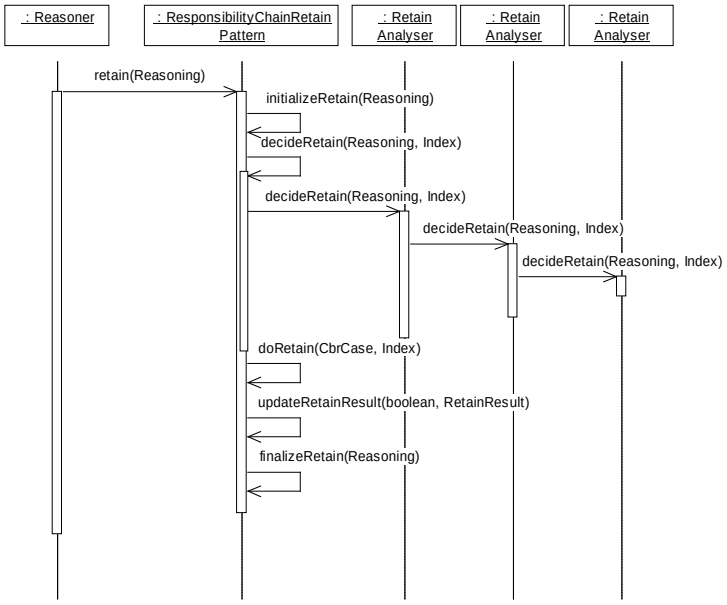


Figure IV-13 : protocole d'apprentissage par chaîne de responsabilité

Le protocole liant ces objets est donné dans la Figure IV-13, et repose sur trois étapes principales dont les deux premières sont exécutées pour chaque stratégie d'indexation définie dans le système de RàPC :

1. *Décision de l'ajout du cas* (méthode decideRetain) : la chaîne de responsabilité est parcourue. Au sein de cette chaîne, chaque analyseur a la possibilité, soit de passer la main à son successeur s'il est d'accord avec l'ajout du cas ou s'il ne sait pas décider, soit d'arrêter l'analyse en prenant une décision définitive.
2. *Ajout effectif du cas* (méthode doRetain) : si l'ajout a été décidé, il est alors réalisé.
3. *Mise à jour des résultats* (méthode updateRetainResult).

Le profil d'entrée par défaut de ce schéma est vide, et le profil de sortie est le singleton {RetainResult}.

1.2 Représentation des cas

Pour la représentation des cas, nous montrons la réalisation des quatre points d'ouverture correspondants (cf. chapitre III, §2.3.1.2) : tout d'abord la représentation des cas eux-mêmes, puis la représentation des indices, de leurs types et des vecteurs de poids. La modélisation proposée s'appuie en particulier sur le patron *Composite* pour structurer les indices.

1.2.1 Cas et situation

Dans notre modélisation, un cas est constitué de deux composants par défaut (cf. Figure IV-14) :

- un identificateur (classe Key) qui permet d'identifier un même cas dans différents systèmes, dans des fichiers d'échange et dans les dialogues avec l'utilisateur. La clef d'un cas peut prendre n'importe quelle forme (chaîne de caractères, entier ou autre) et peut être spécifiée en spécialisant la classe Key.
- une situation (classe CompoundIndice) qui donne un ensemble d'indices effectifs d'un cas parmi ces indices potentiels.

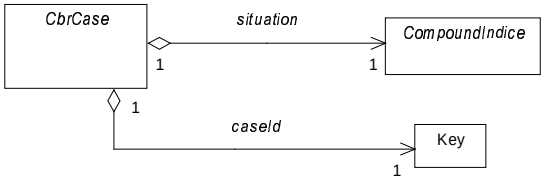


Figure IV-14 : cas et situation

Enfin, la classe CbrCase est un point d'ouverture représentant les cas, qui doit être spécialisé pour contenir les informations spécifiques à une application : informations de gestion (date de création, auteur), représentation du problème (en dehors des indices) et de la solution.

1.2.2 Indices structurés et poids d'importance

Pour la représentation des indices, nous appliquons le patron de conception *Composite* permettant de représenter de manière uniforme des objets simples ou composés (cf. Figure IV-15) : la classe Indice est l'abstraction des indices simples (classe SimpleIndice) et des indices composés (classe CompoundIndice). Un indice composé fait donc référence à un ensemble ordonné de sous-indices pouvant, à leur tour, être indifféremment des indices composés ou simples. Cette structuration permet de regrouper les indices suivant leur sémantique et sera utilisée dans la recherche, d'une part pour restreindre la mise en correspondance à un groupe d'indices, et d'autre part pour spécifier des mesures de similarité adaptées à chaque groupe.

L'expression d'un type d'indices composés correspond à la notion de *classe* d'un langage à objets. Il s'agit en effet de décrire les propriétés nécessaires au raisonnement pour chaque sous-indice : domaine de valeurs avec ou sans relation d'ordre, valeurs maximale et minimale si elles sont définies, convention de représentation des valeurs inconnues ou manquantes. Nous ne supposons pas la présence d'un métaniveau au sein du langage hôte utilisé pour l'implantation, qui nous permettrait de définir une métaclasse afin de d'ajouter les informations nécessaires au niveau de chaque type d'indices composés. Nous complétons alors la définition d'un type d'indices composés (classe CompoundIndice) par des informations supplémentaires (classes IndiceClassDescriptor et IndiceDescriptor). Ces informations sont partagées par les instances d'un même type d'indices composés et permettent principalement de connaître le type de chaque sous-indice (point d'ouverture IndiceType).

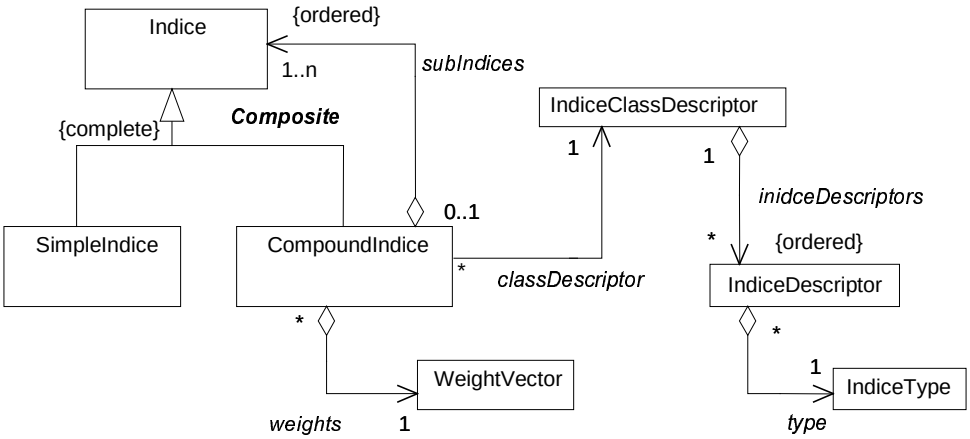


Figure IV-15 : indices structurés et types d'indices

A chaque indice composé est associé un vecteur de poids. Le poids d'importance de chaque indice n'a de sens que comparativement aux autres poids. C'est pourquoi un poids n'est pas un attribut d'un indice simple. De plus, la représentation d'un vecteur de poids comme un objet (point d'ouverture

WeightVector) permet la modification aisée de l'ensemble des poids d'un indice composé, l'encapsulation de la gestion de la cohérence des poids (normalisation des poids par exemple) et l'optimisation mémoire de la représentation. Nous avons défini une hiérarchie de vecteurs de poids (cf. Figure IV-16) : vecteurs donnant le même poids pour chaque sous-indice (classe ConstantWeightVector), vecteurs permettant d'associer à chaque sous-indice un poids différent (classe ArrayWeightVector), et des vecteurs calculant les poids de manière décroissante en fonction de la position du sous-indice dans la relation de composition (classe DecreasingFctWeightVector). Cette hiérarchie est extensible et d'autres types de vecteurs peuvent être définis. Nous avons également appliqué le patron de conception *Prototype*, ainsi les vecteurs de poids peuvent être clonés pour les affecter dans différents cas.

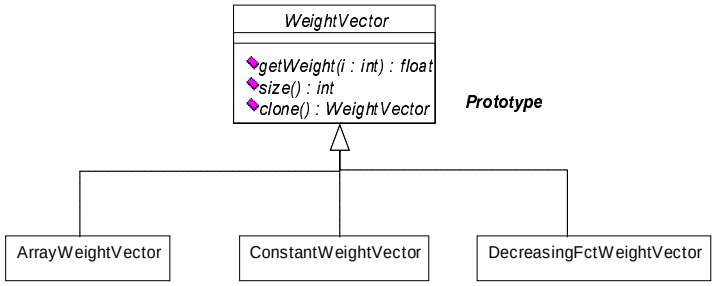


Figure IV-16 : hiérarchie de poids d'importance pour les indices

La Figure IV-17 donne un exemple de la représentation d'un ensemble structuré d'indices. Il s'agit d'une application chargée de calculer un facteur de risque pour un assureur automobile. Les indices d'un cas donnent les caractéristiques d'une voiture ayant un impact sur ce facteur. Dans cet exemple, nous avons séparé les attributs liés à la performance, au design et au prix. Nous montrons plus précisément les indices simples associés à la taille de la voiture (hauteur, largeur et longueur). Un vecteur de poids est associé à chaque indice composé.

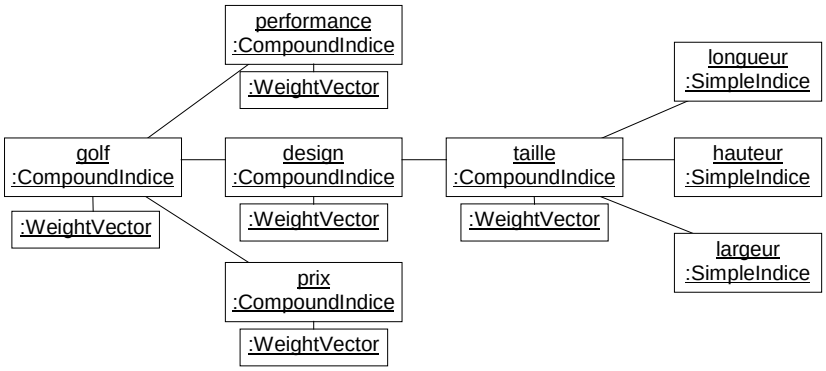


Figure IV-17 : extrait du diagramme d'objets représentant les indices d'un cas

1.3 Organisation de la mémoire

Dans cette section, nous précisons la conception des 11 points d'ouverture pour l'organisation de la mémoire que nous avons précédemment introduits (cf. chapitre III, §2.3.1.3). Plus précisément, pour l'organisation des cas, nous proposons l'utilisation de bases de cas structurées améliorant ainsi la manipulation des cas durant leur cycle de vie, quelle que soit leur localisation physique. Pour l'organisation de l'indexation, nous définissons le concept *d'index* encapsulant à la fois l'algorithme de recherche et la structure de données associée. Ces index sont également structurés et permettent ainsi d'utiliser finement les connaissances du domaine et les caractéristiques des indices pour

construire des stratégies de recherche adaptées. Enfin, nous proposons une modélisation à objets des mesures de similarité aisément modifiables et extensibles. Cette modélisation met en avant les patrons de conception *Façade*, *Composite*, *Stratégie* et *Prototype*.

1.3.1 Indépendance des index et du stockage des cas

Nous proposons de séparer la responsabilité du stockage (logique et physique) des cas et celle de la gestion de leur indexation (cf. Figure IV-18). La mémoire (classe Memory) est alors composée d'une base de cas (classe CaseBase) et d'une base d'index (classe IndexBase). Nous faisons ainsi apparaître trois points d'ouverture dans lesquels la mémoire est une façade (patron de conception *Façade*) gérant l'accès global aux cas. Une base de cas encapsule l'accès aux cas étant donné leur identificateur : elle permet de parcourir l'ensemble des cas, de connaître leur nombre et d'ajouter ou de retirer un cas de cet ensemble. La base d'index permet de réunir un ensemble d'index dont chacun permet l'accès aux identificateurs des cas indexés à partir de leurs indices.

Cette modélisation simplifie la maintenance de la base de cas et des index. En effet, elle permet d'effectuer des modifications et des évolutions tout en gardant le contrôle de la cohérence sous la responsabilité de la mémoire. Par exemple, l'ajout d'un cas peut être effectué directement dans une base de cas ou au niveau de la mémoire. Dans ce dernier cas, il sera automatiquement à la fois ajouté dans la base de cas et indexé dans l'index par défaut. De même, le retrait d'un cas au niveau de la mémoire implique son retrait de la base de cas et des index. D'autres fonctions de gestion sont également définies (cf. Figure IV-18) : reconstruction des index à partir des cas de la base (méthode buildFromCaseBase) et effacement des cas non indexés (méthode deleteNotIndexedCases). Cette gestion peut alors être modifiée ou enrichie en spécialisant le comportement de la mémoire.

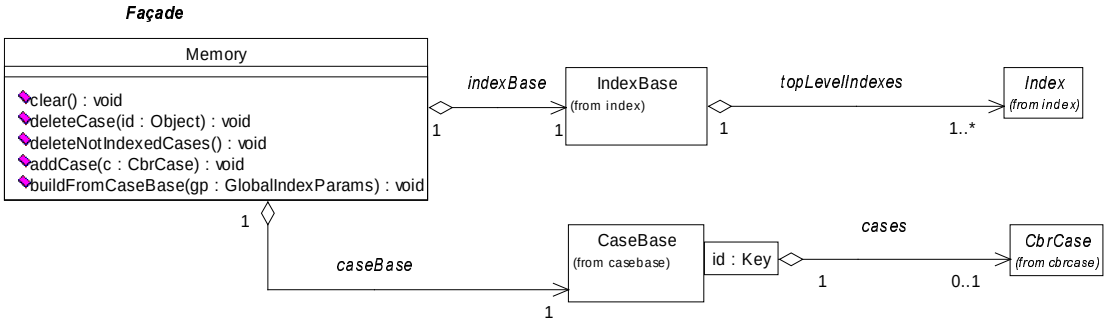


Figure IV-18 : délégation du stockage et de l'indexation

1.3.2 Bases de cas structurées : organisation logique et physique des cas

Dans notre modélisation, nous offrons un point d'ouverture pour l'organisation hiérarchique des bases de cas suivant, d'une part leurs caractéristiques logiques liées au cycle de vie des cas et, d'autre part leur localisation physique. Dans ce but nous appliquons le patron de conception *Composite* (cf. Figure IV-19) : la classe CaseBase est alors la classe de base d'une hiérarchie de classes séparant les bases de cas composées (classe CompoundCaseBase) et les bases de cas simples (classe SimpleCaseBase). Chaque base de cas peut être ouverte en deux modes (en lecture seule ou en modification) et possède des informations sur les données (classe CaseBaseMetadata). L'ajout d'un nouveau cas dans une base n'est correct que s'il est compatible (méthode isCaseCompatible). Ce critère est utilisé pour ajouter un nouveau cas par classification dans la hiérarchie formée par des bases de cas composées.

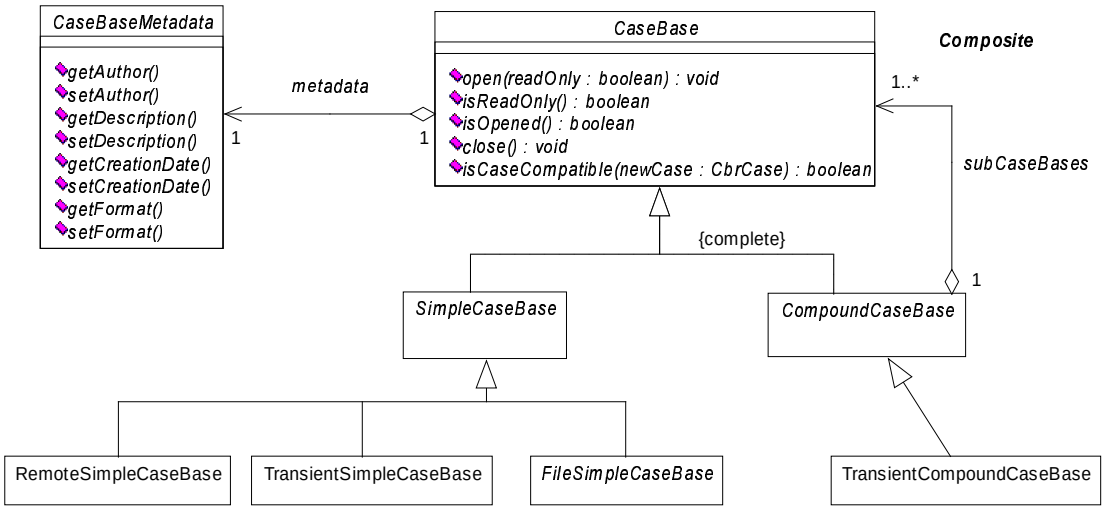


Figure IV-19 : hiérarchie de bases de cas

Plusieurs types de bases de cas simples peuvent alors être définis en spécialisant la classe SimpleCaseBase : bases de cas volatiles (classe TransientSimpleCaseBase), bases de cas avec sauvegarde sur un fichier (classe FileSimpleCaseBase), ou bases de cas distantes (classe RemoteSimpleCaseBase). Les bases de cas composées servent à regrouper les bases de cas suivant une sémantique liée à l'application. Les bases de cas doivent assurer l'unicité de l'affectation des identificateurs aux nouveaux cas. Dans ce but, nous fournissons des objets chargés du calcul des identificateurs (classe IdentifierGenerator). Soit les bases de cas utilisent des générateurs de type différent, l'unicité est alors garantie de par la conception des bases, soit plusieurs bases utilisent le même générateur. Un générateur est en fait une version objet de la notion de *séquence* du système de gestion de bases de données Oracle (Oracle, 1992, pp. 4-188–192).

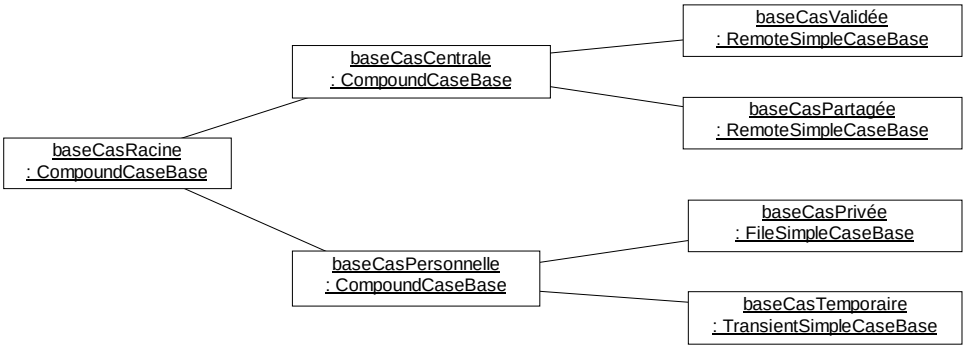


Figure IV-20 : exemple d'une structuration de bases de cas

La Figure IV-20 donne un exemple dans lequel deux principales bases de cas sont réunies : la base de cas personnelle de l'utilisateur et la base de cas centrale. La base de cas personnelle est à son tour décomposée en une base de cas temporaire et une base de cas privée. La base de cas temporaire contient les cas créés lors d'une session de raisonnement. La base de cas privée contient les cas dont l'utilisateur ne souhaite pas une utilisation générale. D'un autre côté, la base de cas centrale contient d'une part les cas publiés par les utilisateurs et n'ayant pas encore été validés (base de cas partagée), et d'autre part les cas acceptés et vérifiés par l'administrateur ou les experts de l'application (base de cas validée). Cet exemple met en évidence l'apport de cette structuration pour mieux suivre les cas dans leur cycle de vie et de validation, tout en montrant l'utilisation des différents types de bases de cas simples. Grâce au patron *Composite* cette structuration est transparente pour le raisonnement.

1.3.3 Index et stratégie de recherche

Un index est un objet qui encapsule une structure de données et une méthode de recherche d'un ensemble de cas sources suivant la situation du cas cible. Nous proposons tout d'abord l'association d'un filtre d'indices à un index permettant ainsi à chaque index de prendre en compte un sous-ensemble d'indices suivant une représentation adaptée. Nous définissons ensuite une hiérarchie extensible d'index mettant l'accent sur la possibilité de concevoir des index composites pour la réalisation de stratégies de recherche complexes. Enfin, nous étudions les conditions de composition permettant de garantir la cohérence de la construction d'une telle stratégie.

1.3.3.1 *Index et filtrage des indices*

Plus précisément, un index offre un ensemble d'opérations (indexation ou retrait d'un cas, reconstruction de sa structure interne) et permet notamment la recherche d'un ensemble de cas suivant une situation (méthode searchCases). Un index offre également un accès direct à la liste des cas qu'il indexe, il est ainsi possible de consulter les propriétés de cette relation (nombre de cas indexés, test d'appartenance).

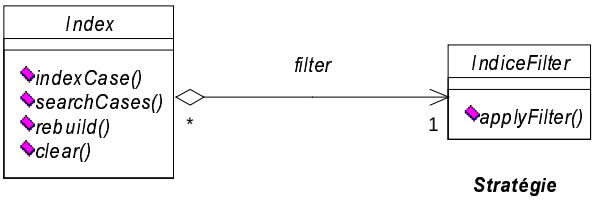


Figure IV-21 : index et filtre d'indices

Nous introduisons la notion de *filtre d'indices* (classe IndiceFilter) permettant de modifier l'ensemble des indices pris en compte à partir de la structure générale des situations (cf. Figure IV-21). Un filtre permet d'effectuer deux types d'opérations (qui peuvent être combinées) sur les indices :

- *sélection* d'une sous-partie,
- *transformation* de la représentation.

Par exemple, il est possible de concevoir un index n'utilisant que la sous-partie décrivant le design d'une voiture (indice composé design dans l'exemple de la Figure IV-17), au lieu de prendre l'ensemble des indices (design, prix et performance). Un index peut également nécessiter de regrouper les indices de manière différente (vecteur à plat par exemple) ou bien d'inclure des indices calculés à partir des indices initiaux. Un filtre permet la construction de cette représentation spécifique. Si cette transformation est coûteuse, elle devra être faite au niveau de la représentation initiale, ainsi la transformation sera réduite à une sélection. Un filtre est utilisé par l'index lors de la recherche ou de l'ajout de cas. Cette conception suit le patron *Stratégie*, ce qui permet une évolution indépendante des filtres et leur partage entre différents index.

1.3.3.2 *Hiérarchie extensible d'index*

Pour la construction de stratégies complexes de recherche, nous proposons une hiérarchie extensible d'index pouvant être composés (cf. Figure IV-22). Dans ce but, nous utilisons encore une fois le patron de conception *Composite* : la classe Index est la racine de la hiérarchie d'héritage comportant notamment les index simples (classe SimpleIndex) et les index composés (classe CompoundIndex). Grâce à cette modélisation, il est possible d'utiliser l'index adapté à un sous-espace des indices ou aux caractéristiques d'indices spécifiques (notamment les propriétés de leur domaine de valeurs), tout en les combinant pour former une organisation complète. Au sein de cette structure

hiérarchique, l'indexation d'un nouveau cas ou la recherche des cas sources sont effectuées suivant un parcours récursif dans lequel chaque index composite contrôle l'exécution.

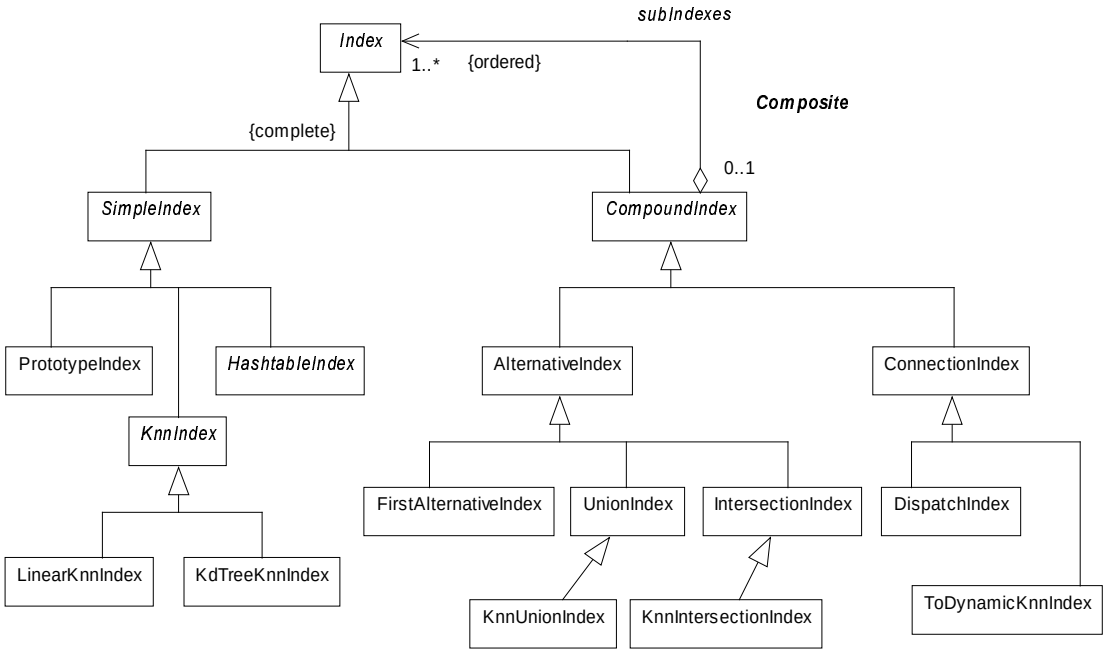


Figure IV-22 : hiérarchie extensible d'index

Les index simples permettent la manipulation d'une seule structure de données, utilisée par une méthode de recherche de cas. Par exemple, dans CBR*Tools, nous offrons quatre index simples :

- un index de prototypes (classe PrototypeIndex) : cet index repose sur la définition d'une hiérarchie de prototypes où chaque prototype[40] définit une fonction de compatibilité. Un prototype est alors relié à un ensemble de cas compatibles. La recherche s'effectue par une opération de classification dans cette hiérarchie, et les identificateurs des prototypes compatibles ainsi que les cas reliés à ces prototypes sont retournés. La représentation des prototypes et le protocole de classification peuvent être spécialisés. Cet index permet de réaliser des arbres discriminants construits *a priori* ou avec un algorithme d'induction. La compatibilité d'un cas avec un prototype peut être booléenne ou basée sur le calcul d'une similarité.
- un index à base d'une table de hachage (classe HashtableIndex) : cet index permet de retrouver un ensemble de cas à partir d'une clef calculée en prenant en compte un sous-ensemble d'indices.
- un index par plus proches voisins linéaire (*K-Nearest Neighbours*, classe LinearKnnIndex) : cet index parcourt un ensemble de cas sources, calcule les similarités entre les situations des cas sources et celle du cas cible. Finalement, les *k* meilleures situations sont retenues suivant une relation d'ordre définie sur le domaine de valeurs des similarités. Cet index est le plus souple dans son utilisation : il permet de gérer tout domaine de valeurs (y compris avec des valeurs manquantes) du moment qu'une similarité a été définie. Par contre, la complexité de cette recherche est linéaire en fonction du nombre de cas à parcourir, dans la mesure où le calcul des similarités et le nombre d'indices sont négligeables devant le nombre de cas.

[40] Au sens de l'outil REMIND (cf. chapitre III, §1.1.1).

- un index par plus proches voisins optimisé, utilisant des *k*-d trees (classe KdTreeKnnIndex) : cet index utilise les relations d'ordre définies sur les domaines de valeurs des indices pour construire un arbre partageant l'espace de recherche en sous-espaces. Le résultat est identique à une recherche linéaire par plus proches voisins, mais certains sous-espaces sont d'office éliminés d'après les propriétés de construction (Wess *et al.,* 1994). La recherche est alors plus efficace[41] (complexité moyenne en $O(\log_2 n)$, où n est le nombre de cas) mais nécessite la construction d'un arbre au préalable (complexité moyenne en $O(k\, n \log_2 n)$).

Nous pouvons identifier deux types d'index composés dont le comportement diffère lors de leur utilisation :

- les index *d'embranchement* (sous-classes de AlternativeIndex) qui permettent d'effectuer des recherches suivant un ensemble de sous-index et de traiter les résultats pour les réunir. Par exemple, nous proposons d'effectuer : l'union des résultats (classe UnionIndex), l'intersection des résultats (classe IntersectionIndex), ou la sélection du premier sous-index donnant des résultats de recherche satisfaisants (classe FirstAlternativeIndex).
- les index de *connexion* (sous-classes de ConnectionIndex) qui permettent d'utiliser deux index en séquence : les résultats d'un premier index sont utilisés comme paramètres pour le second. Par exemple, l'index de répartition (DispatchIndex) permet d'activer des sous-index suivant les résultats du premier index. L'index de connexion vers un index dynamique (ToDynamicKnnIndex) permet d'opérer une recherche uniquement en prenant en compte les cas identifiés par le premier index.

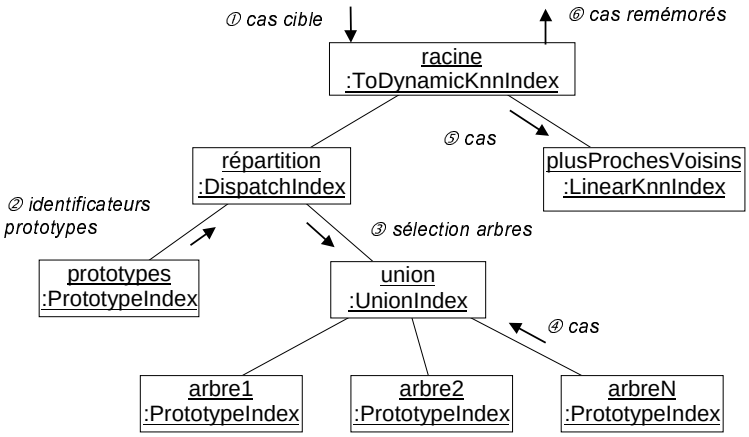


Figure IV-23 : exemple de composition d'index

La Figure IV-23 donne l'exemple d'un diagramme d'objets représentant une stratégie de recherche basée sur la combinaison des techniques d'indexation de REMIND (cf. chapitre III, §1.1.1) : prototypes puis arbres d'induction (ou plus généralement arbres discriminants) et plus proches voisins. Nous montrons ainsi que notre modélisation permet d'effectuer ce type de combinaison tout en apportant une flexibilité accrue : des étapes peuvent en effet être ajoutées ou supprimées suivant les besoins. Dans cette structure, la recherche des cas sources les plus utiles au raisonnement, pour un cas cible donné, est effectuée par un parcours récursif en profondeur d'abord. Pour commencer, un ensemble de prototypes compatibles avec le cas cible est identifié (index prototypes). Chaque prototype représente un ensemble de cas en compréhension dont l'identificateur est relié sémantiquement à un arbre discriminant. L'index prototypes retourne les identificateurs des prototypes compatibles et l'index union continue le parcours récursif en ne considérant que les arbres

[41] L'implantation de cet index dans CBR*Tools a été réalisée par M. Golinelli durant un stage sous notre encadrement (Golinelli, 1997). De nombreux tests comparatifs avec l'index par plus proches voisins linéaire ont été également réalisés.

discriminants ainsi sélectionnés. Chaque arbre permet d'identifier un sous-ensemble de cas (index arbre1 à arbreN). L'union de ces ensembles (index union) est alors passée en paramètre pour effectuer une recherche par plus proches voisins (index plusProchesVoisins). Finalement, les meilleurs cas suivant cette recherche sont retournés (index racine).

Un parcours similaire est effectué lors de l'indexation d'un nouveau cas. Le nouveau cas est tout d'abord classé dans les prototypes (index prototypes) puis ajouté dans les arbres discriminants associés aux prototypes (index arbre1 à arbreN). Si la structure interne des arbres discriminants n'est pas incrémentale, les index doivent être reconstruits pour que le cas soit pris en compte.

1.3.3.3 *Communication entre les index et cohérence de l'assemblage*

A partir des index introduits précédemment, nous avons identifié six types de résultats rendus par un index lors d'une recherche ou d'une indexation. Ces types de résultats sont organisés de manière hiérarchique (cf. Figure IV-24) :

- *Ensemble de cas* (interface CaseSetIndexResult).
- *Ensemble ordonné de cas suivant leur similarité* (interface CaseSimIndexResult) : une similarité est associée à chaque cas, l'ensemble des cas est alors une liste ordonnée suivant cette similarité.
- *Ensemble ordonné de cas suivant leur similarité, complété de leur similarité détaillée* (interface CaseDetailedSimIndexResult) : chaque cas possède une similarité détaillée au niveau de chacun de ses indices.
- *Sous-ensembles de cas* (interface CaseSubsetIndexResult) : un ensemble d'identificateurs de sous-ensembles de cas est défini (sous-ensembles de cas en compréhension).
- *Sous-ensembles en extension* (interface CaseSubsetExtensionIndexResult) : les sous-ensembles sont en extension, c'est-à-dire que chaque identificateur de sous-ensemble est lié à un ensemble en extension de cas.
- *Sous-ensembles évalués* (interface CaseSubsetSimIndexResult) : chaque sous-ensemble est lié à une similarité.

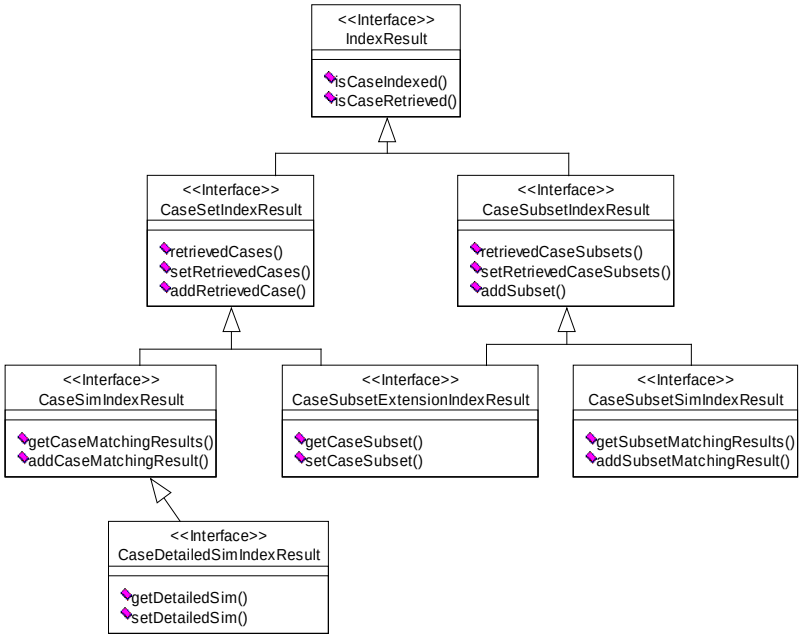


Figure IV-24 : hiérarchie d'interfaces des résultats d'un index (recherche et indexation)

Comme nous venons de le montrer, les index peuvent être composés pour former des structures complexes. Lors d'une recherche, chaque index est utilisé pour affiner progressivement la sélection des cas. Lors d'une indexation, chaque index met à jour sa structure en fonction des mises à jour faites par les autres index. Ainsi, au sein d'une stratégie complexe, un index utilise les résultats des index déjà parcourus. Nous proposons alors de passer en paramètres, lors de l'utilisation d'un index, deux types de résultats :

- le résultat courant (cf. Figure IV-25),
- les résultats intermédiaires qui sont sauvegardés dans une table.

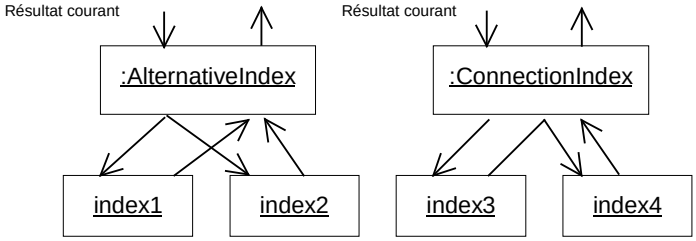


Figure IV-25 : transmission du résultat courant par les index composés

Pour assurer la cohérence d'un assemblage d'index, il s'agit de déterminer si chaque index dispose de l'accès aux résultats attendus. Nous utilisons la notion de *profil* que nous avons introduite pour l'assemblage des phases du raisonnement (cf. §1.1.1.2.1). Un profil est un ensemble d'interfaces issues de la hiérarchie dont la classe de base est IndexResult. Chaque index définit un *profil de paramètre* pour le résultat courant, ainsi que pour chaque entrée de la table des résultats intermédiaires dont il a besoin. De même, chaque index définit un *profil de résultat* indiquant le type de données retournées. Nous représentons alors un objet index avec ses profils de paramètre et de résultat en les plaçant respectivement en-dessous et au-dessus, comme l'exemple de la Figure IV-26. Le terme « pour » indique qu'il s'agit d'un profil de paramètre pour un résultat intermédiaire sauvegardé dans la table sous un nom particulier (« Index0 » dans l'exemple). Si ce terme est absent, le profil porte sur le résultat courant. Le terme optionnel « dans » indique que le résultat de l'index, respectant le profil de résultat mentionné, est sauvegardé dans la table en lui assignant un nom (« index1 » dans l'exemple).

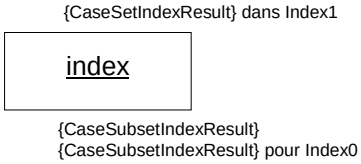


Figure IV-26 : indication des profils associés à un index

Les index simples définissent explicitement leurs profils qui peuvent toutefois être sélectionnés parmi plusieurs possibles suivant leur configuration. Les index composés transmettent le résultat courant et peuvent poser des contraintes sur celui-ci. Par exemple, pour les index d'embranchement, le profil de paramètre est, soit {CaseSubsetIndexResult}, soit le profil vide. Dans le premier cas, chaque identificateur de sous-ensemble de cas est associé à un des sous-index. Lors de la recherche ou de l'indexation, la présence d'un identificateur rendra actif le sous-index associé. Le profil de résultat est par défaut égal au profil du dernier sous-index. De plus, un index d'embranchement comme UnionIndex peut imposer un type de résultat pour ses sous-index de manière à pouvoir effectuer son opération (une union). De manière similaire, l'index de connexion ToDynamicKnnIndex impose que le profil de paramètre du second sous-index soit au minimum {CaseSetIndexResult}.

La vérification de la cohérence d'une stratégie doit se faire lors de la conception, les profils servant de moyens pour expliciter les contraintes. Ces profils sont alors utilisés pour configurer concrètement les index et indiquer s'il doivent sauvegarder leur résultat dans la table des résultats intermédiaires.

Dans la Figure IV-27, nous reprenons l'assemblage des index de la Figure IV-23 en montrant la configuration des profils des index. Cet assemblage respecte les contraintes énoncées. L'index prototypes ne suppose rien sur le résultat courant et retourne un résultat de type {caseSubSetIndexResult} qui vérifie le profil de paramètre de l'index union. Le résultat courant est alors transmis à chaque sous-index sélectionné parmi arbre1 à arbreN. Ces index ont un profil de paramètre vide et retourne un résultat de type {CaseSetIndexResult} qui vérifie le profil de résultat imposé par l'index union. A son tour, ce résultat vérifie le profil de paramètre de l'index plusProchesVoisins qui finalement retourne un résultat de type {caseSimIndexResult}.

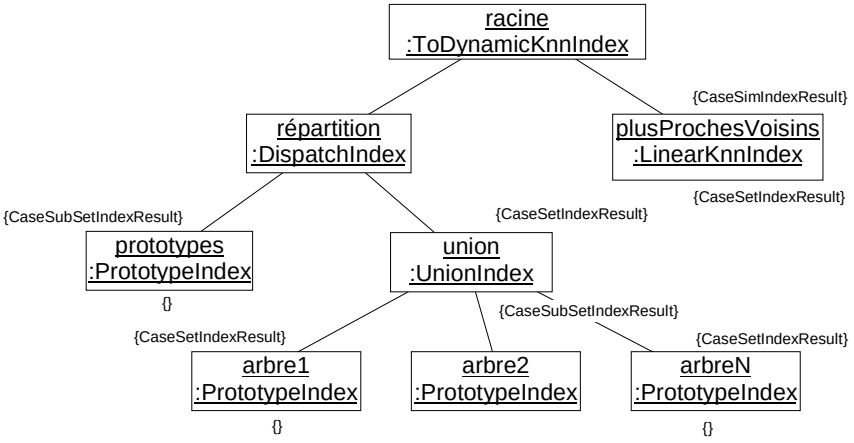


Figure IV-27 : exemple d'assemblage valide des index

1.3.4 Similarités structurées, modifiables et extensibles

La modélisation que nous proposons repose sur une représentation structurée des similarités et permet une modification aisée de leur définition.

1.3.4.1 *Domaines de valeurs des similarités*

Le résultat du calcul d'une mesure est un objet de classe CmpValue. L'encapsulation des valeurs de comparaison dans des objets permet de gérer différents types de résultats (point d'ouverture CmpValue). Par exemple, CBR*Tools offre trois types de valeurs (cf. Figure IV-28) :

- des booléens (classe CmpBoolValue),
- des *facteurs* de similarité donnant un réel élément de [0,1] (classe CmpFactorValue) comme dans la plupart des systèmes de RàPC,
- et des couples (possibilité, nécessité) (classe CmpPossibilityValue). Ce dernier domaine de valeurs est utilisé lorsqu'il est nécessaire de prendre en compte des indices dont les valeurs sont imprécises ou incertaines. Par exemple, ce domaine est utile dans le cadre de systèmes de RàPC en conception (Visser & Trousse, 1993 ; Trousse & Visser, 1993). Dans la suite du document, nous ne développerons pas l'utilisation de ce domaine, et le lecteur intéressé pourra se reporter à (Jaczynski & Trousse, 1994 ; Jaczynski, 1993).

D'autres types de valeurs peuvent être ajoutés par spécialisation de la classe CmpValue. Tout domaine de valeurs doit être muni d'une relation d'ordre permettant d'utiliser la similarité et de constituer des classements notamment dans un index par plus proches voisins. Nous avons défini une

hiérarchie extensible de relations d'ordre (point d'ouverture CmpValueOrder), permettant de définir d'une part des relations adaptées aux nouveaux domaines de valeurs éventuels, et d'autre part des relations différentes pour le même domaine de valeurs.

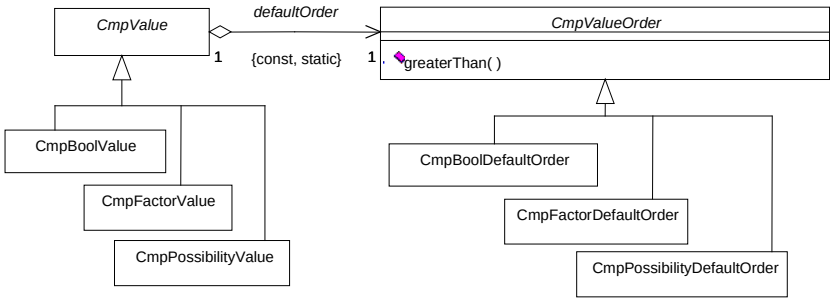


Figure IV-28 : valeurs des similarités et relations d'ordre

1.3.4.2 *Similarités structurées et modifiables*

La représentation des mesures de similarité doit être adaptée à la représentation des indices, c'est pourquoi nous utilisons de manière identique le patron de conception *Composite* nous permettant de définir des mesures structurées (cf. Figure IV-29). Une mesure (de classe Similarity) peut donc être une mesure de similarité adaptée à un indice simple (classe SimpleIndiceSimilarity) ou à un indice composé (classe CompoundIndiceSimilarity).

Nous appliquons également le patron de conception *Stratégie* pour la représentation des fonctions élémentaires de similarité et des fonctions d'agrégation. Ce patron permet la définition d'une famille de fonctions parmi lesquelles il est possible de sélectionner celle à utiliser. Ainsi, l'objet de similarité d'un indice simple délègue le calcul de la similarité à une fonction de similarité (classe ValueSimilarityFct). De même, l'objet de similarité d'un indice composé délègue le calcul de l'agrégation des valeurs de similarité issues de la comparaison des sous-indices, à une fonction d'agrégation (classe AggregationFct). Un vecteur de poids d'importance est également défini au niveau de chaque objet de similarité pour les indices composés. Enfin, les objets de similarité utilisent le patron de conception *Prototype* qui permet de créer une nouvelle similarité par clonage d'une similarité existante (méthode clone).

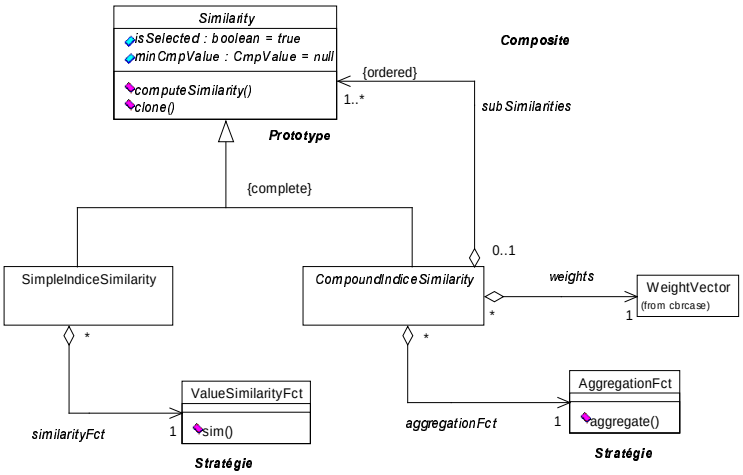


Figure IV-29 : mesures de similarité structurées et modifiables

Cette modélisation permet une modification aisée des mesures de similarité. Il est tout d'abord possible d'affecter dynamiquement de nouvelles fonctions élémentaires de similarité ou des fonctions

d'agrégation sans modifier la structure de la mesure de similarité. D'autre part, le clonage des similarités permet de gérer des mesures qui ne diffèrent que par les modifications apportées. Nous pouvons ainsi constituer de manière simple une bibliothèque de mesures de similarité. Cette modélisation présente également une bonne répartition des responsabilités : les objets formant la structure de la mesure de similarité contrôlent le calcul, alors que les fonctions sont utilisables dans différentes mesures et sont issues de hiérarchies de classes pouvant être étendues de manière indépendante. Le calcul est effectué par un parcours récursif de la structure : les mesures de similarité d'un indice simple contrôlent l'appel de la fonction élémentaire de similarité, et les mesures de similarité d'un indice composé contrôlent l'appel récursif aux sous-similarités ainsi que l'agrégation des résultats intermédiaires. Le calcul de la similarité suppose que les deux ensembles d'indices à comparer et la représentation de la similarité aient une structure identique. Si deux cas de structures différentes doivent être comparés, la similarité peut être construite dynamiquement pour refléter la structure commune aux deux cas.

Nous proposons la prise en compte de deux types de paramètres lors du calcul de la similarité : les *paramètres de construction* et les *paramètres d'évaluation* :

- Les paramètres de construction permettent de modifier finement le comportement de la similarité au niveau de chaque objet de la structure (attributs de la classe Similarity). Deux paramètres de construction sont proposés dans CBR*Tools permettant de fixer un seuil de similarité minimum au niveau de chaque indice (simple ou composé), et de sélectionner les indices qui participeront au calcul. Le premier paramètre est utile pour assurer un certain degré de similarité et permet d'anticiper l'échec d'une comparaison : si le seuil n'est pas respecté, le calcul de la similarité est de suite abandonné et les comparaisons restantes ne sont pas effectuées. Le deuxième paramètre permet d'utiliser une structuration de la similarité existante tout en sélectionnant les indices effectivement utilisés. Les paramètres de construction peuvent être modifiés et leur changement est persistant.
- Les paramètres d'évaluation sont des objets (de classe SimilarityEvalParams) transmis lors du calcul. Ces paramètres sont donc facilement modifiables à chaque appel et permettent notamment : l'activation des seuils de similarité, la sélection d'une méthode de choix des poids d'importance, et la définition de la valeur à utiliser lorsqu'un indice simple a une valeur inconnue. La méthode de sélection des poids est intéressante, puisque nous pouvons associer des poids d'importance au niveau des indices des cas sources, du cas cible et de la similarité. Les poids associés à chaque cas source indiquent quelles sont les importances relatives des indices permettant de juger de l'utilité du cas, et sont mis à jour lors de l'apprentissage. Les poids associés aux indices du cas cible proviennent d'une évaluation faite *a priori* lors de l'élaboration au début d'un raisonnement. Les poids issus de la similarité sont considérés comme des poids par défaut qui ont été estimés lors de la construction de la similarité. La sélection des poids est déléguée à un objet passé en paramètre, permettant ainsi de sélectionner ou de combiner les poids de manière ouverte. Par exemple, si l'utilisateur courant est un novice, le système utilisera les poids de la similarité ou celui des cas sources, par contre, si l'utilisateur est un expert, les poids du cas cible qu'il aura spécifiés seront utilisés. Enfin, il est intéressant de pouvoir modifier la valeur de comparaison dans le cas d'un indice de valeur inconnue. Par exemple, un facteur de 0.5 est habituellement utilisé ; cependant, des évaluations optimistes (valeur 1) ou pessimistes (valeur 0) peuvent être choisies.

1.3.4.3 *Hiérarchies extensibles de fonctions élémentaires de similarité et d'agrégation*

Le calcul des similarités s'appuie sur l'utilisation de deux types de fonctions : les fonctions élémentaires de similarité et les fonctions d'agrégation. Pour la représentation de ces fonctions, nous proposons deux hiérarchies extensibles de classes. La hiérarchie des fonctions élémentaires de similarité est donnée dans la Figure IV-30 et l'expression des fonctions est précisée dans Tableau IV-

2. Le premier critère de classification est constitué du domaine de valeurs de la fonction. Nous développons uniquement les sous-classes permettant de calculer des facteurs de similarité, et les sous-classes applicables pour n'importe quel type de domaine. Lors du calcul de la comparaison, le descripteur de l'indice considéré est passé en paramètre, ce qui permet à la méthode de calcul, d'une part, de vérifier que les propriétés nécessaires de l'indice sont bien respectées, et d'autre part, de récupérer les données descriptives de l'indice dont notamment son domaine de valeurs (par exemple domaine numérique borné, liste de valeurs).

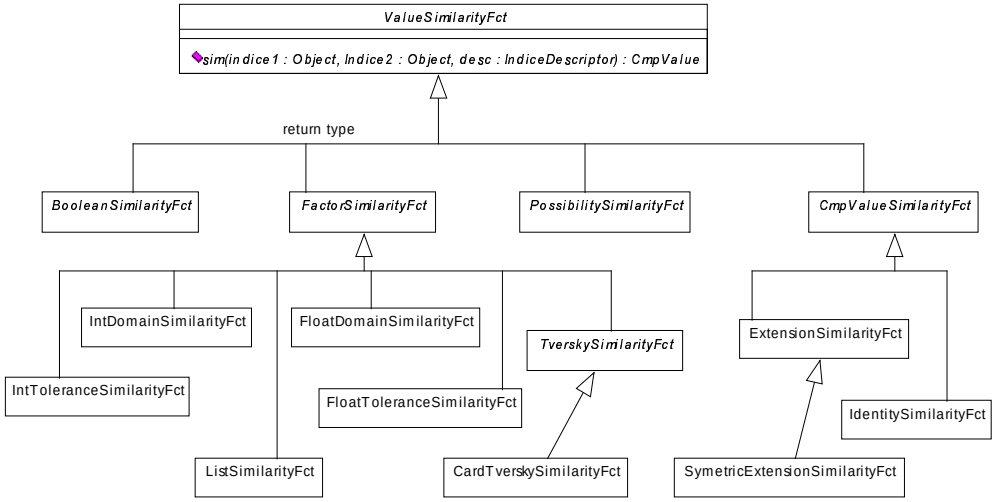


Figure IV-30 : extrait de la hiérarchie des fonctions élémentaires de similarité

Classe	*Fonction*	*Type de valeurs de l'attribut*
ExtensionSimilarityFct	$s(x,y) = M(x,y)$ M matrice de similarité	tout type de valeurs
SymetricExtensionSimilarityFct	$s(x,y) = M(x,y) = M(y,x)$ M matrice de similarité	tout type de valeurs
IdentitySimilarityFct	$s(x,y) = \begin{cases} v_1 \text{ si } x = y \\ v_2 \text{ sinon} \end{cases}$	tout type de valeurs
TverskySimilarityFct	$s(x,y) = \dfrac{f(x \cap y)}{f(x \cap y) + \alpha\ f(x-y) + \beta\ f(y-x)}$ $\alpha \geq 0, \beta \geq 0$	ensemble
CardTverskySimilarityFct	$s(x,y) = \dfrac{\text{Card}(x \cap y)}{\text{Card}(x \cap y) + \alpha\ \text{Card}(x-y) + \beta\ \text{Card}(y-x)}$ $\alpha \geq 0, \beta \geq 0$	ensemble
IntDomainSimilarityFct FloatDomainSimilarityFct	$s(x,y) = 1 - \dfrac{\lvert x-y \rvert}{D_{\max} - D_{\min}}$	valeurs numériques dont le domaine est borné par $D_{\max}$ et $D_{\min}$
IntToleranceSimilarityFct FloatToleranceSimilarityFct	$s(x,y) = \begin{cases} 0 \text{ si } \lvert x-y \rvert \geq d \\ 1 - \dfrac{\lvert x-y \rvert}{d} \text{ sinon} \end{cases}$ $d \geq 0$, est appelé la tolérance	valeurs numériques
ListSimilarityFct	$s(x,y) = 1 - \dfrac{\lvert pos(x) - pos(y) \rvert}{l}$ l longueur du domaine de valeurs pos donne la position d'une valeur sur le domaine.	tout type de valeurs dont le domaine est une liste

Tableau IV-2 : fonctions élémentaires de similarité dans CBR*Tools

De même, nous proposons une hiérarchie de fonctions d'agrégation (cf. Figure IV-31). Les fonctions d'agrégation prennent en paramètres les différentes similarités calculées pour les sous-indices d'un indice composé, et un vecteur de poids. Ce vecteur peut être constitué, suivant l'application, des poids liés aux cas sources, au cas cible ou à la similarité. L'expression des fonctions d'agrégation est précisée dans le Tableau IV-3.

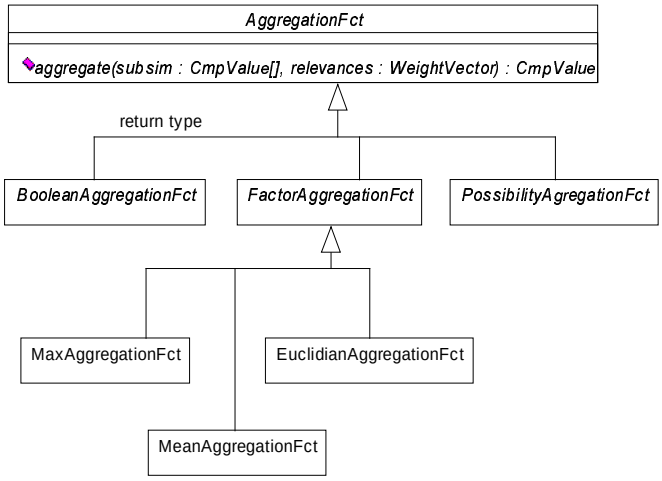


Figure IV-31 : extrait de la hiérarchie des fonctions d'agrégation

Classes	*Fonctions*
MeanAggregationFct	$g(v,w) = \frac{\sum_{i=1}^{n} v_i . w_i}{\sum_{i=1}^{n} w_i}$
EuclidianAggregationFct	$g(v,w) = \frac{\sqrt{\sum_{i=1}^{n} v_i^2 . w_i^2}}{\sqrt{\sum_{i=1}^{n} w_i^2}}$
MaxAggregationFct	$g(v,w) = \max_{i=1..n}(w_i v_i)$

Tableau IV-3 : fonctions d'agrégation de CBR*tools

2 Modèles à objets ouverts pour l'indexation par situations comportementales

L'intérêt de notre modélisation est de proposer un cadre général pour la modélisation du RàPC qui est enrichi par des représentations et des mécanismes spécifiques à l'indexation par situations comportementales. Ainsi, il est possible d'utiliser ce type d'indexation au sein d'une architecture *cohérente* pour construire des systèmes de raisonnement complets. Pour la modélisation de ce type d'indexation, nous avons introduit 12 nouveaux points d'ouverture (cf. chapitre III, §2.3.2) venant compléter les points d'ouverture généraux du RàPC. Ces nouveaux points d'ouverture ne concernent que les axes de la représentation des cas et de l'organisation de la mémoire. En suivant ces deux axes de variabilité, nous montrons comment les points d'ouverture existants sont utilisés et nous introduisons également les modèles à objets ouverts spécifiques pour ce type d'indexation. Les

différents concepts de notre modèle d'indexation (cf. chapitre II, §2) sont en effet représentés sous forme d'objets et de hiérarchies extensibles. Un enregistrement est un objet composé de différents objets chroniques. Une situation comportementale est composée de deux objets, l'un pour la partie instantanée, et l'autre pour la partie comportementale. La classe de ce dernier est une façade offrant ainsi une interface de manipulation simplifiée des indices comportementaux. Enfin, les comportements élémentaires, les fonctions de localisation et les contraintes temporelles sont représentés par des objets issus de hiérarchies extensibles de classes. Cette modélisation met en œuvre les patrons *Façade*, *Composite*, *Itérateur*, *Patron de méthode*, *Fabrication* et *Stratégie*.

2.1 Représentation des cas

Nous introduisons tout d'abord la modélisation des données d'observation puis celle des cas. Pour notre modèle d'indexation, un cas comprend une situation comportementale qui référence les données brutes. Enfin, les patrons de cas potentiels permettent de créer dynamiquement de nouveaux cas à partir des données brutes.

2.1.1 Représentation des données d'observation

Dans notre modèle d'indexation par situations comportementales, les données d'observation sont structurées en chroniques et en enregistrements. Nous présentons la représentation à objets de ces deux concepts, tout en insistant sur les propriétés qui faciliteront leur utilisation dans les mécanismes de l'indexation : encapsulation de l'hétérogénéité des chroniques et représentation d'un instant de référence.

2.1.1.1 *Chroniques*

Une chronique représente une succession de données ordonnées suivant leur occurrence ou leur date de mesure. Nous définissons alors une interface abstraite de manipulation d'une chronique (point d'ouverture TimeSeries) semblable à un tableau (accès aux données par leur position) dont la représentation concrète peut prendre plusieurs formes (cf. Figure IV-32). La hiérarchie des chroniques montre les différents types de chroniques possibles et notamment la chronique avec prévision (classe ForecastTS). Ce type de chroniques permet de prendre en compte les prévisions des observations (comportements futurs estimés) de manière transparente, tout en facilitant leur mise à jour par composition d'une chronique d'observations et d'une chronique de prévisions. Cette modélisation suit les principes du patron de conception *Composite* dans la mesure où la composition de ces deux sous-chroniques reste masquée. Toutefois, il n'est pas utile dans ce cas présent de modéliser une structuration hiérarchique (comme pour les indices, les bases de cas et les similarités).

Enfin, la hiérarchie des chroniques peut être étendue grâce à la définition de nouvelles classes, pour permettre la représentation de types particuliers de chroniques (domaine spécifique de valeurs, sémantique particulière des événements). Toutefois, cette modélisation reste simple et présente l'intérêt de mettre en évidence nos besoins pratiques dans le cadre de l'indexation par situations comportementales. Nous laissons en perspective l'extension de ce modèle, afin de permettre l'utilisation de travaux existants comme ceux relatifs au système de gestion de bases de données TEMPOS (Canavaggio, 1997 ; cf. chapitre II, §1.2.2.2).

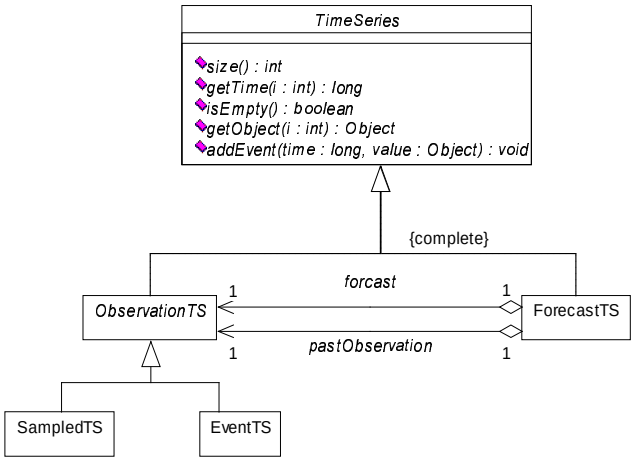


Figure IV-32 : hiérarchie des chroniques

2.1.1.2 *Enregistrements et curseurs*

Un enregistrement regroupe un ensemble de chroniques correspondant aux variables d'observation du processus, pour une période de temps définie et pour un contexte donné (cf. Figure IV-33). Chaque chronique (classe TimeSeries) est accessible depuis l'enregistrement (point d'ouverture Record) qui joue alors le rôle d'une façade (patron *Façade*) pour simplifier et prétraiter l'ajout de mesures dans les chroniques. Le contexte de l'enregistrement est utilisé pour exprimer des indices instantanés qui sont utilisés dans la définition des situations comportementales (cf. section suivante). Le contexte repose alors sur le point d'ouverture général CoumpoundIndice.

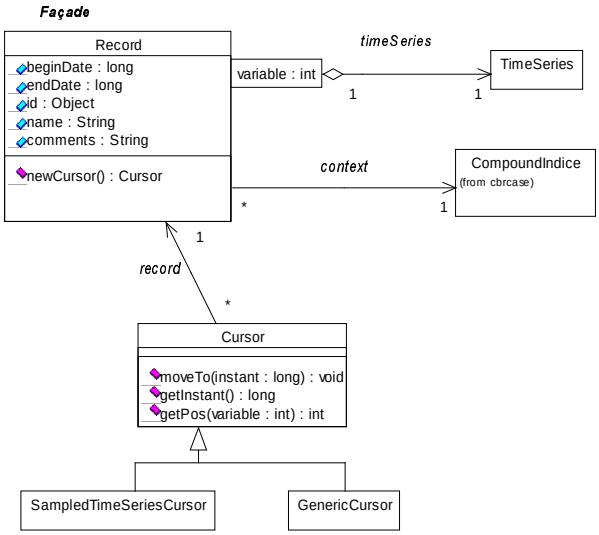


Figure IV-33 : représentation des enregistrements

De plus, dans l'indexation par situations comportementales, il est souvent nécessaire de référencer un instant précis à l'intérieur d'un enregistrement. Cet instant est représenté par un objet nommé *curseur* (point d'ouverture Cursor) qui détient cet instant ainsi que la position de la dernière donnée de chaque chronique ayant une date inférieure ou égale à l'instant de référence (application de l'opérateur Pos sur l'instant de référence, cf. chapitre II, §2.1.3.2). La sauvegarde de ces positions

permet d'éviter de les calculer à chaque manipulation des chroniques. Un curseur peut être déplacé et doit maintenir la cohérence des positions des données des chroniques qu'il référence. Un curseur est un objet abstrait permettant la réalisation de différents types de curseurs pour tirer parti des propriétés des chroniques utilisées. En particulier, la représentation et le déplacement d'un curseur deviennent très simples lorsque toutes les chroniques d'un enregistrement sont échantillonnées à la même fréquence (classe SampledTimeSeriesCursor). Toutefois, il est possible de définir un curseur ne prenant aucune hypothèse spécifique (classe GenericCursor). Enfin, un enregistrement définit une méthode de création d'un nouveau curseur (méthode newCursor suivant le patron *Fabrication*) permettant aux sous-classes de Record de définir le type de curseur par défaut à employer.

2.1.2 Représentation des cas et des situations comportementales

Conformément au modèle d'indexation par situations comportementales, une situation est composée d'une partie instantanée et d'une partie purement comportementale. Nous respectons cette séparation et la situation d'un cas est alors spécialisée (classe TimeExtendedSituation) pour contenir deux composants (cf. Figure IV-34) : l'un pour les indices comportementaux (classe BehaviouralSitiation) et l'autre pour les indices instantanés (classe InstantaneousSituation). Les indices instantanés comprennent principalement les indices du contexte de l'enregistrement auquel est liée une situation. Chaque composant représentant des indices hérite de la classe CompoundIndice pour respecter l'accès uniforme aux indices que nous avons définis. La classe CbrCase est également spécialisée (classe TimeExtendedCase) de manière à indiquer le type du cas (concret ou potentiel) et son origine (par exemple le nom du patron utilisé pour sa création), et pour référencer une situation comportementale.

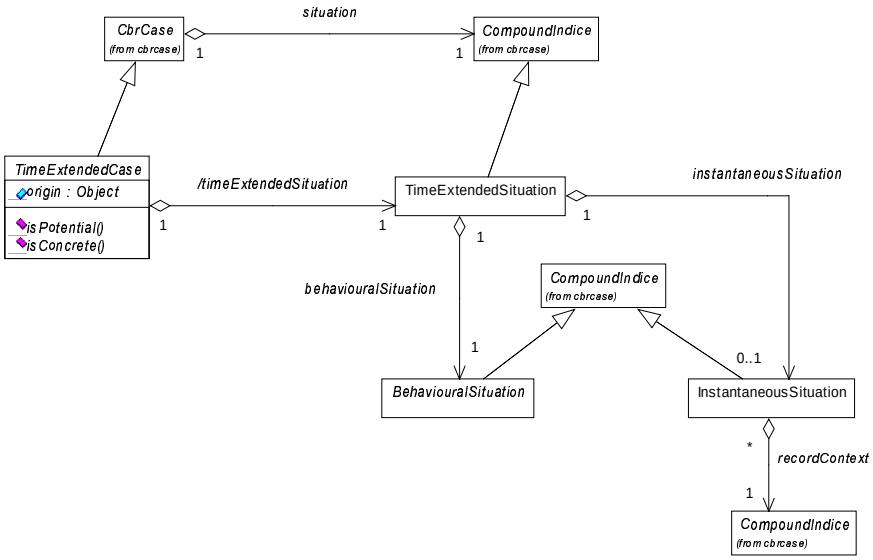


Figure IV-34 : représentation des cas et structuration des indices

La partie comportementale est à son tour décomposée pour obtenir une représentation ouverte des concepts nécessaires (cf. Figure IV-35) : les contraintes temporelles (classe IntervalContraint), les comportements élémentaires (classe RelevantSequence) et les contraintes de localisation (classe SequenceLocation). Suivant le patron *Façade*, la classe BehaviouralSituatuion donne l'accès à ces différents composants tout en simplifiant leur manipulation. En particulier, cette classe doit être spécialisée dans une application concrète pour réaliser les interfaces fonctionnelles des indices (consultation, sélection, modification et analyse). Les comportements élémentaires permettent de sélectionner les données issues d'une chronique et transforment ces données en indices (la classe

RelevantSequence hérite de CoumpoundIndice). De plus, la partie comportementale référence de manière indirecte, sur le plan du typage, les comportements élémentaires. Ainsi, les comportements élémentaires peuvent être regroupés et structurés, de manière spécifique pour chaque application, au sein d'indices composés intermédiaires.

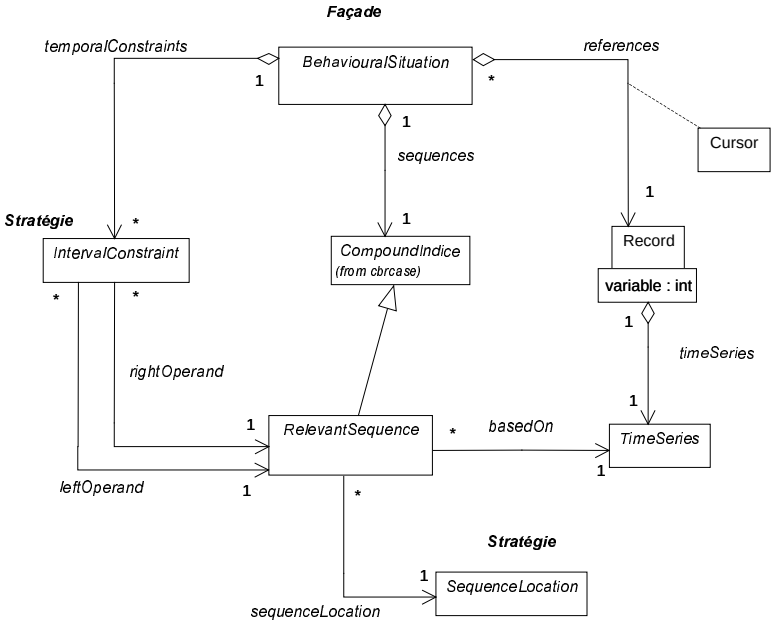


Figure IV-35 : décomposition de la partie comportementale

La représentation et le calcul de la satisfaction d'une contrainte temporelle sont délégués à un objet (classe IntervalConstraint) suivant le patron *Stratégie*. A titre d'exemple, nous avons représenté dans la Figure IV-36 trois contraintes temporelles portant sur la précédence de deux comportements élémentaires (classe IntervalAfter et IntervalBefore) et sur l'inclusion temporelle de comportements élémentaires (classe IntervalDuring). Cette hiérarchie est extensible et permet ainsi la définition des types de contraintes voulus.

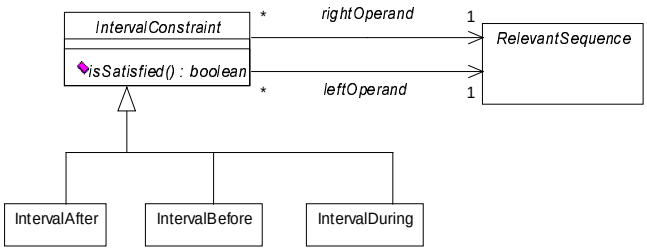


Figure IV-36 : hiérarchie des contraintes temporelles

Suivant le patron *Stratégie*, la gestion d'une contrainte de localisation est également déléguée à un objet implantant une méthode qui retourne un comportement élémentaire homologue dans une composante comportementale donnée. Différents types de contraintes de localisation ont été définis dans notre modèle d'indexation et sont ainsi représentés comme des sous-classes concrètes de la hiérarchie (cf. Figure IV-37) : localisation par temps écoulé (classe ByElapsedTimeSequenceLocation), par position (classe ByPositionSequenceLocation), par valeur (classe ByValueSequenceLocation) et enfin par référence (classe ByReferenceSequenceLocation). Il est donc possible de représenter de manière uniforme les types de contraintes existants et de nouveaux types peuvent être aisément définis.

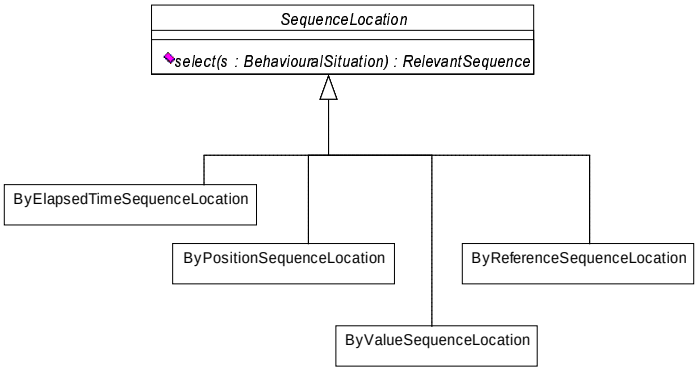


Figure IV-37 : hiérarchie des contraintes de localisation

2.1.3 Représentation des patrons de cas potentiels

La gestion d'un patron de cas potentiels est déléguée à une classe offrant deux méthodes qui peuvent être spécialisées suivant le patron *Patron de méthode* (cf. Figure IV-38) :

- prédicat d'applicabilité du patron (méthode isApplicable),
- instanciation du patron à un instant de référence dans un enregistrement (curseur) pour la création d'un cas potentiel (méthode instanciate).

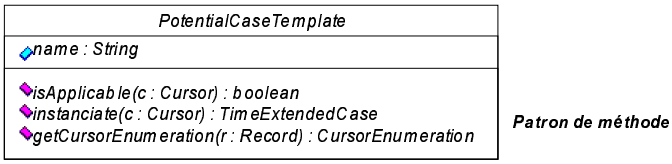


Figure IV-38 : représentation des patrons de cas potentiels

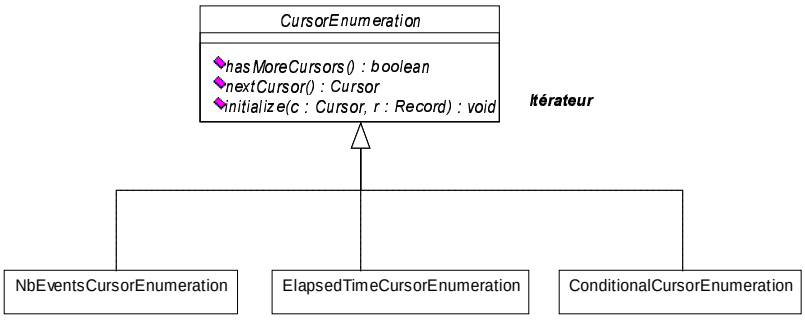


Figure IV-39 : hiérarchie d'algorithmes d'itération sur les enregistrements

Enfin, l'instanciation des cas potentiels nécessite le parcours des enregistrements, c'est-à-dire le calcul au sein d'un enregistrement d'un ensemble d'instants de référence pour lesquels les conditions d'application d'un patron sont valides. Ce parcours peut être contrôlé par un client de l'interface du patron, tout en appelant le prédicat d'applicabilité sur chaque instant. Toutefois, nous proposons d'appliquer le patron de conception *Itérateur* en encapsulant la logique du parcours dans un objet autonome (classe CursorEnumeration). Cette modélisation permet alors de concevoir des itérations adaptées à un patron particulier et aux chroniques des enregistrements considérés, tout en respectant une interface d'itération standard. A titre d'exemple, nous proposons trois types de parcours généraux qui permettent de déplacer un curseur en fonction : d'un nombre d'événements produits sur une variable (classe NbEventCursorEnumeration), d'une durée (classe ElapsedTimeCursorEnumeration) ou d'une condition à satisfaire sur l'une des variables (classe ConditionalCursorEnumeration). Un patron de

cas potentiels encapsule alors le choix du parcours approprié grâce à une méthode de création respectant le patron de conception *Fabrication* (méthode getCursorEnumeration).

2.2 Organisation de la mémoire

Nous rappelons que l'organisation de la mémoire comprend l'organisation des cas et l'organisation de l'indexation. Toutefois, l'indexation par situations comportementales introduit deux concepts spécifiques : les patrons de cas potentiels et les enregistrements. La mémoire d'un système de RàPC est alors étendue (classe TimeExtendedMemory) de manière à intégrer deux points d'ouverture qui gèrent l'accès à ces éléments (cf. Figure IV-40) : la base de patrons de cas potentiels et la base d'enregistrements (respectivement TemplateBase et RecordBase). Le point d'ouverture Memory de l'axe « gestion du raisonnement » est alors spécialisé pour intégrer la déclaration de ces composants. Ainsi, il est possible d'organiser les patrons et les enregistrements suivant les besoins d'une application, tout en offrant une interface d'accès standard. De plus, la mémoire joue encore son rôle de façade (patron *Façade*) pour gérer et simplifier l'accès et surtout l'ajout de nouveaux patrons et enregistrements. Par exemple, la mémoire peut être chargée de propager l'ajout d'un nouvel enregistrement dans les index pour qu'il puisse être utilisé lors d'un prochain raisonnement.

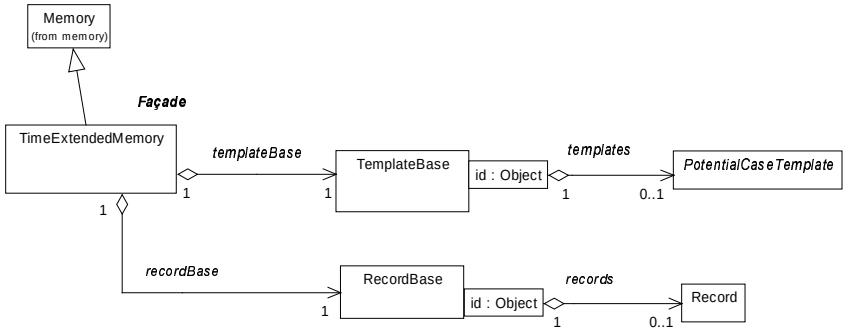


Figure IV-40 : extension de la mémoire

L'organisation des cas proprement dite s'appuie sur le point d'ouverture CaseBase. Dans le cadre de notre modèle d'indexation, une configuration typique de ce point d'ouverture consiste à définir une base composée, formée d'une base de cas volatile pour les cas potentiels et d'une base de cas persistante pour les cas concrets (cf. Figure IV-41). La base de cas volatile comprend les cas potentiels instanciés temporairement au cours d'un raisonnement. Lors de la phase d'apprentissage, ces cas seront, soit effacés, soit intégrés dans la base de cas persistante sous la forme de cas concrets. Le suivi du cycle de vie des cas est ainsi facilité. Toutefois, la structuration des bases de cas n'est pas restreinte à cet exemple, et le cadre de composition permet de définir d'autres structurations.

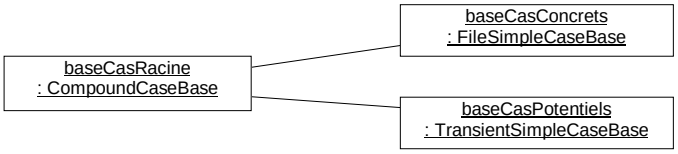


Figure IV-41 : structuration typique des bases de cas concrets et potentiels

Pour l'organisation de l'indexation, nous enrichissons tout d'abord la hiérarchie d'index avec un index spécifique pour la recherche des cas potentiels (classe PotentialCaseKnnIndex). Cet index (héritant de KnnIndex) permet d'effectuer une recherche par plus proches voisins en parcourant un ensemble d'enregistrements et de patrons de cas potentiels pour lesquels il classe les meilleurs cas

suivant leur restriction (utilisation d'un filtre d'indices pour construire dynamiquement la restriction). Les meilleurs cas potentiels sont alors retournés complètement instanciés.

La Figure IV-42 donne un exemple typique d'utilisation de cet index. Tout d'abord, l'index templateIndex recherche les patrons applicables pour le cas cible courant, le résultat est alors sauvé dans une variable nommée Template. L'index d'embranchement (nommé a) lance également la recherche sur la partie instantanée de la situation limitée au contexte des enregistrements (index recordIndex). Ce résultat est alors transmis à l'index chargé d'instancier les patrons (index p). Cet index interprète les identificateurs de sous-ensemble du résultat courant comme des identificateurs d'enregistrement. De même, le résultat de la variable Template est récupéré pour identifier les patrons à utiliser. Un enregistrement ou un patron représente en effet un sous-ensemble de cas en compréhension. Enfin, les meilleurs cas sont retournés avec un résultat de type {CaseSimIndexResult}. Cette modélisation est très flexible car elle permet d'opérer le filtrage des patrons et des enregistrements avec différents index simples ou même composés.

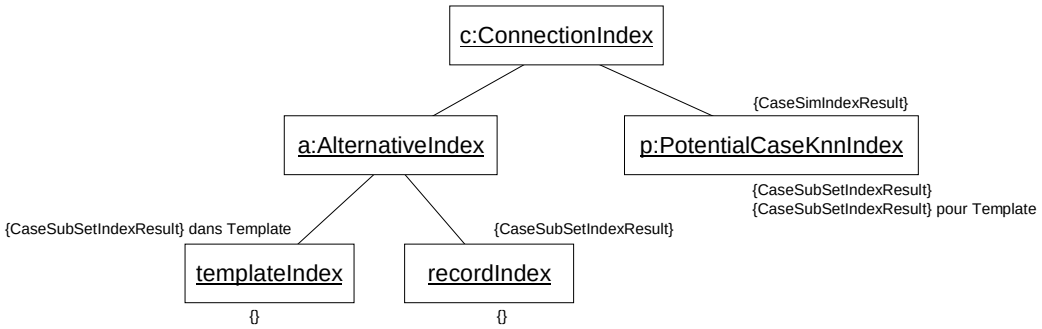


Figure IV-42 : structuration typique des index pour l'instanciation des patrons de cas potentiels

Nous introduisons également l'index BehaviouralKnnIndex qui effectue une recherche par plus proches voisins, tout en construisant la composante comportementale qui représente l'interprétation du cas cible par chaque cas (concret ou potentiel). Cet index possède le profil de paramètre {CaseSetIndexResult} et retourne un type de résultat spécifique {BehaviouralIndexResult} permettant d'accéder à cette interprétation pour chaque cas retourné. Pour permettre l'utilisation de ce type de résultat lors d'une phase de réutilisation, une nouvelle interface de sortie de la phase de recherche est également déclarée (classe BehaviouralRetrieveResult).

3 Réalisation et utilisation de CBR*Tools

Les points d'ouverture et les modèles à objets que nous avons présentés sont implantés dans notre plate-forme à objets nommée CBR*Tools. Cette plate-forme regroupe plus de 200 classes programmées dans le langage à objets Java. Nous présentons tout d'abord quelques points importants de la réalisation puis nous décrivons les principes d'utilisation de la plate-forme.

3.1 Réalisation de la plate-forme

La réalisation de CBR*Tools repose tout d'abord sur une structuration en sous-systèmes qui garde la décomposition en axes de variabilité pour faciliter la recherche des classes à utiliser. De plus, l'implantation en Java a été particulièrement étudiée pour permettre une utilisation aisée.

3.1.1 Structuration en sous-systèmes

Les classes de CBR*Tools sont regroupées en deux sous-systèmes principaux : le sous-système core ou *noyau* qui fournit les abstractions et les composants pour le RàPC de manière générale, et le sous-système time qui regroupe les classes spécifiques pour l'indexation par situations comportementales (cf. Figure IV-43).

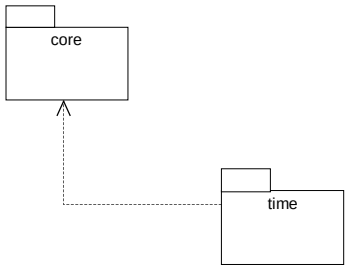


Figure IV-43 : sous-systèmes principaux de CBR*Tools

Le sous-système time comprend 37 classes et le noyau de CBR*Tools est composé de trois sous-systèmes spécifiques, permettant d'organiser les différentes classes suivant les axes de variabilité (cf. Figure IV-44) :

- reasoner (40 classes) : ce sous-système regroupe les classes d'objets de contrôle du raisonnement et les schémas de phase.
- cbrcase (28 classes) : ce sous-système contient les classes responsables de la représentation des cas : indices, types d'indices et vecteurs de poids.
- memory (109 classes) : enfin, ce sous-système, qui permet l'organisation de la mémoire, est lui-même décomposé en trois sous-systèmes différents, en plus des objets de façade contenus à son niveau :
  - casebase (16 classes) : classes pour les bases de cas,
  - index (40 classes) : classes gérant les index et les bases d'index,
  - similarity (50 classes) : classes nécessaires à la représentation des mesures de similarité : objets de contrôle, paramètres, domaines de valeurs, fonctions élémentaires de similarité et fonctions d'agrégation.

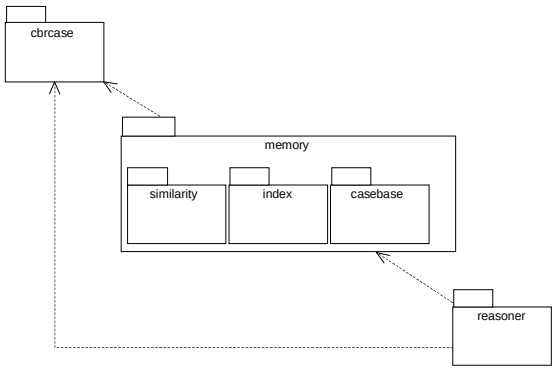


Figure IV-44 : organisation du sous-système core

3.1.2 Implantation en Java

L'implantation générale de CBR*Tools suit une méthodologie rigoureuse sachant que les classes offertes ont la possibilité d'être spécialisées et réutilisées : mécanismes d'assertion et d'exception pour détecter le plus tôt possible la violation de propriétés, conventions de nommage et

d'implantation. Par exemple, chaque point d'ouverture présenté au chapitre précédent est représenté par une hiérarchie de classes en trois niveaux :

- *interface* définissant les opérations publiques du point d'ouverture sans aucun code sous-jacent,
- *classes abstraites* proposant une implantation par défaut des parties constantes du point d'ouverture ainsi que des protocoles internes,
- *classes concrètes* réalisant des variations possibles et directement réutilisables du point d'ouverture.

Ainsi, il est possible de redéfinir complètement le comportement du point d'ouverture à partir de son interface. Les classes abstraites organisées en une hiérarchie sont alors des outils pour construire les variations concrètes du point d'ouverture. La distinction entre l'interface et les classes abstraites des points d'ouverture a été masquée dans les diagrammes de classes présentés dans ce chapitre, puisqu'elle n'apporte pas d'information pertinente au niveau conceptuel.

De plus, la représentation des indices a été particulièrement étudiée pour son intégration dans le langage Java. Ainsi, le modèle conceptuel présenté dans la Figure IV-15 a été adapté pour la réalisation. Les classes Indice et SimpleIndice sont avantageusement remplacées par la classe Object (la classe mère de toutes les classes en Java) permettant ainsi d'utiliser n'importe quelle classe d'objets pour des indices simples. Enfin, une classe abstraite (JavaClassIndice) héritant de l'interface d'un indice composé (CompoundIndice) fournit l'ensemble des opérations permettant d'utiliser une classe Java comme un indice composé. Ainsi, suivant la déclaration des indices effectuée, un attribut public ou un attribut encapsulé respectant les conventions de nommage des Java Beans (méthodes d'accès sous la forme get<nom attribut> et set<nom attribut> ; Beans, 1997), peut être reconnu comme un indice. Les attributs encapsulés permettent de plus la représentation d'indices calculés dynamiquement. La classe JavaClassIndice réalise ces opérations grâce aux fonctions de réflexivité de Java (sorte de métaniveau où les classes sont considérées comme des objets). Nous permettons également la spécialisation des classes d'indices ainsi que l'utilisation des types de base (entier et réel) tout en évitant la création d'objets pour les encapsuler. La classe suivante donne un exemple typique de déclaration d'indices dans le langage Java grâce à CBR*Tools :

```
public class CarCaseSituation extends JavaClassIndice {
   public String make;
   public float width;
   public float length;


   float getSurface() {return width*length;}


   static {
        ArrayIndiceDescriptor desc =
                    new ArrayIndiceDescriptor("aid.cbr.samples.cars.CarCaseSituation");
        desc.addIndiceDescriptor(new IndiceDescriptor("make", StringType.STRING));
        desc.addIndiceDescriptor(new IndiceDescriptor("width", new RangeFloatType(60,73)));
        desc.addIndiceDescriptor(new IndiceDescriptor("length", new RangeFloatType(141,209)));
        desc.addIndiceDescriptor(new IndiceDescriptor("surface",new RangeFloatType(8460,15257)));
   }
}
```

Pour finir, l'expression des indices dans une classe java étant considérée par CBR*Tools comme un cas particulier, d'autres classes abstraites permettent de représenter les indices de différentes manières : structure dynamique (table de hachage, tableau) et séquence d'une chronique (classe RelevantSequence).

Toutefois, nous pouvons regretter que Java n'offre pas en standard des moyens de représentation à objets des connaissances. Le niveau de réflexivité que nous avons utilisé est très limité, et ne permet

pas une personnalisation fine de la description des classes comme dans un langage muni d'un métaniveau (Jaczynski, 1993). La modélisation que nous proposons représente un bon compromis, mais nous laissons en perspective l'ajout de composants pour étendre et gérer la cohérence de la description des indices et de leurs facettes (cf. le système ROCADE, Fuchs, 1997).

3.2 Utilisation de CBR*Tools

L'utilisation de notre plate-forme est facilitée par son intégration au sein d'un environnement graphique permettant de visualiser et de manipuler directement les diagrammes de classes. Toutefois, le raisonnement à partir de cas est intrinsèquement un processus complexe, et nous avons dû introduire de nombreux points d'ouverture pour sa modélisation. Nous donnons alors deux critères d'estimation de la complexité d'utilisation de notre plate-forme pour réaliser un système de RàPC : l'effort minimum et les dépendances systématiques entre points d'ouverture.

3.2.1 Environnement d'utilisation

La plate-forme CBR*Tools peut bien évidemment être directement utilisée comme tout autre ensemble de classes en Java grâce à un éditeur ou à un environnement de programmation classique. Toutefois, nous proposons son utilisation au sein d'un atelier de génie logiciel permettant de visualiser et de manipuler directement l'intégralité des modèles UML présentés.

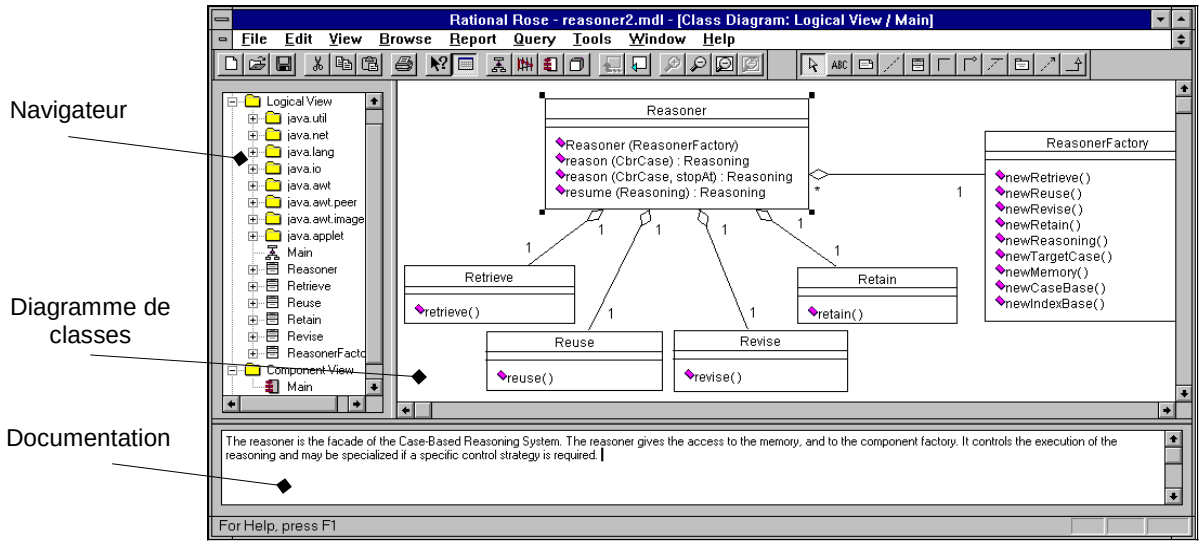


Figure IV-45 : interface d'utilisation de CBR*Tools avec Rational Rose

Durant la conception de CBR*Tools, nous avons en effet utilisé l'atelier de génie logiciel Rational Rose (Rose, 1998 ; Quatrani, 1998). Cet atelier offre une interface graphique (cf. Figure IV-45) permettant de naviguer dans les différents éléments conceptuels (classes, paquetages, scénarios), d'afficher ces éléments sous forme de diagrammes exprimés en UML, et de consulter les documentations associées à chaque élément (classe, attribut, méthode).

L'utilisation d'un tel outil ne s'arrête pas à la conception initiale de CBR*Tools mais a aussi pour but de fournir un environnement de conception permettant de faciliter l'utilisation de CBR*Tools. En effet, cet outil est complémentaire de notre approche reposant sur le concept de plate-forme à objets :

- *Il facilite la compréhension de la plate-forme*. Cet outil permet effectivement de manipuler visuellement les différents concepts de CBR*Tools ainsi que leurs interactions. Ces manipulations mènent non seulement à une meilleure compréhension de CBR*Tools mais

aussi à une meilleure conception par la mise en évidence des points difficiles le plus tôt possible dans le cycle de développement.

- *Il facilite l'extension de la plate-forme*. La spécialisation d'une classe de CBR*Tools se fait alors par simples manipulations graphiques : définition d'une nouvelle classe, déclaration de l'héritage, et déclaration des méthodes à spécialiser par un simple mécanisme de copier/coller.
- *Il permet de générer une documentation de qualité*. La documentation est importante pour une plate-forme, et cet outil permet d'exporter l'ensemble des diagrammes qui peuvent alors être inclus dans la documentation utilisateur.
- *Il facilite l'implantation*. A partir des diagrammes, les structures des classes sont automatiquement générées dans le langage choisi (Java) et doivent être complétées manuellement. Cette génération automatique évite donc un travail fastidieux qui est une source d'erreurs, pour pouvoir se concentrer sur la réalisation des algorithmes.
- *Il facilite le développement itératif*. Cet outil est également pourvu d'un module de ré-ingénierie (*reverse engineering*) particulièrement utile dans un processus itératif de conception et d'utilisation comme celui d'une plate-forme à objets. En effet, le mécanisme de génération et celui de la ré-ingénierie représentent les seuls moyens d'assurer la cohérence entre les diagrammes et la réalisation effective.

3.2.2 Estimation de la complexité d'utilisation de CBR*Tools

Dans cette section, nous donnons une estimation de la complexité d'utilisation de notre plate-forme. Tout d'abord, nous estimons l'effort minimum à réaliser pour construire des applications simples. Puis nous donnons les dépendances systématiques entre points d'ouverture qui sont dictées par la plate-forme.

3.2.2.1 *Effort minimum*

L'effort minimum représente les points d'ouverture à *spécialiser obligatoirement*, les autres points d'ouverture étant munis d'une valeur par défaut ou d'un ensemble de composants directement réutilisables par instanciation. Nous distinguons également les spécialisations guidées par l'application et les spécialisations nécessaires pour déclarer et construire le système avec la plate-forme. Pour le niveau « RàPC », un nouveau système doit obligatoirement définir les phases du raisonnement (recherche, réutilisation et apprentissage, la phase de révision étant souvent absente), la représentation des cas et des indices, et enfin une stratégie d'indexation qui utilise généralement une similarité (cf. Tableau IV-4). Il s'agit enfin de définir la fabrique du système de raisonnement qui permet de déclarer les composants à utiliser (ReasonerFactory). L'application donnée en exemple dans l'annexe C illustre cet effort minimum. Le système de raisonnement ainsi construit configure les 8 points d'ouverture obligatoires, tout en définissant une base de cas spécifique (la base de cas volatile par défaut n'étant pas adaptée).

	Axe « gestion du raisonnement »	*Axe « représentation des cas »*	*Axe « organisation de la mémoire »*	*Total*
Points d'ouverture orientés application	Retrieve, Reuse, Retain	CbrCase, Compoundndice	IndexBase, Similarity	*7*
Points d'ouverture orientés plate-forme	ReasonerFactory	-	-	*1*
Total	*4*	*2*	*2*	*8*

Tableau IV-4 : points d'ouverture à spécialiser obligatoirement pour une application au niveau « RàPC »

Pour une application utilisant l'indexation par situations comportementales, 14 points d'ouverture sont à spécialiser obligatoirement dont uniquement 5 sont nouveaux et spécifiques à ce type d'indexation. Il s'agit en effet de définir les enregistrements, la structure de la composante instantanée, les types de comportements élémentaires, au moins un patron de cas potentiels, un filtre d'indices pour construire la restriction et la déclaration du patron dans la base des patrons (cf. Tableau IV-5).

	Axe « gestion du raisonnement »	***Axe « représentation des cas »***	***Axe « organisation de la mémoire »***	***Total***
Points d'ouverture orientés application	Retrieve, Reuse, Retain	TimeExtendedCase, CompoundIndice, *BehaviouralSituation*, *RelevantSequence*, *Record*, *PotentielCaseTemplate*	IndexBase, Similarity, IndiceFilter, *TemplateBase*	*13*
Points d'ouverture orientés plate-forme	TimeExtendedReasonerFactory	-	-	*1*
Total	*4*	*6*	*4*	*14*

Tableau IV-5 : points d'ouverture à configurer obligatoirement pour une application au niveau « Indexation par situations comportementales »

3.2.2.2 *Dépendances systématiques entre les points d'ouverture*

Pour faciliter l'utilisation de la plate-forme et évaluer la complexité des relations entre les points d'ouverture de CBR*Tools, nous donnons la liste exhaustive des dépendances systématiques entre ces derniers qui sont imposées par la plate-forme. Ces dépendances sont de deux sortes :

- *dépendance déclarative.* La configuration d'un point d'ouverture (spécialisation ou instanciation) se traduit par une déclaration au sein d'un ou plusieurs points d'ouverture. Cette déclaration peut se faire par une simple affectation de paramètre ou par une spécialisation triviale. En d'autres termes, *si A dépend déclarativement de B, l'utilisation d'un nouveau type de A implique une déclaration dans B*. Ce type de dépendance est représenté par une flèche en pointillé de A vers B.
- *dépendance algorithmique*. La configuration d'un point d'ouverture entraîne une modification algorithmique d'un ou plusieurs autres points d'ouverture. Dans ce cas, il est nécessaire de spécialiser ces points d'ouverture. En d'autres termes, *si A dépend algorithmiquement de B, l'utilisation d'un nouveau type de A implique la programmation de B*. Ce type de dépendance est représenté par une flèche pleine de A vers B.

L'axe de gestion du raisonnement comprend le point d'ouverture ReasonerFactory qui est remplacé par TimeExtendedReasonerFactory pour l'indexation par situations comportementales. Ce point d'ouverture est central pour la déclaration des types d'objets à utiliser dans un système de raisonnement. Ainsi de nombreuses relations de dépendance pointent vers la fabrique (cf. Figure IV-46).

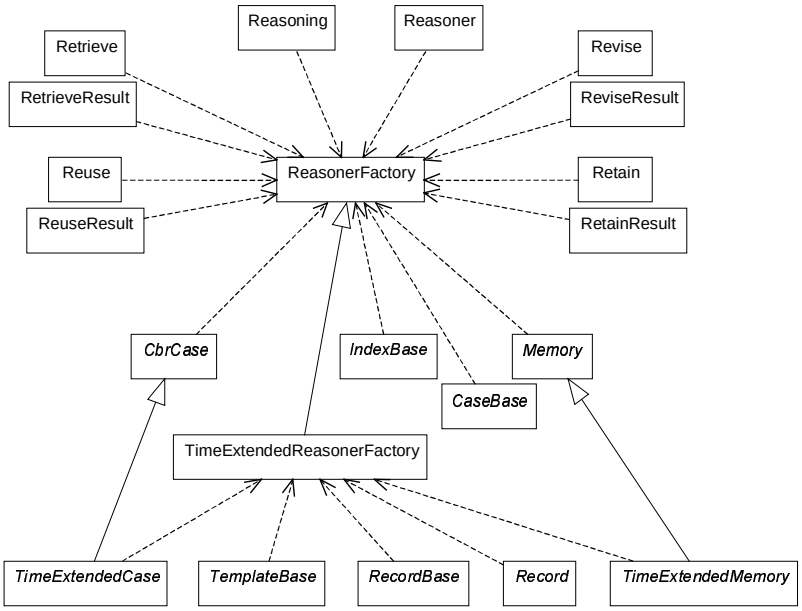


Figure IV-46 : dépendances de l'axe « gestion du raisonnement »

Pour l'axe de représentation des cas, la définition du patron de cas potentiels est un point d'ouverture important. Un patron de cas potentiels doit en effet utiliser l'ensemble des types d'objets servant à construire un cas (cf. Figure IV-47).

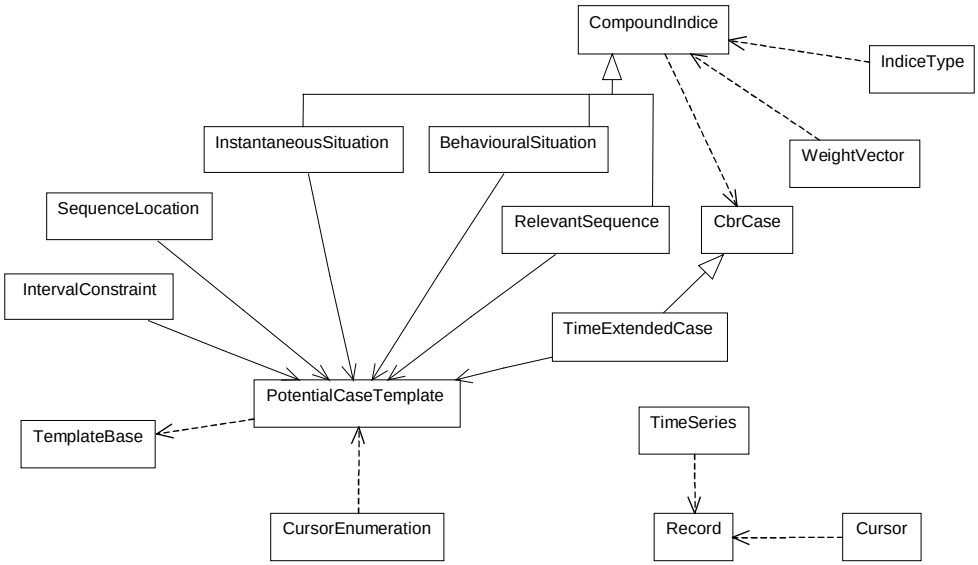


Figure IV-47 : dépendances de l'axe « représentation des cas »

Enfin, l'axe de l'organisation de la mémoire comprend surtout des dépendances déclaratives (cf. Figure IV-48). Toutefois, la définition d'un domaine de valeurs pour les similarités nécessite que des fonctions élémentaires de similarité ainsi que des fonctions d'agrégation soient définies sur ce domaine.

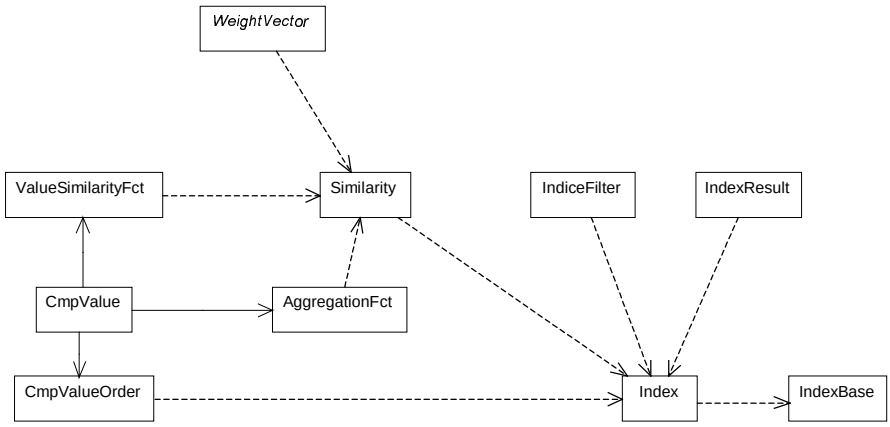


Figure IV-48 : dépendances de l'axe « organisation de la mémoire »

Nous concluons que la configuration d'un point d'ouverture entraîne peu de dépendances systématiques, surtout grâce à la centralisation de la plupart des déclarations dans la fabrique. Toutefois, ces dépendances ne prennent en compte qu'une partie des relations sémantiques qui peuvent exister entre les points d'ouverture. Ces relations, fortement dépendantes de l'application visée, peuvent devenir très complexes. Par exemple, l'introduction d'un nouveau contrôleur de raisonnement peut nécessiter la modification des types de résultats des phases. Pour produire ces résultats, les phases elles-mêmes sont susceptibles d'être modifiées ainsi que l'organisation de l'indexation de la mémoire et la représentation des cas. La documentation de telles dépendances dépasse le cadre de notre thèse, et nécessiterait une méthodologie de conception de systèmes RàPC couplée avec une documentation de haut niveau sur la plate-forme elle-même, par exemple sous la forme de scénarios d'utilisation (Johnson, 1992 ; Meusel *et al*., 1997 ; Froedlish *et al.*, 1997).

4 Conclusion

Suite à la présentation des points d'ouverture nécessaires au RàPC et à l'indexation par situations comportementales, nous avons proposé dans ce chapitre un ensemble de modèles à objets ouverts pour la conception d'une plate-forme en RàPC. Notre démarche originale tient principalement dans l'explication des décisions de conception en termes de patrons. Dans le domaine du génie logiciel, nous avons ainsi montré concrètement comment des opérations complexes, en l'occurrence un raisonnement, peuvent être modélisées avec des patrons, la difficulté majeure étant d'identifier les patrons à appliquer pour atteindre les objectifs de flexibilité.

Dans le domaine du RàPC, notre utilisation des patrons de conception présente deux apports importants. Premièrement, grâce au langage abstrait formé par les patrons et leur instanciation dans un contexte de RàPC, nous avons mis en évidence des objets-métier (schémas de phase, index, similarités, bases de cas) ainsi que leurs façons de collaborer. Les modèles obtenus sont ainsi réutilisables au niveau conceptuel pour guider la réalisation d'applications RàPC de manière générale.

Deuxièmement, ces modèles ont mené à la réalisation concrète d'un outil pour le raisonnement à partir de cas et plus particulièrement pour la gestion de l'indexation par situations comportementales. Cet outil, sous la forme d'une plate-forme à objets nommée CBR*Tools, offre un ensemble de composants réutilisables au sein de l'architecture abstraite modélisée. L'utilisation de la plate-forme est également facilitée par son intégration dans un atelier graphique où les diagrammes de classes sont accessibles et peuvent être spécialisés. Cet outil permet enfin la capitalisation d'objets-métier

notamment sous les formes d'index, de fonctions de similarité et surtout de schémas de phase de raisonnement.

CBR*Tools est implanté en Java et représente plus de 200 classes. Nous avons premièrement donné une estimation *a priori* de la complexité de son utilisation (effort minimum et dépendances systématiques). Deuxièmement, différentes applications ont été implantées avec CBR*Tools. Tout d'abord, des systèmes simples de raisonnement ont été réalisés. Le calcul d'un facteur de risque pour un assureur de voitures a nécessité la spécialisation d'uniquement dix classes (cf. annexe C). Nous avons également réalisé une application de recherche des cas similaires pour l'assistance à la conduite de la nutrition des plantes et une application d'aide à la navigation sur le Web (Broadway) que nous présentons dans le chapitre suivant. Cette dernière application nous a permis d'évaluer concrètement notre plate-forme (cf. chapitre V, §2.2 et §2.3.2)

Chapitre V : application pour l'assistance à la navigation sur le Web

Résumé

Pour évaluer notre modèle d'indexation et notre plate-forme à objets CBR*Tools, nous avons réalisé un assistant de navigation sur le Web appelé Broadway. Cet assistant utilise le raisonnement à partir de cas pour recommander des pages à visiter suivant le comportement d'un utilisateur courant. Broadway est accessible par un groupe d'utilisateurs et permet ainsi une collaboration indirecte en réutilisant les cas issus des navigations du groupe. Les navigations des utilisateurs sont enregistrées suivant quatre variables décrivant l'adresse, le contenu, l'évaluation explicite et le ratio d'affichage de chaque page Web visitée. Ces navigations sont utilisées pour en extraire des expériences utiles (cas). Un cas permet d'associer à un comportement (succession de pages), un ensemble de pages évaluées. Nous montrons ainsi que notre modèle d'indexation permet la modélisation de ce type de cas. Pour évaluer l'utilisation de CBR*Tools, nous montrons premièrement que, pour la réalisation de Broadway, l'expertise approfondie nécessaire est limitée à la moitié des points d'ouverture. Deuxièmement, nous montrons également l'aide apportée par CBR*Tools tant sur la modélisation que sur l'implantation, grâce à la réutilisation de son architecture abstraite et de ses composants (index, similarité). De plus, avec Broadway, nous proposons un *nouveau type d'assistants* dans lesquels les comportements des utilisateurs sont observés suivant un ensemble extensible de variables. Cette observation par variables, combinée avec notre modèle d'indexation, permet la gestion de comportements plus détaillés dans un cadre flexible et générique. Une première évaluation expérimentale de notre assistant montre l'intérêt et la faisabilité de notre approche.

Ce dernier chapitre présente l'application de notre modèle d'indexation par situations comportementales (cf. chapitre II) et l'utilisation de notre plate-forme à objets CBR*Tools (cf. chapitres III et IV) pour le système Broadway[42], notre assistant à la navigation sur le Web.

Le World-Wide-Web est un gigantesque hypermédia dans lequel trouver un document pertinent n'est pas une tâche facile malgré les outils existants pour effectuer une recherche (cf. chapitre I, §1.2). Broadway permet de suivre un groupe d'utilisateurs durant leurs navigations et recommande des pages potentiellement intéressantes en fonction de leurs comportements. Broadway utilise le raisonnement à partir de cas pour réutiliser des expériences précises issues des navigations passées du groupe. Notre approche soutient l'hypothèse suivante : *si deux utilisateurs ont navigué suivant une séquence similaire de pages, c'est qu'ils ont des intérêts similaires, nous pouvons alors proposer au second utilisateur les pages jugées pertinentes par le premier.*

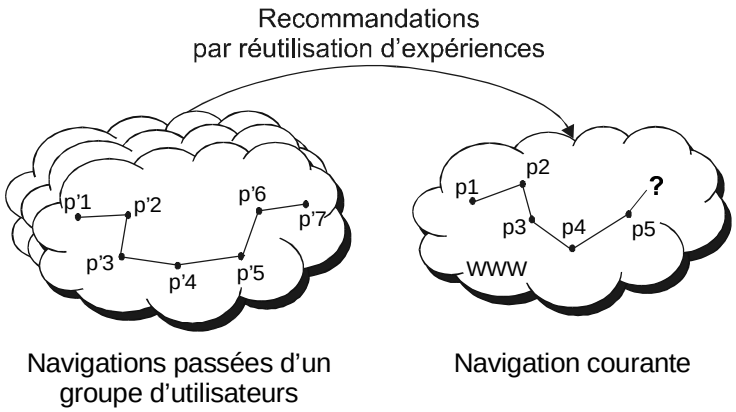


Figure V-1 : réutilisation d'expériences pour déterminer des recommandations

Nous présentons, dans la première partie, les caractéristiques générales du système Broadway : interface utilisateur, architecture et principes du cycle de raisonnement à partir de cas. Dans la deuxième partie, nous détaillons l'application de notre modèle d'indexation et l'utilisation de CBR*Tools dans le cadre de ce système. Nous faisons alors un bilan sur les qualités et les limites du modèle d'indexation ainsi que sur l'aide apportée par CBR*Tools. Enfin, dans la troisième partie, nous montrons que l'approche proposée mène à un nouveau type de systèmes d'assistance, dont Broadway est un exemple.

1 Présentation de Broadway

Broadway est un assistant de navigation répondant à des contraintes d'application précises. Après avoir introduit ces contraintes, nous présentons l'interface graphique et l'architecture répartie de Broadway. Enfin, nous donnons les principes de base du raisonnement mené par ce système pour déterminer les recommandations durant les navigations des utilisateurs sur le Web.

1.1 Contraintes de l'application

Notre objectif est de réaliser un *assistant de navigation*, c'est-à-dire un outil de navigation pour le Web qui soit capable de suivre un utilisateur, tout en essayant de déterminer automatiquement son

[42] Le nom « Broadway » est basé sur l'expression suivante : « *a BROsing ADvisor reusing pathWAYs* ».

intérêt pour lui recommander des pages à visiter. La principale contrainte de Broadway est de pouvoir *prendre en compte les historiques des actions d'utilisateur* pour mieux déterminer les recommandations. Nous pensons en effet, comme d'autres (Corvaisier *et al.,* 1997 ; Shahabi *et al.,* 1997 ; Chen & Park, 1998 ; Schechter *et al.,* 1998), qu'un état particulier de la navigation (la page courante ou une description instantanée de la navigation) n'est pas suffisant pour caractériser le comportement d'un utilisateur. C'est pourquoi nous voulons considérer les pages passées et leur ordre de consultation pour induire son intérêt implicite.

De plus, pour obtenir les recommandations et pour permettre une large utilisation du système, nous abordons cette problématique avec cinq autres exigences :

1. Broadway doit *être conçu pour le Web*, c'est-à-dire que son utilisation ne doit pas être limitée à un site Web et qu'il doit prendre en compte l'aspect dynamique de cet hypermédia dans lequel des pages peuvent être effacées ou modifiées à tout moment.
2. Broadway doit *apprendre d'un groupe d'utilisateurs* afin d'améliorer et de maintenir à jour ses connaissances lui permettant de déterminer les recommandations.
3. Broadway doit permettre *d'associer des annotations textuelles* aux pages pour décrire plus précisément leur pertinence.
4. Broadway doit *être indépendant du navigateur* pour pouvoir être utilisé sur différentes plates-formes.
5. Enfin, Broadway doit *présenter une architecture ouverte* facilitant son extension et sa modification, sachant que des évaluations en milieu réel seront menées et que des réglages seront nécessaires.

1.2 Interface utilisateur et architecture de Broadway

L'assistant de navigation Broadway peut être utilisé par un groupe d'utilisateurs qui veulent partager leurs expériences de navigation sur le Web. Chaque utilisateur a la possibilité d'utiliser son propre navigateur et peut communiquer avec Broadway au travers d'une interface graphique dédiée. Derrière cette interface, Broadway repose sur une architecture répartie pour une meilleure efficacité et pour faciliter son extension.

1.2.1 Interface utilisateur

L'utilisateur est capable d'interagir avec Broadway par deux moyens : les *barres d'outils* et le *contrôleur*. Les barres d'outils sont insérées dans chaque page visualisée par le navigateur et offrent des fonctions contextuelles. Le contrôleur est affiché dans une fenêtre séparée. Il permet à chaque utilisateur de contrôler le fonctionnement global de Broadway et de sélectionner les recommandations.

1.2.1.1 *Barre d'outils*

Une barre d'outils est insérée au début de chaque page Web chargée à partir du protocole de communication HTTP (cf. Figure V-2). Lorsqu'une page est composée de différents cadres (*frames*), une barre d'outils est également insérée en haut de chaque cadre. La barre d'outils est par défaut masquée et peut être affichée entièrement par simple clic sur un onglet dans le coin supérieure gauche de la fenêtre[43]. La barre d'outil permet ainsi d'accéder à des fonctions contextuelles à la page ou au

[43] Cette fonctionnalité n'a été testée que dans le cas du navigateur Netscape en utilisant les possibilités simples qu'il offre pour superposer différentes couches HTML.

cadre dans lequel elle est affichée, même dans le cas où l'utilisateur a besoin de charger des pages simultanément dans des fenêtres différentes.

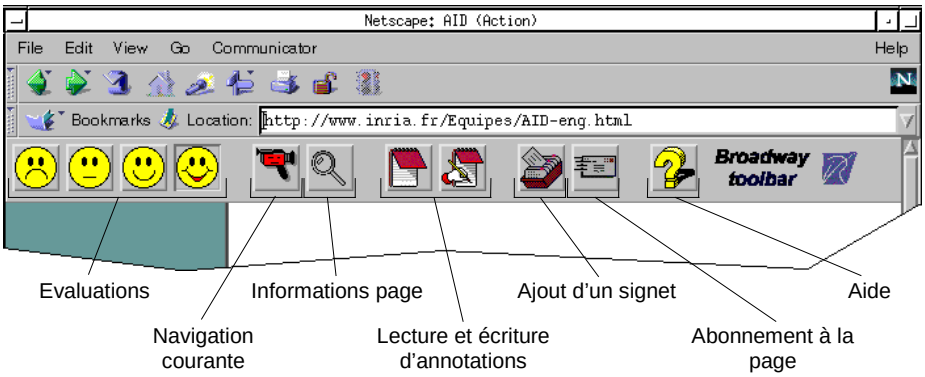


Figure V-2 : barre d'outils de Broadway

Une barre d'outils fournit un ensemble de fonctions : évaluation de la page avec quatre niveaux (pas pertinent, indifférent, pertinent et très pertinent), consultation de la navigation courante et des informations statistiques sur la page courante, écriture et lecture d'annotations textuelles, ajout de la page dans le serveur partagé de signets[44], abonnement à la page[44] (notification par mail de son changement), et enfin accès à une page d'aide pour l'utilisation de la barre d'outils. Les boutons d'évaluation permettent d'indiquer la pertinence de la page courante vis-à-vis de l'objectif d'une navigation particulière. Si l'utilisateur repasse sur une page déjà visitée, la dernière évaluation est affichée par défaut et peut être ajustée. Enfin, lorsqu'une page ayant des annotations est affichée, le bouton de lecture change d'apparence pour l'indiquer directement à l'utilisateur.

1.2.1.2 *Contrôleur*

Le contrôleur permet la configuration de Broadway ainsi que l'affichage des recommandations (cf. Figure V-3). Une navigation peut être lancée ou stoppée et correspond à une tâche de recherche d'une information sur le Web. Durant cette navigation, le calcul des recommandations peut s'effectuer à la demande ou en continu. Dans le premier mode, un bouton permet de lancer une requête. Les recommandations sont alors calculées de manière asynchrone et sont affichées dans une liste dès leur disponibilité. Dans le mode de calcul en continu, les recommandations, également calculées de manière asynchrone, sont régulièrement mises à jour. Ces mises à jour sont déclenchées par les changements de pages dans le navigateur.

Les recommandations sont présentées dans une liste donnant un pourcentage et le titre de la page (si la page n'a pas de titre, son adresse est alors affichée). Le pourcentage représente l'estimation faite par Broadway du degré de pertinence dans une certaine page vis-à-vis du comportement de l'utilisateur. Si l'utilisateur sélectionne une recommandation, la zone d'information sur la recommandation est alors mise à jour pour donner l'adresse, le titre, les mots clefs et la description de la page. Ainsi, il est possible d'analyser rapidement les recommandations pour en choisir une. Le bouton « *go* » permet finalement de se déplacer directement à la page sélectionnée qui est affichée dans le navigateur.

[44] Cette fonction n'as pas été implantée mais montre les possibilités d'extension de Broadway en cours d'étude.

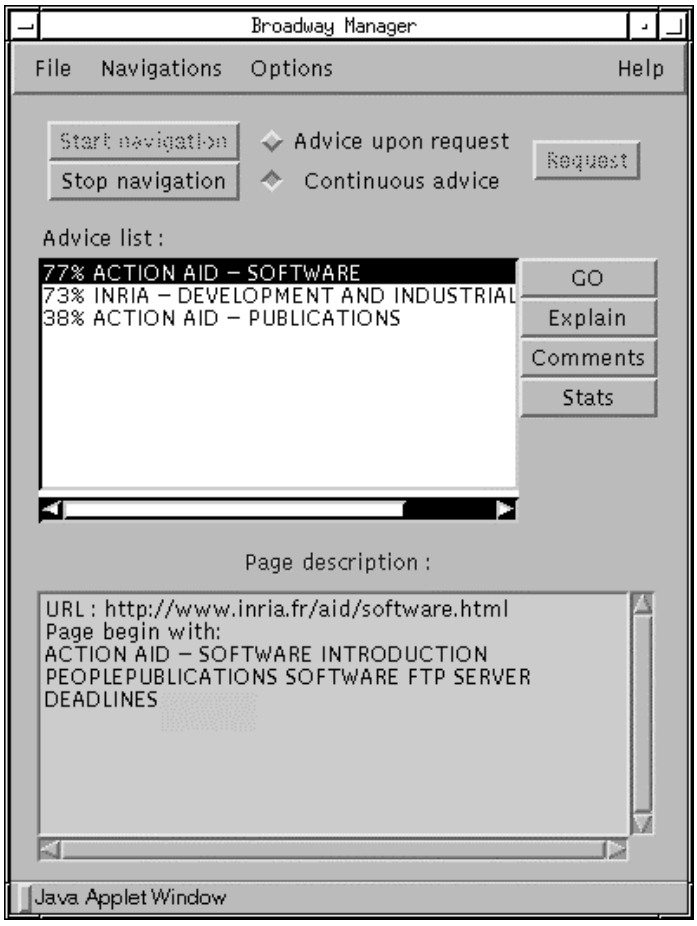


Figure V-3 : contrôleur de Broadway

1.2.2 Architecture logicielle

Nous introduisons tout d'abord l'architecture globale de Broadway qui repose sur la notion de serveur HTTP proxy. Puis nous présentons plus précisément l'architecture répartie mise en place pour une meilleure modularité des fonctionnalités. Enfin, nous décrivons les outils réalisés pour faciliter l'extension et la modification de Broadway.

1.2.2.1 *Architecture globale*

En se basant sur l'analyse donnée dans (Meeks *et al.*, 1996), il existe deux types d'architecture pour concevoir un assistant de navigation pour le Web : des transducteurs de flux (*stream transducers*) et des associés de navigation (*browsing associates*). Un transducteur est un module inséré dans les communications HTTP entre un client et le reste du Web pour analyser et modifier les flux de requêtes et de réponses. Un associé de navigation est un module autonome qui accède au Web de manière indépendante pour la réalisation d'une tâche. Notre objectif est de suivre les utilisateurs durant leurs navigations et de les assister en fonction de leurs actions. Ce type d'assistant est très dépendant du flux de requêtes et de réponses, c'est pourquoi nous avons choisi une architecture de type transducteur pour Broadway.

Plus précisément, Broadway repose sur la notion de serveur HTTP *proxy* (ou tout simplement proxy) qui est un des moyens pour réaliser un transducteur de flux. Un proxy est un serveur HTTP spécial accessible par un groupe d'utilisateurs. Un proxy attend les requêtes des clients et les transfère au besoin vers le reste du Web ; puis les réponses provenant du Web sont alors retournées vers le

client initial. Les premières utilisations des proxies étaient limitées à des fonctions de sécurité ou de mémoire cache (Luotonen & Altis, 1994). Dans le premier cas, le proxy, appelé pare-feu (*firewall*), intercepte et vérifie les communications HTTP entre le réseau intérieur ainsi sécurisé, et le réseau extérieur. Dans le deuxième cas, le proxy gère une mémoire cache des pages déjà visitées de sorte que, si un utilisateur redemande la même page, le proxy renvoie sa copie locale (si elle n'est pas obsolète) sans recharger la page à partir du serveur d'origine. Ces deux usages du proxy peuvent être combinés mais restent de bas niveau. Dans ce cadre, l'utilisation de proxies est répandue sur le Web et fait partie du protocole HTTP. Tous les navigateurs courants (par exemple, Internet Explorer et Netscape) permettent leur configuration pour utiliser de manière transparente un proxy.

Toutefois, l'intérêt de ce type d'architecture dépasse ces fonctions de bas niveau et mène à la définition de *proxies d'applications*. En effet, un proxy est un outil adapté à l'analyse transparente des communications HTTP et du contenu des pages Web visitées. De plus, le proxy est capable de modifier le contenu d'une page avant que l'utilisateur soit en mesure de l'afficher. Ces propriétés ont été étudiées dans de nombreux travaux (Brooks *et al.*, 1995 ; Luotonen & Altis, 1994) et ont mené à l'utilisation de proxies pour des applications de haut niveau comme la gestion d'un serveur d'annotations pour un groupe d'utilisateurs (Schicker *et al.*, 1996). Broadway est dans la même lignée que ces travaux. D'autres assistants de navigation ont d'ailleurs également choisi ce type d'architecture (Mladenic, 1996).

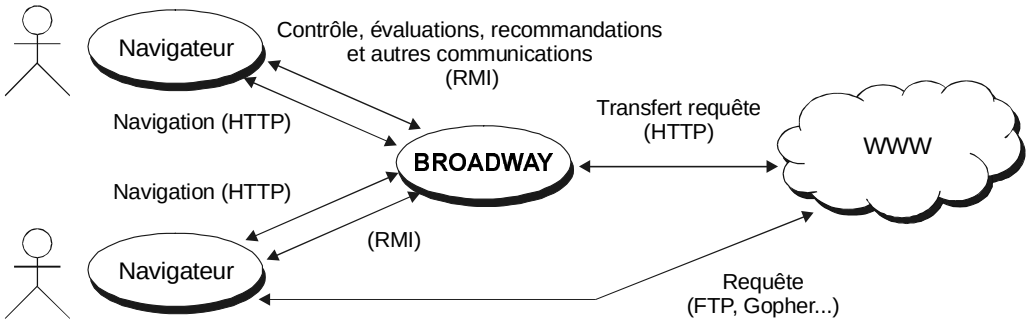


Figure V-4 : architecture globale de Broadway

Ainsi, Broadway est un système qui est intercalé entre les navigateurs et le reste du Web pour les requêtes utilisant le protocole HTTP (cf. Figure V-4). Broadway accepte de multiples connexions simultanées provenant d'un groupe d'utilisateurs. Chaque utilisateur est identifié par un nom et un mot de passe suivant le protocole d'authentification prévu dans HTTP. Au lancement d'un navigateur, une fenêtre est alors affichée à l'écran pour que l'utilisateur puisse s'identifier. Ces données sont alors stockées par le navigateur qui les transmettra avec chaque requête. Cette authentification est importante car elle permet de contrôler l'accès au système qui contient des informations privées, notamment sous la forme de navigations. De plus, ce mécanisme permet de suivre un utilisateur dans la succession de ces requêtes. En effet, HTTP est un protocole de communication *sans état*, et le proxy n'a pas de moyen par défaut pour savoir s'il traite des requêtes issues du même utilisateur ou d'utilisateurs différents. La transmission automatique et transparente des informations d'identification avec chaque requête permet alors de pallier ce problème[45].

[45] Pour ce dernier problème, une solution souvent utilisée est celle d'un *cookie* (RFC 2109, 1997). Un cookie est une information que le serveur HTTP demande de sauver dans le navigateur de l'utilisateur et qu'il peut consulter à chaque requête. Cependant, les cookies sont accessibles uniquement par les serveurs du domaine qui les a déclarés (un proxy n'est pas censé les manipuler) et les utilisateurs peuvent refuser ou effacer les cookies. D'autres solutions existent ou sont en cours d'étude : solution coûteuse de la réécriture des hyperliens (Iyengar, 1997), extension du protocole HTTP (Hallam-Baker & Connolly, 1996).

Broadway intercepte toutes les requêtes dans le protocole HTTP et permet ainsi d'analyser les différentes pages HTML chargées par les utilisateurs. Les requêtes effectuées dans d'autres protocoles (FTP ou Gopher par exemple) sont exécutées directement ou avec d'autres mécanismes indépendants de Broadway. De plus, Broadway est muni d'une mémoire cache ce qui lui donne une plus grande efficacité dans le chargement des pages pour le groupe. Enfin, Broadway gère un espace d'information local qui est accessible par le protocole de communication RMI (*Remote Method Invocation*) du langage de programmation Java (RMI, 1997). Ainsi, toutes les communications nécessaires aux barres d'outils et aux contrôleurs sont réalisées suivant ce protocole. Nous avons choisi ce protocole de communication plutôt que HTTP ou CORBA (Geib *et al.*, 1997) pour plusieurs raisons. Tout d'abord, par rapport à HTTP et l'utilisation de pages HTML, RMI permet de récupérer simplement sur les clients les objets représentant les informations requises ; ces objets peuvent alors être traités localement dans le navigateur pour améliorer l'interactivité et la personnalisation de l'affichage[46]. Puis, par rapport à CORBA qui suppose un environnement hétérogène, l'ensemble de nos composants est programmé en Java, et RMI offre des fonctionnalités spécifiques à cet environnement homogène : gestion de la sécurité lors du chargement dynamique de classes distantes et ramasse-miettes réparti par exemple. De plus, dans cet environnement, RMI est disponible en standard et reste simple d'emploi.

1.2.2.2 *Architecture détaillée de Broadway*

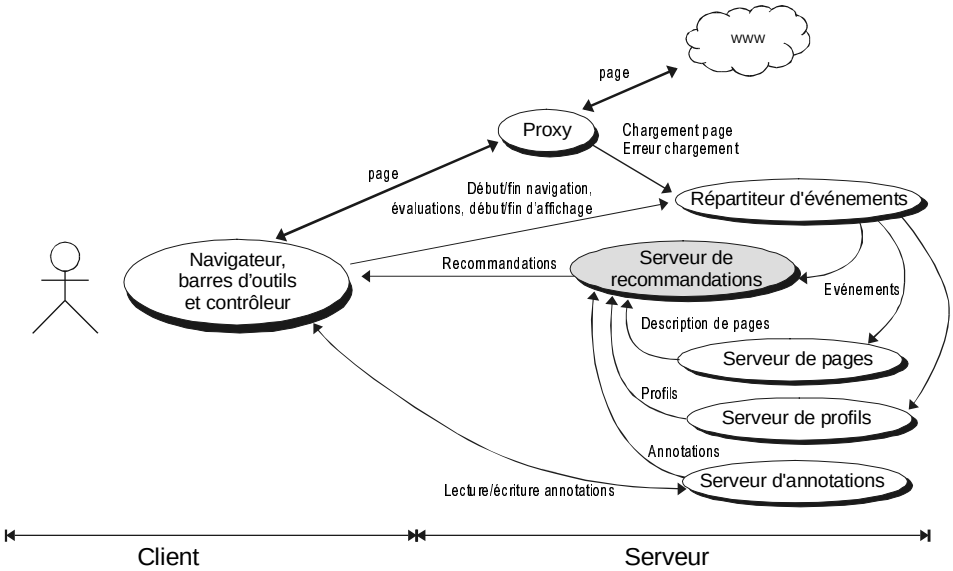


Figure V-5 : architecture détaillée de Broadway

L'architecture détaillée de Broadway présente un ensemble de modules réalisés dans le langage de programmation Java et dont la plupart peut être exécutée dans des processus différents (cf. Figure V-5). Du coté client, le navigateur de l'utilisateur abrite différents modules : les barres d'outils, le contrôleur et l'observateur. Ces modules sont des *applets*, c'est-à-dire des programmes Java qui s'exécutent au sein du navigateur dans des processus légers (*threads*) différents. Nous avons déjà décrit les rôles des barres d'outils et du contrôleur au travers de leur interface graphique. L'observateur est une applet ne possédant pas d'interface et qui est insérée comme les barres d'outils dans chaque page HTML par Broadway. Cette applet est alors activée lors du début et de la fin de l'affichage d'une page et en informe Broadway, même dans le cas où la page est chargée à partir de la mémoire cache locale du navigateur d'un utilisateur (Shahabi *et al.*, 1997).

[46] L'utilisation de RMI n'est pas obligatoire, la première version de Broadway n'utilisait d'ailleurs que le protocole HTTP.

Du côté du serveur, Broadway est composé de six modules exécutés comme des processus distincts : le répartiteur d'événements, le proxy, le serveur de descriptions de pages, le serveur de profils, le serveur d'annotations, et enfin le serveur de recommandations. Pour les communications entre ces processus, nous utilisons pour les mêmes raisons que précédemment le protocole de communication RMI.

Le répartiteur d'événements a une position centrale dans cette architecture répartie et permet d'abonner un module à certains événements correspondant à des actions des utilisateurs ou du système. Les producteurs d'événements liés aux actions des utilisateurs sont les barres d'outils (les boutons d'évaluation par exemple émettent chacun un événement), les contrôleurs (début et fin d'une navigation par exemple), les observateurs dont l'unique rôle est d'envoyer des événements au début et à la fin de l'affichage d'une page et enfin le proxy qui indique les chargements de page et les erreurs éventuelles. Les consommateurs de ces événements sont les modules d'analyse du comportement de l'utilisateur comme par exemple le serveur de recommandations.

La fonction proxy de Broadway s'appuie sur le serveur HTTP Jigsaw (Baird-Smith *et al.*, 1997) conçu et réalisé par le W3C (World-Wide-Web Consortium). Au-dessus de ce serveur, préalablement configuré en tant que proxy, nous avons personnalisé les deux fonctions essentielles dans le cadre de Broadway : l'identification des utilisateurs lors de leurs requêtes, et le traitement des pages HTML avant leur transfert aux navigateurs des utilisateurs pour l'insertion des barres d'outils et des observateurs. Le proxy informe également le système des chargements de pages et transmet leur contenu au serveur de descriptions de pages qui est chargé d'extraire et de sauvegarder les caractéristiques pertinentes de chaque page visitée par le groupe d'utilisateurs : titre, mots clefs, date d'expiration, dernière date de modification. Le serveur de profils permet de gérer les identificateurs et les propriétés des utilisateurs (nom, prénom, adresse e-mail)[47]. Le serveur d'annotations permet d'associer à une page un ensemble d'annotations textuelles.

Enfin, le serveur de recommandations est la partie la plus complexe du système. Il permet en effet la sauvegarde des informations pertinentes liées aux navigations des utilisateurs et intègre le système de raisonnement à partir de cas capable de calculer les recommandations. Le serveur est abonné aux événements de navigation des utilisateurs, ainsi qu'aux événements de contrôle permettant de spécifier le mode de calcul (en continu ou à la demande). Quel que soit le mode choisi, les demandes de recommandations sont placées dans une file d'attente dans le serveur de recommandations. Un processus léger séparé traite ces demandes en séquence pour effectuer les raisonnements. Si des recommandations sont trouvées, le serveur transmet directement au contrôleur de l'utilisateur concerné les résultats pour les afficher[48]. Les recommandations sont donc calculées de manière asynchrone et sont affichées dès qu'elles sont disponibles.

1.2.2.3 *Broadway*Tools : des composants ouverts et réutilisables pour Broadway*

L'architecture du prototype actuel de Broadway et son fonctionnement reposent sur la spécialisation et l'intégration d'un ensemble de composants ouverts et réutilisables facilitant d'une part l'extension et la modification du système existant Broadway, et d'autre part, l'application de notre approche dans d'autres systèmes. Ces différents composants sont appelés des outils au regard des tâches de réalisation et de maintenance d'un système comme Broadway. L'ensemble de ces outils est dénommé Broadway*Tools et se compose de quatre groupes : les outils d'interface graphique, les

[47] Ce serveur est en cours d'extension pour associer à chaque utilisateur les groupes auxquels il appartient, les sessions auxquelles il participe, ses préférences, et son profil en termes de mots clefs pertinents pour son activité (Elhaddad, 1998).

[48] Techniquement, le serveur ouvre une connexion RMI avec l'applet située dans le navigateur de l'utilisateur. Ce mécanisme demande que des droits particuliers soient établis par l'utilisateur. Si ce n'est pas le cas, un mécanisme de scrutation (*polling*) du contrôleur est automatiquement lancé pour récupérer régulièrement les nouvelles recommandations.

outils pour la réalisation des serveurs, les outils pour la réalisation des services et enfin les outils de persistance.

Pour les outils d'interface graphique, un ensemble de composants est proposé pour construire des barres d'outils destinées à être insérées dans des pages HTML. Nous proposons des classes d'objets facilitant le décodage des paramètres de l'applet ainsi que des boutons prédéfinis.

Pour les serveurs, quatre ensembles de composants permettent de construire respectivement un serveur de recommandations, un serveur d'annotations, un serveur de profils utilisateurs et un serveur de pages. Pour chaque type de serveurs, nous proposons des classes d'objets simplifiant la gestion des requêtes avec le protocole RMI et des classes permettant de personnaliser le traitement effectué dans le serveur (sous la forme d'une librairie de classes ou d'une plate-forme à objets). Par exemple, le serveur de recommandations s'appuie sur notre plate-forme à objets CBR*Tools et le serveur de descriptions de pages utilise une bibliothèque de fonctions pour analyser les pages HTML et effectuer l'extraction des mots clefs.

Les services sont les modules fonctionnant entre les interfaces graphiques et les serveurs : proxy, observateur, répartiteur d'événements. Alors que l'observateur et le répartiteur d'événements sont des outils directement intégrables, l'outil proxy offre des fonctions de personnalisation pour la gestion de l'identification et du prétraitement des pages Web. Le répartiteur d'événements répond à des besoins similaires au service d'événements de CORBA (Geib *et al.*, 1997, pages 152–155), dans la mesure où il permet d'effectuer des communications asynchrones sans connexion directe entre un producteur d'événements et un nombre quelconque de consommateurs. La séparation entre les producteurs d'événements et leurs consommateurs est, dans notre approche, très importante pour répondre aux exigences d'ouverture. En effet, les producteurs peuvent être modifiés durant la maintenance du système sans affecter le fonctionnement du reste de Broadway. Par exemple, les événements de début et de fin de navigation, qui sont générés dans le présent prototype par le contrôleur, pourraient être générés automatiquement par un module d'analyse. De même, de nouveaux consommateurs peuvent être ajoutés au système sans modifier ou notifier les producteurs. Enfin, les consommateurs existants peuvent être facilement maintenus en modifiant les types d'événements auxquels ils sont abonnés. Techniquement, nous utilisons les possibilités offertes par le langage Java pour transmettre des classes d'objets et des objets entre différents processus. Un consommateur doit alors implanter un objet de filtrage d'événements (*listener*) qui est transmis au répartiteur d'événements (cf. Figure V-6). Lorsqu'un nouvel événement doit être traité par le répartiteur, les objets de filtrage correspondants sont activés et transmettent à leur tour uniquement les informations utiles à leur consommateur. Ainsi, l'envoi de messages inutiles entre le répartiteur et les consommateurs est supprimé, et l'abonnement d'un consommateur peut être dynamiquement changé au cours de l'exécution du système.

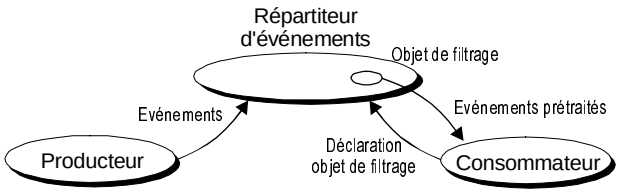


Figure V-6 : abonnement d'un module à des événements

Enfin, les outils de persistance reposent sur le produit ObjectStrore PSE Pro commercialisé par la société Object Design. Ce produit permet de réaliser la persistance d'objets Java dans une base de données, soit localement dans un fichier, soit dans un serveur distant ObjectStore. Un ensemble de classes et des outils sont fournis avec ce produit dont notamment un préprocesseur de code Java automatisant les annotations du code nécessaires à la persistance. Toutefois, des outils supplémentaires ont été réalisés pour faciliter l'utilisation de ObjectStorePSE dans le cadre de

Broadway (Golinelli, 1998)[49] : activation et désactivation de la persistance à la compilation, méthodologie pour rendre un serveur persistant.

1.3 Serveur de recommandations

Bien que Broadway soit accessible simultanément par de multiples utilisateurs, les demandes de recommandations sont ordonnées chronologiquement pour un traitement séquentiel. Ce traitement permet d'éviter les accès et les mises à jour concurrentes de la base de cas[50]. Le serveur de recommandations réutilise des cas qui sont principalement composés de la description du comportement d'un utilisateur et d'une liste de pages recommandées pour ce comportement. Alors que les navigations représentent les données brutes d'observation, les cas permettent de manipuler des associations pertinentes issues des navigations entre un comportement et des pages évaluées :

Cas = (comportement utilisateur issu d'une navigation, liste de pages évaluées et recommandées)

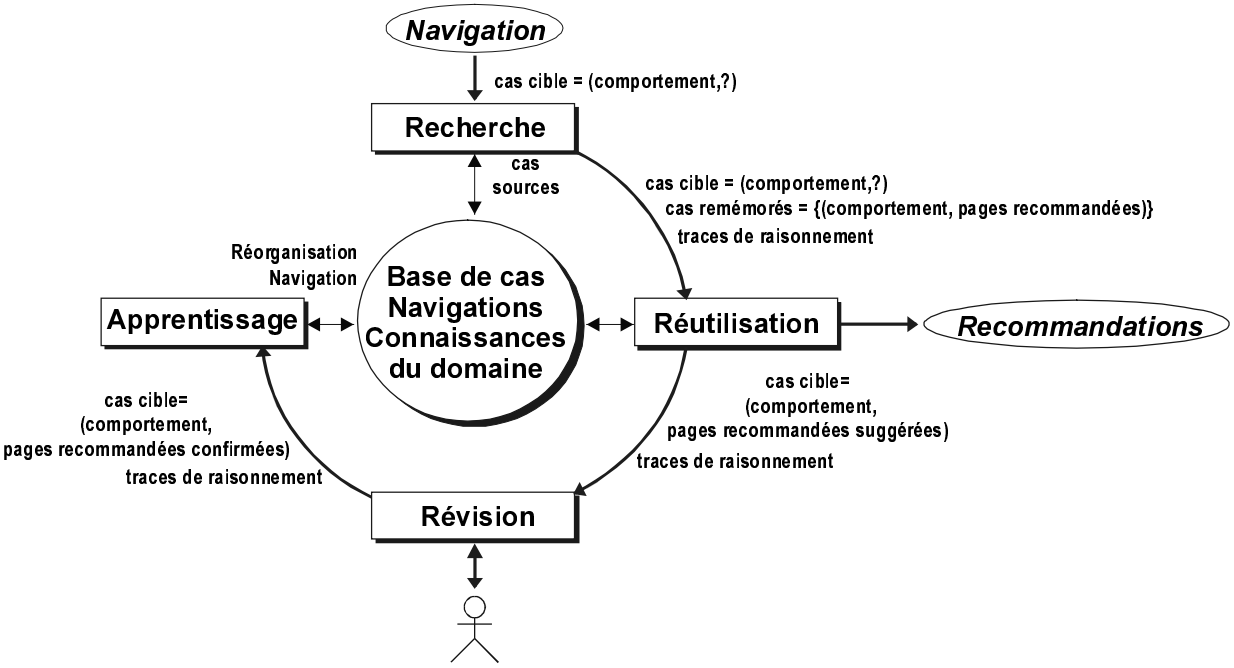


Figure V-7 : cycle du raisonnement dans Broadway

Le calcul des recommandations est alors effectué durant un raisonnement en quatre phases, étant donné la navigation courante d'un utilisateur (cf. Figure V-7) :

1. *Recherche*. La base de cas est parcourue pour identifier les cas dont la partie comportement peut être mis en correspondance avec la navigation courante. Les meilleurs cas sont alors retournés.
2. *Réutilisation*. Chaque cas retrouvé propose une liste de pages recommandées, et la phase de réutilisation opère la fusion de ces pages pour déterminer la liste des recommandations à présenter à l'utilisateur courant (pages recommandées suggérées).
3. *Révision*. La phase de révision est effectuée par l'utilisateur lorsqu'il continue sa navigation et qu'il évalue les pages visitées s'il le désire. La révision se poursuit jusqu'à

[49] La réalisation de ces outils est le résultat du projet de fin d'études (DESS) de M. Gollinelli que nous avons encadré (Golinelli, 1998).
[50] L'étude de la réalisation de raisonnements en parallèle dépasse la cadre de nos travaux.

la fermeture de la navigation courante et met en attente tous les raisonnements effectués pour une navigation (pages recommandées confirmées).

4. *Apprentissage et maintenance de la mémoire.* L'apprentissage prend en compte la navigation courante et l'ensemble des raisonnements associés pour en faire leur synthèse. La mémoire est alors mise à jour par la création de nouveaux cas, par la mise à jour des cas et par l'ajout de la navigation courante. A côté de ce processus de raisonnement, et étant donné que le Web est un espace en continuel changement, les cas et les navigations devenus obsolètes sont régulièrement effacés de la mémoire.

Dans la section suivante, nous détaillons l'application de notre modèle d'indexation par situations comportementales, et en particulier le processus de raisonnement.

2 Evaluation de notre modèle d'indexation et de CBR*Tools

Nous présentons tout d'abord l'application du modèle d'indexation par situations comportementales puis l'utilisation de notre plate-forme CBR*Tools pour la réalisation concrète du prototype Broadway. Enfin, nous faisons le bilan des apports et des limites du modèle d'indexation et de la plate-forme dans le cadre de cette application.

2.1 Application du modèle d'indexation par situations comportementales

L'utilisation de notre modèle d'indexation reposent sur quatre étapes d'ingénierie des connaissances (cf. chapitre II, §2.2) : identification des variables d'observation, détermination de la sémantique d'un enregistrement et de son contexte, définition de la représentation des cas, et enfin conception des phases du raisonnement. Nous suivons alors ces étapes pour montrer, dans le cadre de Broadway, l'utilisation des concepts introduits. La modélisation proposée de Broadway repose sur une analyse la plus réaliste possible du problème de l'assistance à la navigation. Toutefois, de nombreuses décisions liées au domaine doivent être approfondies notamment par des études ergonomiques et psychologiques. Ce travail dépasse le cadre de notre thèse, toutefois nous identifions certains paramètres importants de notre modélisation pour permettre leur modification (seuils, fonctions d'agrégation, ou autres méthodes). La liste de ces paramètres est donnée dans l'annexe D. Ainsi, Broadway représente un type de modélisation dont l'objectif est de donner un cadre opérationnel pour l'évaluation de notre modèle d'indexation.

2.1.1 Identification des variables d'observation

Pour l'observation du comportement des utilisateurs, nous avons retenu quatre variables représentant l'adresse, le contenu, l'évaluation et le ratio d'affichage des pages Web visitées. Ces variables qui sont notées respectivement Adresse, Contenu, Evaluation et Affichage, prennent des valeurs successives au cours du processus de recherche d'information pour chaque utilisateur. Nous décrivons l'utilisation de chacune de ces variables. Nous donnons également les caractéristiques de leur domaine de valeurs ainsi que les modules de Broadway utilisés pour les calculer.

2.1.1.1 *Adresse d'une page Web*

Comme nous l'avons expliqué auparavant (cf. chapitre I, §1.2.1), le Web est un ensemble interconnecté de ressources dans différents formats et accessibles par de multiples protocoles. Dans le cadre de Broadway, nous nous limitons à un espace bien précis au sein du Web. Cet espace est constitué des ressources accessibles via le protocole HTTP et qui respectent le format HTML. Cet

espace représente la partie la plus importante du Web actuel. Les ressources de cet espace sont accessibles via une adresse appelée URI (*Universal Resource Identifier*) dont la forme générale est la suivante (RFC 1738, 1994 ; RFC 2068, 1997,§3.2.2 ; RFC 1808, 1995) :

HTTP_URI = "http:" " //" machine [":" port] ["/" chemin] ["?" requête] ["#" fragment]

Toutefois, les contraintes d'écriture de ces adresses et de leur interprétation par les serveurs ne sont pas suffisamment strictes pour garantir l'unicité de l'association entre une adresse et une ressource. En effet, un même fichier peut être accessible par plusieurs URI non identiques. C'est le cas, par exemple, des URI suivantes :

http://www.inria.fr/aid/
http://www.inria.fr/aid
http://www.inria.fr/aid/index.html
http://www.inria.fr/aid/pub/../
http://www.inria.fr/aid/.
http://www.inria.fr/aid/./

Cependant aucune règle du protocole HTTP ou des URI ne permet de garantir que, dans tous les serveurs HTTP, ces types d'URI mèneront effectivement à un contenu identique. C'est pourquoi, nous clarifions les contraintes de comparaison utilisées dans Broadway pour déterminer si deux adresses sont identiques. Tout d'abord, la partie fragment, servant à référencer un point précis dans une page, est ignorée. De plus, nous reprenons et précisons les contraintes d'égalité reconnues entre deux URI pour le protocole HTTP (RFC 2068, 1997, §3.2.3 ; Bray, 1996). Ainsi deux adresses sont égales si et seulement si les parties machine, port, chemin et requête sont identiques avec les contraintes suivantes :

- Deux machines sont égales si leurs adresses IP sont identiques. Si les machines sont spécifiées symboliquement par leur nom sous forme de chaînes de caractères qui sont égales sans prise en compte de la casse, les machines sont considérées identiques, sans comparaison de leurs adresses IP.
- Deux ports sont identiques s'ils indiquent le même numéro. Un port vide est équivalent au port par défaut, soit 80.
- Deux chemins sont identiques si les chaînes de caractères correspondantes sont égales avec prise en compte de la casse. Un chemin vide est équivalent au chemin « / ».
- Deux requêtes sont identiques si les chaînes de caractères correspondantes sont égales avec prise en compte de la casse.

Le Tableau V-1 donne l'exemple de deux pages ayant différentes URI qui seront considérées comme identiques par Broadway.

URI d'une même page A	*URI d'une même page B*
http://www.inria.fr	http://www.inria.fr/index.html
http://www.inria.fr:80	http://WWW.inria.fr/index.html
http://www.inria.fr/	http://www.inria.fr/index.html#theme

Tableau V-1 : URI de pages Web

Enfin, Broadway considère comme des adresses de page Web uniquement les URI qui permettent d'accéder à des ressources dont le format est HTML. Cette information est extraite des communications HTTP avec les serveurs, dans lesquelles le champ Content-Type doit indiquer le type MIME (RFC 2046, 1996) text/html. Les adresses des pages Web visitées par un utilisateur sont transmises à Broadway par les observateurs. Les observateurs sont des programmes (applets) insérés par Broadway dans les pages Web et sont activés lors de l'affichage d'une page.

2.1.1.2 *Contenu d'une page Web*

Le contenu d'une page Web est décrit par du texte brut augmenté de directives de formatage, l'ensemble respectant le langage HTML (Raggett, 1997). Ces directives sont interprétées par le navigateur de manière à afficher le texte avec différents effets : hyperliens, insertion d'images, changement de la taille de la fonte, titre, sous-titres, liste numérotée… Plus précisément, le texte est enrichi de descripteurs (*tags*) qui apportent principalement des informations de mise en page et de structuration (cf. Figure V-8). Par exemple, les deux descripteurs <B> et </B> permettent de définir une zone de texte à afficher en gras (*bold*). Sur le plan de la structuration, les descripteurs <H1> </H1>, <H2> </H2>, <H3> </H3> permettent de définir des titres de niveaux respectifs 1, 2 et 3. De plus, une page est divisée en deux parties principales : une partie descriptive identifiée par les descripteurs <HEAD> et </HEAD>, et une partie informative entre les descripteurs <BODY> et </BODY>. La partie descriptive donne le titre de la page entre les descripteurs <TITLE> et </TITLE> ainsi que des métadonnées (*metadata*) sur les pages dans des descripteurs <META...> : mots clefs, description de la page, nom de l'auteur, logiciel utilisé pour écrire cette page. La partie informative contient le texte formaté à afficher. Ces deux parties sont réunies dans une zone encadrée par les descripteurs <HTML> et </HTML>, et la page commence par un descripteur <!DOCTYPE... > donnant la version du format utilisé.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>Bienvenue sur le site Web de l'INRIA</TITLE>
.....
<META  NAME="keywords"
       CONTENT=   "INRIA, recherche, informatique,
                  automatique, institut">
<META  NAME="description"
       CONTENT="INRIA, page d'accueil">
</HEAD>
<BODY>
.....
<H1>Actualitées</H1>
....
<H1>Présentation</H1>
....
</BODY>
</HTML>
```

Figure V-8 : exemple de structuration d'une page Web

L'objectif de la variable Contenu est de résumer les informations de la page pour pouvoir effectuer ultérieurement des comparaisons. Nous ne pouvons pas enregistrer le contenu intégral de chaque page car cette solution demanderait trop de ressources sachant que les pages considérées sont potentiellement issues du Web tout entier. L'algorithme utilisé dans Broadway permet alors de ne retenir d'une page qu'un ensemble de *m* mots clefs au maximum. Comme nous l'avons indiqué, une page Web donne déjà un ensemble de mots clefs grâce au descripteur <META...>. Cependant, ce descripteur n'est pas obligatoire et, dans la pratique, peu de pages Web l'utilisent. Des travaux en cours étudient la normalisation et l'extension de la gestion des métadonnées en vue de meilleures possibilités de traitements automatiques des pages Web (cf. *Resource Description Framework*, Lassila & Swick, 1998). Nous utilisons donc un algorithme d'extraction de mots clefs, spécifique et adapté aux pages HTML :

1. élimination des mots toujours considérés comme insignifiants (*stopwords*),
2. réduction des mots à leur racine (*stem*) pour, par exemple, ne pas différencier les accords d'un même mot (pluriel/singulier, féminin/masculin, conjugaisons),

3. affectation d'un coefficient de pertinence δ à chaque racine,
4. sélection des *m* mots ayant les plus grands coefficients de pertinence δ supérieurs au seuil δ_{min}.

Le principe de ce traitement est classique dans le domaine de la recherche d'information (Korfhage, 1997, chapitre 5 ; Porter, 1980). Le coefficient de pertinence d'une racine ω dans un document est généralement calculé de la manière suivante, étant donné la fréquence d'occurrence de la racine dans le document normalisée par sa taille, notée *tf*, et la fréquence des documents qui contiennent cette racine, notée *df*, dans une collection de *N* documents :

$$\delta(\omega) = tf\,.\,idf = tf\,.\,(\log_2 \frac{N}{df})$$

Ce coefficient permet de mieux noter une racine qui apparaît souvent dans une page, tout en regardant si elle est discriminante à la vue des autres documents. Cependant, ce type d'approche ne prend pas du tout en compte la spécificité des pages HTML qui présentent une certaine structuration. C'est pourquoi une extension a été récemment proposée (Cutler *et al.*, 1997) et permet de calculer le coefficient δ, en définissant six classes d'occurrence d'une racine : le titre de la page, les titres de niveaux 1 et 2, les titres de niveaux 3, 4 et 5, les mots en gras, le texte normal et le texte des hyperliens qui mènent à la page considérée. Un vecteur de fréquence de la racine dans ces classes ainsi qu'un vecteur de poids d'importance entre ces classes sont alors utilisés dans la formule suivante :

$$\delta(\omega) = (tf_1 \quad \cdots \quad tf_6)\begin{pmatrix} w_1 \\ \vdots \\ w_6 \end{pmatrix} idf$$

L'affectation des poids entre les classes est effectuée expérimentalement à partir de requêtes dont les experts ont manuellement identifié les documents pertinents, et dont le système tente d'optimiser les résultats obtenus automatiquement. Toutefois, cette technique n'est pas directement applicable dans Broadway puisqu'il n'existe pas de collection figée de documents. Les utilisateurs sont effectivement libres de naviguer sur un nombre quelconque de pages Web qui ne sont pas connues *a priori*. La valeur du coefficient *df* n'est donc pas fixe. De plus, cette approche ne prend pas en compte la classe des mots clefs déjà définis par la page via les métadonnées.

Nous proposons alors une méthode similaire et adaptée à nos contraintes. Nous définissons uniquement trois classes de mots pour une page : les mots clefs existants (métadonnées), les mots mis en valeur (titre de la page, description et sous-titres de niveaux 1, 2 et 3), et les mots apparaissant dans les autres zones. La prise en compte d'uniquement trois classes permet, plus facilement et de manière heuristique, de donner des poids d'importance aux classes : les mots clefs existants sont très pertinents, puis les mots mis en valeur sont plus pertinents que les autres. Enfin, nous supposons que le facteur évaluant la qualité discriminante d'un mot ω est donné par la fonction Δ et ne change pas dans le temps. Ces facteurs peuvent être précalculés à partir d'une collection de documents typiques du domaine d'activité du groupe utilisant Broadway. Ainsi, pour une page donnée Π, Broadway extrait les *m* mots ayant les plus grands coefficients d'importance δ calculés sur ces trois classes :

$$\delta(\omega) = (tf_{MotClefs}, tf_{MisEnValeur}, tf_{Autre})\begin{pmatrix} w_{MotsClefs} \\ w_{MisEnValeur} \\ w_{Autre} \end{pmatrix} \Delta(\omega)$$

Dans l'architecture de Broadway, le serveur de descriptions de pages est chargé d'effectuer ces traitements et de sauvegarder les mots clefs identifiés. De plus, grâce aux informations données par les serveurs HTTP (date d'expiration et dernière date de modification), ce serveur ne met à jour ces mots clefs que lorsqu'une modification a eu lieu.

2.1.1.3 *Evaluation explicite des pages*

La variable mesurant l'évaluation explicite des pages Web permet d'obtenir une information directe sur la satisfaction de l'utilisateur au regard de son objectif implicite. Cet objectif peut être plus ou moins précis et peut évoluer durant une navigation. L'observation des évaluations va permettre d'interpréter cet objectif et de noter les recommandations faites par le système. Grâce à la barre d'outils de Broadway, les évaluations sont directement associées à la page visualisée avec quatre niveaux sémantiques qui sont représentés par des entiers :

- *pas pertinent* (valeur –1) : la page courante n'a aucun rapport avec l'objectif de recherche et représente un contresens.
- *indifférent* (valeur 0) : il s'agit de la valeur par défaut et dénote un avis indifférent de l'utilisateur.
- *pertinent* (valeur 1) : la page courante est connectée au thème de la recherche, ou présente des hyperliens qui traitent de thèmes proches de l'objectif de recherche.
- *très pertinent* (valeur 2) : la page courante répond précisément à l'objectif de recherche.

2.1.1.4 *Evaluation implicite des pages : le ratio d'affichage*

La variable précédente permet d'observer les évaluations explicites des pages Web données par les utilisateurs. Dans Broadway, nous choisissons également de considérer le temps de lecture comme critère d'intérêt *implicite* d'un utilisateur dans une page. La combinaison de critères explicites et implicites permet une meilleure représentation du comportement des utilisateurs et mène généralement à de meilleures interprétations (Sakagami & Kamba, 1997). Nous prenons l'hypothèse que, plus un utilisateur passe de temps à lire une page, plus il est intéressé par son contenu. Cependant, le temps de lecture est difficilement observable et nous le supposons égal au temps d'affichage. De plus, pour pouvoir comparer des temps d'affichage de pages différentes, nous normalisons cette durée par la taille de la page et nous obtenons le *ratio d'affichage*. Ce ratio représente en fait l'inverse de la vitesse de lecture.

Concrètement, les valeurs de cette variable sont mesurées en effectuant le rapport entre la durée de l'affichage en millisecondes, et le nombre de mots lisibles dans une page. Le temps d'affichage est obtenu par l'observateur qui informe très précisément du temps écoulé entre le début de l'affichage de la page et son effacement pour afficher une nouvelle page. Le nombre de mots lisibles d'une page HTML est calculé par le serveur de descriptions de pages qui doit préalablement analyser la structure HTML pour en éliminer tous les descripteurs ainsi que leur contenu. Enfin, nous considérons qu'un ratio d'affichage supérieur au seuil a_{max} montre que l'utilisateur a certainement interrompu sa lecture. Nous n'utilisons cette variable que pour évaluer positivement l'intérêt d'un utilisateur lorsque le ratio est supérieur à un seuil donné. En effet, pour un ratio petit, l'utilisateur peut avoir trouvé rapidement l'information recherchée.

2.1.2 Détermination de la sémantique d'un enregistrement et de son contexte

Dans le cadre de Broadway, un enregistrement est appelé une *navigation*. Une navigation contient l'évolution des quatre variables d'observation (adresse, contenu, évaluation, ratio d'affichage) pour la durée d'une recherche d'information. Une navigation débute lorsqu'un utilisateur a le désir d'effectuer une recherche et se termine lorsqu'il a trouvé l'information recherchée ou bien s'il abandonne la

recherche. Une navigation est supposée *cohérente*, c'est-à-dire que l'objectif de recherche peut évoluer ou se préciser, mais il ne doit pas y avoir de changement d'objectif. Par exemple, une navigation commencée pour rechercher des informations sur le langage de programmation Java et les modules de communication disponibles, peut se préciser par la lecture de pages qui présentent des applications spécifiques réalisées en Java avec RMI. Toutefois, cette navigation sera considérée comme incohérente si l'utilisateur parcourt finalement le site de BMW parce qu'il cherche une nouvelle voiture et que l'un des concepteurs de ces applications mentionne ce site sur sa page personnelle.

L'évolution de chaque variable est représentée par une chronique échantillonnée dans la mesure où l'unité de temps pris en compte est le changement d'une page. Un changement de page correspond à l'affichage dans le navigateur d'une page dont l'adresse est différente de la dernière page affichée chronologiquement. Le contexte d'une navigation contient une description instantanée des caractéristiques pertinentes de la navigation. Deux attributs du contexte permettent de sauvegarder respectivement les noms des machines traversées durant la navigation, et les mots clefs de la navigation. Cet ensemble de mots clefs correspond à l'union des mots clefs issus des pages Web de la navigation.

La Figure V-9 donne l'exemple d'une navigation enregistrée par Broadway dans laquelle un utilisateur désire retrouver des informations concernant Broadway lui-même. Il commence alors sa navigation sur la page d'accueil de l'INRIA, passe par les pages de l'équipe de recherche AID et finalement trouve la page relatant les développements de logiciels et notamment ceux de Broadway.

Navigation #16 from mjaczyn

start date: Wed Nov 26 19:28:20 GMT+03:30 1997

end date: Wed Nov 26 19:30:51 GMT+03:30 1997

keywords (stems): PEOPLE, REPORT, INRIA, PUBLIC, CONFER, ACTIV, STAGE, MICHEL, THEME, SITE, SOFTWAR, DEADLIN, JACZYNSKI, ACTION, RESEARCH, PREFER

hosts: www.inria.fr

#	Page address	Content (stems)	Display	Evaluation
0	http://www.inria.fr/Recherche/activites-eng.html	INRIA, ACTIV, RESEARCH	1.0	0
1	http://www.inria.fr/Themes/Theme3-eng.html	INRIA, THEME	1.0	0
2	http://www.inria.fr/Equipes/AID-eng.html	ACTION	3.0	0
3	http://www.inria.fr/aid/aid-eng.html	ACTION	6.0	☺
4	http://www.inria.fr/aid/people.html	PEOPLE, ACTION	8.0	0
5	http://www.inria.fr/aid/personnel/Michel.Jaczynski/michel-eng.html	MICHEL, SITE, JACZYNSKI, PREFER	38.0	0
6	http://www.inria.fr/aid/personnel/Michel.Jaczynski/pub-eng.html	REPORT, PUBLIC, MICHEL, JACZYNSKI	10.0	0
7	http://www.inria.fr/aid/personnel/Michel.Jaczynski/michel-eng.html	MICHEL, SITE, JACZYNSKI, PREFER	1.0	0
8	http://www.inria.fr/aid/people.html	PEOPLE, ACTION	4.0	0
9	http://www.inria.fr/aid/aid-eng.html	ACTION	8.0	0
10	http://www.inria.fr/aid/confs.html	CONFER, DEADLIN, ACTION	0.0	0
11	http://www.inria.fr/aid/aid-eng.html	ACTION	2.0	0
12	http://www.inria.fr/aid/training.html	STAGE, ACTION	1.0	0
13	http://www.inria.fr/aid/software.html	SOFTWAR, ACTION	2.0	☺

Figure V-9 : extrait d'une navigation

2.1.3 Définition de la représentation des cas

Nous décrivons la représentation du cas cible puis celle des deux types de cas sources (potentiels et concrets).

2.1.3.1 *Cas cible*

Un cas cible référence un instant précis dans une navigation et se compose d'un couple (*problème*, *solution*) et des informations de gestion : date de création, nom de l'utilisateur, origine du cas, nombre de remémorations (nombre de fois où le cas a été retrouvé dans les raisonnements), et nombre de confirmations (nombre de fois où au moins une de ses recommandations a été confirmée après révision). Le problème donne la situation comportementale dont uniquement les indices instantanés sont connus (contexte de la navigation) : nom des machines accédées et mots clefs des pages visitées. La solution qui est recherchée doit donner une liste de couples (adresse d'une page recommandée, coefficient entre 0 et 1). Le coefficient d'une page indique l'estimation du *facteur de pertinence* de la recommandation étant donné l'état courant de la navigation. L'expression typique d'un cas cible est la suivante :

```
CasCible {
    Identificateur = "exemple-cas-cible-Broadway",
    UtilisateurNavigation = "mjaczyn", Date = "26/11/97",
    Origine = cas-cible,
    NbRemémorations = -, NbConfirmations = -,
    Problème = {
        Enregistrement= N, Instant= 14,
        SituationComportementale= {
            ComposanteInstantanée= {
                ContextEnregistrement= {
                    Machines= "www.inria.fr",
                    MotsClefs="ACTION, RESEARCH, DEADLIN,..."}},
            ComposanteComportementale= {
                ComportementsElémentaires = {},
                ContraintesTemporelles= {}}}}
    Solution= { ?}}
```

2.1.3.2 *Cas potentiels et cas concrets*

Dans Broadway, les cas concrets sont uniquement issus de l'instanciation d'un seul patron de cas potentiels. Le patron proposé repose sur des règles de construction d'une situation comportementale et d'une liste de pages recommandées pour un instant donné à l'intérieur d'une navigation. Ce patron est applicable uniquement si la position de l'instanciation dans la navigation est supérieur à $l_{\min}$ et si le couple formé de la navigation associée à l'instant d'instanciation ne correspond pas à une instanciation déjà effectuée et mémorisée sous forme de cas concret (liste notée L_C).

Applicable_P : Navigation, Intant(u) $\rightarrow$ Booléen

$\{ \text{Applicable}_P(n,i) = \text{Len}(n) > l_{\min} \text{ et } \langle n,i \rangle \notin L_C \}$

La fonction d'instanciation du patron crée un cas potentiel explicite, en suivant des règles de construction de la situation comportementale et de la solution (l'algorithme d'instanciation est décrit précisément dans l'annexe D, §2.1). La situation comportementale comprend une composante instantanée (noms des machines et mots clefs) et une composante comportementale. La composante comportementale référence un instant précis dans la navigation et sélectionne les indices comportementaux d'un cas par rapport à l'instant de référence :

- les p dernières pages sur les variables de l'adresse et du contenu pour identifier le contexte immédiat de réutilisation d'un cas dans une navigation (notion de restriction, cf. chapitre II, §2.2.1.1),

- un ensemble de pages passées qui sont sélectionnées[51]pour caractériser le comportement de l'utilisateur : une évaluation explicite élément d'une liste e_p, ou ratio d'affichage dépassant le seuil a_p,
- un ensemble de contraintes temporelles permettant d'ordonner les pages passées sélectionnées.

Ainsi, les conditions de réutilisation d'un cas sont manipulées plus facilement à travers la restriction (structure figée) et peuvent être précisées avec la liste des pages passées sélectionnées et ordonnées.

La partie solution du cas qui donne une liste de pages recommandées est construite en sélectionnant les pages pertinentes futures par rapport à l'instant de référence (seuil φ_f). Le facteur de pertinence de chaque page est calculé par agrégation de l'évaluation explicite et de l'évaluation implicite (fonction φ). Pour la sélection des pages passées et futures, si une même page est visitée plusieurs fois dans une navigation, nous ne retenons que la plus récente dépassant les seuils imposés.

La Figure V-10 donne l'exemple d'un cas concret créé à partir de l'instanciation du patron de cas potentiels dans la navigation de la figure précédente (cf. Figure V-9) pour un instant correspondant à la page #9. La restriction permet de sélectionner les trois dernières pages (p=3) et les règles du patron sélectionnent la page #3 pour son évaluation (e_p={2}) et la page #5 pour le ratio d'affichage (a_p=20). A cette situation comportementale, la page #13 est associée (φ_f=0,8) et représente la page recommandée de ce cas. Suivant la notation introduite (cf. chapitre II, §2.1.3.3), nous représentons ce cas de la manière suivante :

CasConcret {
Identificateur = "exemple-cas-Broadway",
UtilisateurNavigation = "mjaczyn", **Date** = "26/11/97",
Origine = cas-potentiel,
NbRemémorations = 0, **NbConfirmations** = 0,
Problème = {
Enregistrement= R, Instant= 9,
SituationComportementale= {
ComposanteInstantanée= {
ContextEnregistrement= {
Machines= "www.inria.fr",
MotsClefs="PEOPLE, REPORT, INRIA…"}},
ComposanteComportementale= {
ComportementsElémentaires = {
b_1={ ts= K_1, loc=K_1(7,9), Ic=Ψ_P(.,Adresse,-2,0))},
b_2={ ts =K_2 loc=K_2(7,9), Ic=Ψ_P(.,Contenu,-2,0)},
b_3={ ts = K_1, loc=K(5,5), Ic=Ψ_B(.,Adresse,K_1(5,5),K_2(5,5))},
b_4={ ts = K_2, loc=K_2(5,5), Ic=Ψ_R(.,Contenu,f(.,3)},
b_5={ ts = K_3, loc=K_3(5,5),Ic=Ψ_R(.,Evaluation,f(.,3))},
b_6={ ts = K_4, loc=K_4(5,5),Ic=Ψ_R(.,Affichage,f(.,3))}
b_7={ ts = K_1, loc=K_1(3,3),Ic=Ψ_B(.,Adresse,K_1(3,3),K_2(3,3))},
b_8={ ts = K_2, loc=K_2(3,3),Ic=Ψ_R(.,Contenu,f(.,7))},
b_9={ ts = K_3, loc=K_3(3,3),Ic=Ψ_R(.,Evaluation,f(.,7))},
b_{10}={ ts = K_4, loc=K_4(3,3),Ic=Ψ_R(.,Affichage,f(.,7))}},
ContraintesTemporelles= {
c1= avant(b_7,b_3) }}},
Solution= { p1= { Adresse=K_1(14,14),
Evaluation= φ(K_3(13,13), K_4(13,13))}}}

[51] Cette sélection peut recouvrir les pages qui constituent la restriction.

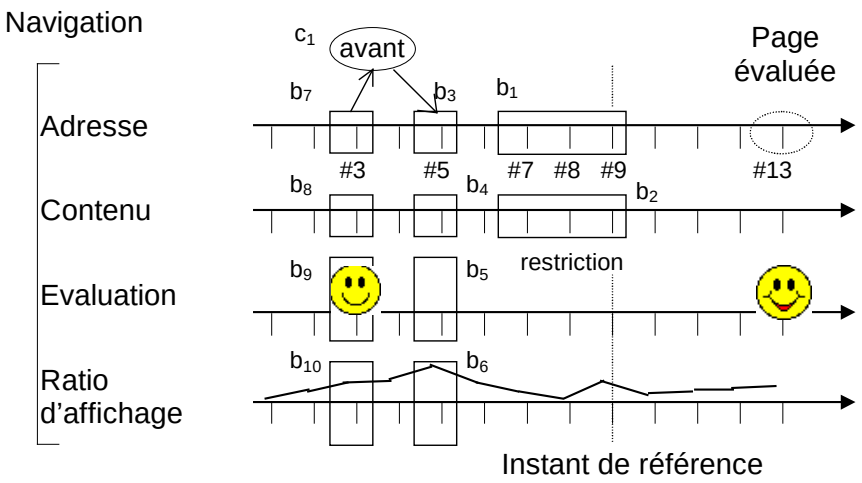


Figure V-10 : exemple d'un cas concret

Dans cette expression, la fonction f permet d'obtenir, suivant son numéro, le comportement élémentaire homologue déjà identifié dans une composante comportementale. De plus, nous introduisons une nouvelle fonction de localisation Ψ_B qui permet d'identifier la première page passée à partir de l'instant de référence, qui dépasse le seuil de similarité τ_{loc}, soit avec l'adresse, soit avec le contenu. Dans ce but, les fonctions de similarité $s_{Adresse}$ et $s_{Contenu}$, que nous décrivons dans la section suivante (cf. §2.1.4.1.1), sont utilisées. Notre objectif est de pouvoir identifier la première page qui présente une similarité très élevée (τ_{loc} proche de 1) avec soit le contenu, soit l'adresse d'une page. Durant la mise en correspondance, cette fonction est donc utilisée pour identifier dans un cas cible la page homologue d'une page sélectionnée dans un cas source.

Ψ_B : ComposanteComportementale, Variable, URL, EnsembleMots → TSchronique(u,T)

{ Ψ_B (c,v,adresse, contenu) = g(Γ(Chr(c,Adresse),1,Ref(c)), Γ(Chr(c,Contenu),1,Ref(c)),
Γ(Chr(c,Evaluation),1,Ref(c)), Γ(Chr(c,Affichage),1,Ref(c)),
adresse, contenu)

g(A.<i,a> , M.<i,m> , E.<i,e> , R.<i,r> , adresse, contenu) =
[<i,a>] si ($s_{Adresse}$ (a, adresse) > τ_{loc} ou $s_{Contenu}$(m, contenu) > τ_{loc}) et (e ∈ e_p ou a_{max} >= a > a_p)
g(A,M,E,R, adresse, contenu) sinon

g([],[],[],[],adresse, contenu)=[]

}

2.1.4 Conception des phases de raisonnement

Le raisonnement effectué par Broadway suit les quatre phases classiques en raisonnement à partir de cas : recherche, réutilisation, révision et apprentissage. Dans la spécification de ces phases, nous montrons comment nous avons utilisé les principes de gestion définis dans le modèle d'indexation par situations comportementales.

2.1.4.1 *Phase de recherche*

La phase de recherche repose sur l'utilisation de fonctions de similarité que nous introduisons en premier lieu, et d'une décomposition en étapes conformément au modèle d'indexation.

2.1.4.1.1 Fonctions de similarité

Pour effectuer les comparaisons entre cas, nous avons défini cinq fonctions élémentaires de similarité (une fonction pour chaque variable d'observation et une fonction pour les contraintes temporelles) ainsi qu'une fonction d'agrégation. Pour la variable Adresse, nous utilisons la structure hiérarchique sous-jacente dans l'organisation des données accessibles depuis un serveur HTTP. Les pages sont en effet généralement organisées en répertoires et nous effectuons une décomposition des adresses (cf. Figure V-11).

Chaque répertoire possède une sémantique et contient des pages qui sont reliées suivant le point de vue du concepteur. De plus, les répertoires les plus profonds contiennent des informations plus précises que les répertoires de surface. Nous utilisons alors la fonction élémentaire de similarité suivante qui prend en compte ces propriétés (REMIND, 1992) :

$$s_{\text{Adresse}}(x, y) = 1 - \frac{\text{h}(x, \text{MSCA}(x, y)) + \text{h}(y, \text{MSCA}(x, y))}{\text{h}(x, \text{root}) + \text{h}(y, \text{root})}$$

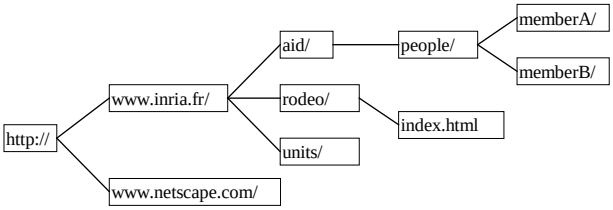


Figure V-11 : décomposition hiérarchique des adresses

Dans cette fonction, une adresse est considérée comme la référence d'un nœud d'une hiérarchie de symboles. MSCA donne le nœud commun le plus spécifique des deux adresses, et h donne le nombre de liens entre deux nœuds. Par exemple, nous obtenons les résultats suivants :

s_{Adresse}(http://www.inria.fr/aid/people/memberA, http://www.inria.fr/aid/people/memberB) = 0,75

s_{Adresse}(http://www.inria.fr/aid/people/memberA, http://www.inria.fr/rodeo/index.html) = 0,28

s_{Adresse}(http://www.inria.fr/rodeo, http://www.inria.fr/aid) = 0,5

Pour la variable Contenu, nous utilisons une fonction élémentaire de similarité prenant en compte le nombre de mots clefs communs entre deux pages :

$$s_{\text{Contenu}}(x, y) = \begin{cases} 0 \text{ si } x \cup y = \varnothing \\ \dfrac{\text{Card}(x \cap y)}{\text{Card}(x \cup y)} \text{ sinon} \end{cases}$$

Pour la variable Evaluation, nous utilisons les entiers associés à chaque niveau d'évaluation {-1,0,1,2} pour définir la fonction suivante :

$$s_{\text{Evaluation}}(x, y) = 1 - \frac{|x - y|}{3}$$

Pour la variable Affichage, nous considérons qu'un ratio d'affichage supérieur à a_{max} montre que l'utilisateur a certainement interrompu sa lecture pour effectuer une autre tâche. Dans ce cas, les ratios ne sont pas comparables, et nous donnons une similarité de 0,5. Sinon, nous définissons une fonction sur le domaine $[0 \ldots a_{max}]$:

$$s_{\text{Affichage}}(x,y)=\begin{cases}0{,}5 \text{ si } x>a_{\max} \text{ ou } y>a_{\max}\\ 1-\dfrac{|x-y|}{a_{\max}} \text{ sinon}\end{cases}$$

La dernière fonction élémentaire de similarité permet de comparer les contraintes temporelles. A partir des n contraintes exprimées dans une situation comportementale d'un cas source, $n(n+1)/2$ contraintes temporelles prises deux à deux sont dérivées (cet ensemble est noté y). Les comportements élémentaires homologues à ce cas source sont identifiés dans le cas cible. Sur l'ensemble y des contraintes initiales, seulement certaines contraintes restent vérifiées (cet ensemble est noté x). La similarité suivant les contraintes temporelles est alors donnée par la fonction suivante :

$$s_{\text{ct}}(x,y)=\frac{\text{Card}(x)}{\text{Card}(y)}$$

Par exemple, si la situation d'un cas source donne la séquence de pages sélectionnées (ABCDEF), nous obtenons les résultats suivants, avec q, la fonction retournant l'ensemble des contraintes vérifiées d'une séquence de pages :

s_{ct}(q(ABCDEF),q(ABCDEF))=1
s_{ct}(q (ABCDEF), q(FEDCBA))=0
s_{ct}(q (ABCDEF), q(ABEF))=6/15=0,4
s_{ct}(q(ABCDEF), q(ABEFCD))=11/15=0,73

Ces différentes fonctions élémentaires de similarité sont utilisées à plusieurs reprises dans les étapes de la recherche, et nous utilisons une fonction d'agrégation calculant une moyenne du vecteur v des différentes similarités élémentaires obtenues, pondérées par un vecteur de poids w :

$$g(v,w)=\frac{\sum_{i=1}^{n} v_i.w_i}{\sum_{i=1}^{n} w_i}$$

2.1.4.1.2 Etapes de recherche

La phase de recherche et, plus globalement, le raisonnement ne sont exécutés que lorsque la navigation courante a une longueur supérieure au minimum requis, noté $l_{\min}$. En effet, en dessous de cette longueur, nous estimons ne pas avoir assez d'information pour tenter de calculer des recommandations. L'objectif de la recherche est d'identifier au maximum les k meilleurs cas, suivant une recherche en deux phases. Premièrement, la phase de recherche des cas concrets est effectuée. Si cette recherche est un échec, c'est-à-dire qu'aucun cas n'est retrouvé, alors la deuxième phase qui recherche des cas potentiels est effectuée. Si la recherche ne permet pas d'identifier des cas, aucune recommandation ne sera fournie. Durant les étapes de recherche, nous mettons en évidence l'utilisation de seuils de similarité qui vont permettre d'assurer un minimum de similarité sur des parties mises en correspondance entre les cas sources et le cas cible. Il est en effet préférable d'arrêter la recherche plutôt que de continuer pour proposer des recommandations qui ne sont pas de qualité suffisante.

Pour la recherche des cas concrets, nous suivons les étapes de recherche suivantes :

1. *Filtrage sur la composante instantanée*. Cette première étape permet d'éliminer de la recherche les cas dont le contexte n'a pas au moins un nom de machine ou un mot clef en commun avec la navigation courante. En effet, ces cas ne peuvent être réutilisés pour le mécanisme de recommandation. Cette étape évite ainsi des comparaisons inutiles et augmente l'efficacité de la recherche.

2. *Filtrage sur la restriction.* L'objectif de cette étape est d'assurer un minimum de similarité des cas sources avec les toutes dernières pages que l'utilisateur a visitées dans la navigation courante. Chaque cas possède une restriction prédéfinie qui représente les p dernières pages sur les variables Adresse et Contenu. Ainsi, chaque cas définit une matrice de $2p$ valeurs qui est mise en correspondance avec les valeurs de la navigation courante. Cette comparaison utilise deux fonctions élémentaires de similarité, s_{Adresse} et s_{Contenu}, ainsi que la fonction d'agrégation g utilisant une matrice de poids *MP*. Cette étape ne retient finalement que les meilleurs cas (plus proches voisins avec un nombre maximum de cas k_r et un seuil de similarité τ_r).
3. *Filtrage sur les comportements élémentaires.* L'objectif de cette étape est d'assurer un minimum de similarité entre les pages pertinentes des cas sources et les pages visitées dans la navigation courante. Ainsi, pour chaque cas source issu de l'étape précédente, les contraintes de localisation sont appliquées pour identifier dans la navigation courante les pages homologues aux pages passées sélectionnées. Puis, la similarité de chaque page suivant les quatre variables (fonctions s_{Adresse}, s_{Contenu}, $s_{\text{Evaluation}}$, $s_{\text{Affichage}}$) est calculée puis agrégée avec la fonction g (vecteur de poids *VP*). Une page non identifiée dans le cas cible donne comme similarité 0. Ensuite, la similarité du cas cible avec un cas source est globalement donnée par le rapport entre la somme de ces similarités, et le nombre de pages sélectionnées dans le cas source. Cette similarité est alors combinée (fonction g avec le vecteur de poids *CP*) avec le ratio de complexité de la composante comportementale du cas source (ratio du nombre de pages sélectionnées sur le nombre maximum de pages sélectionnées dans les cas sources à comparer). Cette dernière opération a pour but de privilégier des cas constitués de nombreux comportements, ce qui dénote un cas plus spécifique pour une meilleure recommandation. Pour finir, uniquement les meilleurs cas sont retournés (plus proches voisins avec un nombre maximum de cas k_c et un seuil de similarité τ_c). Cet algorithme est décrit plus précisément dans l'annexe D, §2.2.
4. *Sélection sur les contraintes temporelles.* Finalement, les cas restants sont ordonnés suivant l'agrégation (avec la fonction de similarité g et le vecteur de poids *GP*) des similarités issues des étapes 2 et 3 et de la similarité des contraintes temporelles, calculée avec la fonction s_{ct}. Les meilleurs cas suivant ce critère sont alors retournés (plus proches voisins avec un nombre maximum de cas k et un seuil de similarité τ).

Pour la recherche de cas potentiels, nous suivons également ces quatre étapes. Toutefois, pour le filtrage sur la restriction, les navigations retenues par le filtrage sur la composante instantanée sont examinées pour chaque instant où le patron de cas potentiels est applicable. Une comparaison sur la restriction est effectuée comme pour les cas concrets. Durant la sélection des meilleurs cas par plus proches voisins, si deux cas issus d'une même navigation sont à comparer, seul le meilleur est retenu[52]. A la fin de cette étape, les cas potentiels identifiés sont effectivement explicités par instanciation du patron.

La phase de recherche retourne un ensemble de cas, et chaque cas donne une liste de pages recommandées. De plus, le nombre de remémorations de chaque cas identifié est incrémenté (attribut NbRemémorations).

2.1.4.2 *Phase de réutilisation*

L'objectif de la phase de réutilisation est de considérer l'ensemble de ces cas retrouvés pour former la liste des pages à recommander, tout en leur associant un facteur de pertinence. L'algorithme suivant est alors effectué :

[52] Cette contrainte évite ainsi de prendre par la suite en considération des cas qui diffèrent uniquement sur l'instant d'instanciation au sein d'une même navigation.

1. constitution de l'ensemble des pages à examiner en effectuant l'union des pages indiquées par les cas,
2. calcul du facteur de pertinence de chaque page,
3. sélection des meilleures pages, étant donné un degré minimum τ_r et un nombre maximum de recommandations *r*.

Le facteur de pertinence d'une page repose sur le calcul d'un vecteur de caractéristiques élémentaires de pertinence, noté *F*, et de leur agrégation suivant une moyenne pondérée avec le vecteur *FP*. Le calcul de chaque caractéristique donne un résultat entre 0 et 1, où 1 représente le cas le plus favorable. Différentes caractéristiques sont utilisées pour une page donnée :

1. l'évaluation globale de la page donnée par la moyenne des évaluations des cas sources proposant cette page,
2. le degré de confiance dans cette évaluation représenté par la moyenne des similarités des cas proposant cette page,
3. la représentativité de l'évaluation globale de la page donnée par le ratio du nombre de cas qui proposent cette page et du nombre de cas à comparer,
4. la moyenne de la qualité des cas proposant cette page (la qualité d'un cas est donnée par le ratio du nombre de confirmations sur le nombre de remémorations),
5. la présence d'annotations textuelles reliées à cette page (0 absence et 1 présence),
6. le temps de chargement moyen normalisé par le maximum des temps de chargement moyen des pages à comparer (utilisation des statistiques maintenues par le serveur de descriptions de pages).

Ainsi, Broadway met à profit dans la réutilisation quatre groupes de critères :

- les solutions proposées par les cas sources (critère 1),
- les informations de la mise en correspondance (critères 2 et 3),
- les statistiques sur la réutilisation des cas (critères 4),
- les informations sur les pages maintenues par les serveurs de descriptions de pages et d'annotations (critères 5 et 6).

Apres ce calcul, la liste des *pages recommandées suggérées* est affichée à l'utilisateur.

2.1.4.3 *Phase de révision*

La phase de révision est effectuée par l'utilisateur, lorsqu'à la suite de l'affichage des recommandations, il continue sa navigation, soit en sélectionnant une de ces pages, soit en choisissant un des hyperliens qui sont proposés dans les pages en cours d'affichage. Plus précisément, l'utilisateur est capable, grâce à la barre d'outils, d'évaluer les nouvelles pages visitées et en particulier celles qui lui ont été recommandées.

La particularité de Broadway est de permettre la révision simultanée de plusieurs raisonnements. En effet, pour une même navigation, de multiples raisonnements ont pu être débutés, soit pour des recommandations en continu, soit pour des recommandations à la demande. Tous ces raisonnements restent dans l'état de révision jusqu'à la fermeture de la navigation. A ce stade final, la solution révisée pour chaque cas cible est constituée des pages futures sélectionnées et notées de manière identique à la construction de la solution par le patron de cas potentiels. Cette solution donne la liste des *pages recommandées confirmées*[53].

[53] Le terme « recommandations confirmées » s'entend dans le sens d'une « solution confirmée » issue de la phase de révision du RàPC. Les recommandations confirmées peuvent contenir, soit des pages effectivement proposées suite à la réutilisation, soit de nouvelles pages évaluées par l'utilisateur.

2.1.4.4 *Phase d'apprentissage et maintenance de la base de cas*

Cette phase prend en entrée la navigation courante et l'ensemble des raisonnements en cours. L'apprentissage doit alors effectuer la synthèse de ces raisonnements pour mettre à jour le système et lui permettre d'améliorer son processus de recommandation. L'apprentissage se fait alors par trois mécanismes :

1. *Mise à jour des cas.* Les cas concrets retrouvés qui ont servi à la réutilisation sont examinés. Si un cas concret a proposé au moins une page qui se retrouve dans la liste des pages recommandées confirmées, son nombre de confirmations est incrémenté (attribut NbConfirmations). Ainsi, la qualité du cas augmente (nombre de confirmations sur le nombre de remémorations) et permettra de mieux classer les recommandations venant de ce cas, lors de la réutilisation pour des raisonnements futurs.
2. *Ajout des cas potentiels*. Les raisonnements pour la navigation courante sont examinés en séquence. Pour chaque raisonnement deux situations se présentent : soit uniquement la recherche des cas concrets a été effectuée, soit la recherche des cas potentiels a été effectuée à la suite de l'échec de la recherche des cas concrets.
   (a) Dans la première situation, la recherche des cas potentiels est effectuée. Si un cas potentiel ainsi retrouvé propose comme recommandation au moins une page qu'aucun cas concret ne proposait et qui fait partie des pages confirmées, il est ajouté comme cas concret. Cette opération a pour but de mettre à jour l'ensemble des cas concrets pour des recommandations que le système aurait pu donner si la recherche des cas potentiels avait été faite au début du raisonnement. Pour garder le gain de performance amené par la stratégie de recherche séquentielle, cette opération doit être faite hors ligne.
   (b) Dans la deuxième situation, si un cas potentiel propose au moins une recommandation qui a été confirmée, il est ajouté à la mémoire en tant que cas concret. Il s'agit dans cette situation de mémoriser les cas potentiels qui proposent des recommandations qui peuvent être utiles dans les raisonnements futurs, puisqu'elles sont confirmées par la navigation courante.

   Dans les deux cas, si un cas potentiel est mémorisé, sa navigation associée à son instant de référence est ajoutée à la liste L_C. Ainsi, la patron de cas potentiels ne pourra plus être instancié pour cet instant dans la navigation considérée.
3. *Ajout de la navigation courante* : si la navigation comporte au moins une évaluation implicite ou explicite (évaluation élément de e_n, ou ratio d'affichage supérieur à a_n) et possède une longueur suffisante l_{min}, elle est ajoutée dans la base des navigations. Grâce à la recherche de cas potentiels effectuée au besoin, cette technique permet d'effectuer l'apprentissage de comportements non encore identifiés (apprentissage paresseux). De plus, cet apprentissage permet d'éviter la mémorisation des cas cibles qui sont très nombreux dans Broadway puisqu'un raisonnement peut être lancé à chaque changement de page d'un utilisateur.

D'autre part, le Web est un espace dynamique dans lequel les pages Web peuvent rapidement changer de contenu ou devenir inaccessibles. C'est pourquoi Broadway est doté d'un module de maintenance des cas et des navigations, qui est exécuté régulièrement pour les effacer lorsqu'ils ne sont plus utilisables. Une page Web est dite *obsolète* si les échecs des chargements dépassent un certain taux f_{obs}, ou si la similarité de son propre contenu (fonction $s_{Contenu}$) entre deux visites tombe en dessous d'un seuil[54] τ_{obs}. Lorsque le serveur de descriptions de pages repère une page obsolète, il

[54] Nous ne gérons pas le cas que nous pensons improbable, dans lequel le contenu de la page évoluerait par petits changements et ne serait donc pas détecté.

envoie un événement qui sera alors traité par le serveur de recommandations pour éliminer de la mémoire les navigations (et les cas associés) dont le taux de pages obsolètes atteint un seuil fixé η.

2.2 Utilisation de la plate-forme CBR*Tools

Nous détaillons dans cette section la configuration des points d'ouverture de CBR*Tools, nécessaire à la réalisation de Broadway. Plus précisément, nous indiquons la nature de cette configuration pour chaque point : spécialisation (création d'une sous-classe), instanciation (choix et paramétrage d'une classe existante), ou utilisation d'une classe par défaut. Si un même point d'ouverture est utilisé de différentes manières, nous ne retenons que la plus spécifique suivant cet ordre. Nous utilisons alors la structuration en cas d'utilisation (présentée au chapitre III, §2.3) et la notion de stéréotype de la notation UML (cf. annexe A) pour montrer la nature de chaque configuration.

2.2.1 Gestion du raisonnement

La réalisation du cycle de raisonnement pour Broadway est effectuée par la spécialisation de cinq points d'ouverture (notés en gras dans la figure Figure V-12) : la phase de recherche (Retrieve), la phase de réutilisation (Reuse), la phase d'apprentissage (Retain), l'objet de sauvegarde du raisonnement (Reasoning) et la fabrique (TimeExtendedReasonerFactory). Pour Broadway, le raisonnement doit être stoppé à la phase de révision puis, lors de la fermeture d'une navigation, l'ensemble des raisonnements liés à cette navigation est pris en compte par l'apprentissage. Dans ce but, le cycle du raisonnement est premièrement configuré pour être suspendu à la phase de révision (paramètre à donner à l'instance de la classe Reasoning). Deuxièmement, la classe Reasoning est specialisée pour pouvoir relier entre eux, les états des raisonnements effectués pour une même navigation. La gestion de ce chaînage est déléguée au client du système de raisonnement qui lui seul est en mesure de contrôler les raisonnements en fonction des navigations effectuées par les utilisateurs de Broadway. Lorsqu'une navigation est fermée, le système finalise le dernier raisonnement qui prend ainsi en compte les états des raisonnements qui lui sont reliés durant la phase d'apprentissage.

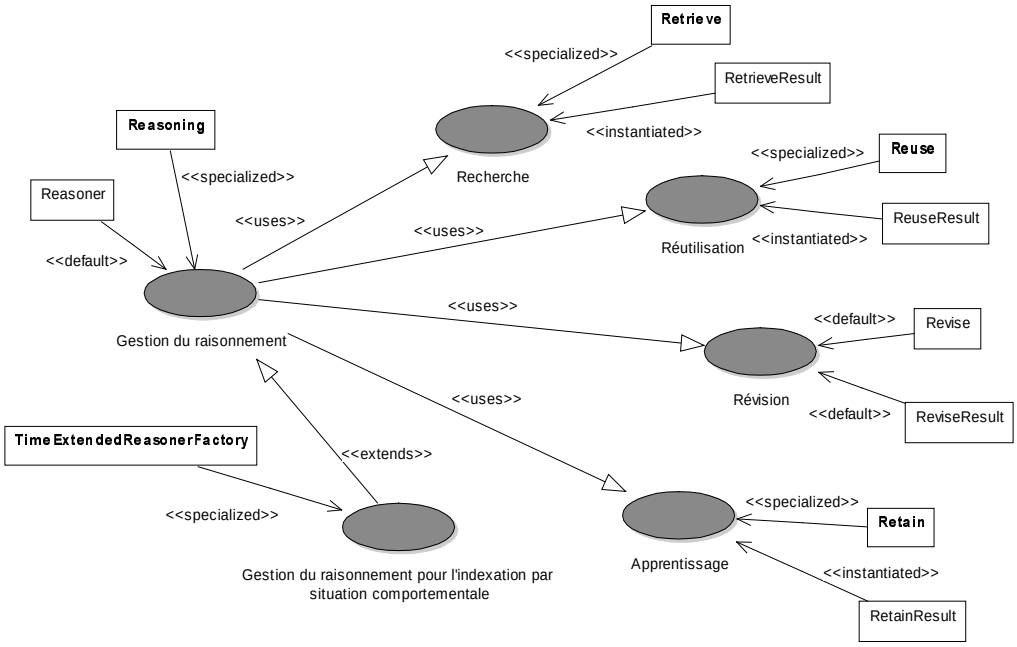


Figure V-12 : utilisation des points d'ouverture pour le raisonnement

Pour le raisonnement lui-même, la phase de recherche est tout d'abord configurée pour retourner les résultats sous la forme d'une liste de cas associés à leur similarité. La phase de réutilisation met alors en œuvre un nouveau schéma de réutilisation de recommandations que nous détaillons dans le paragraphe suivant. La phase de révision n'est pas exécutée puisqu'elle n'est pas automatisée. Cette phase correspond en effet à l'observation de la navigation effective de l'utilisateur. A la fin d'une navigation, les raisonnements sont repris pour effectuer la phase d'apprentissage qui implante de manière spécifique les directives définies (cf. §2.1.4.4).

La phase de réutilisation repose sur un nouveau schéma de réutilisation de recommandations. Ce schéma peut être configuré pour manipuler des recommandations sous la forme de pages Web pour Broadway et fournit également un cadre plus général pour la réutilisation de recommandations. Nous illustrons ainsi notre approche de capitalisation de composants pour le RàPC soutenue par CBR*Tools. Cette approche pragmatique est centrée sur la généralisation de solutions qui sont conçues pour une application particulière. Ainsi, nous obtenons non seulement une modélisation plus flexible pour l'application directement visée, mais aussi des composants réutilisables dans d'autres contextes.

Le schéma de réutilisation de recommandations suppose qu'un cas source donne comme solution une liste d'éléments évalués (recommandations). Le raisonnement doit alors former la liste des éléments évalués correspondant au cas cible courant. Ce schéma suppose également qu'un ensemble de cas a été retrouvé à la suite de la phase de recherche. La réutilisation consiste alors à réunir l'ensemble des recommandations données par les cas sources pour former la liste finale des éléments évalués qui constitue la solution au problème courant. La modélisation de cette stratégie est donnée dans la Figure V-13. De manière similaire aux autres schémas présentés, ce schéma est constitué d'un protocole ouvert (classe RecommendationsAdaptationPattern) basé sur le patron de conception *Patron de méthode*. Puis, ce protocole délègue l'analyse des éléments évalués à des objets différents suivant le patron *Stratégie* (classe RecommendationFeature). Chaque objet d'analyse retourne l'évaluation d'un élément suivant une caractéristique déterminée.

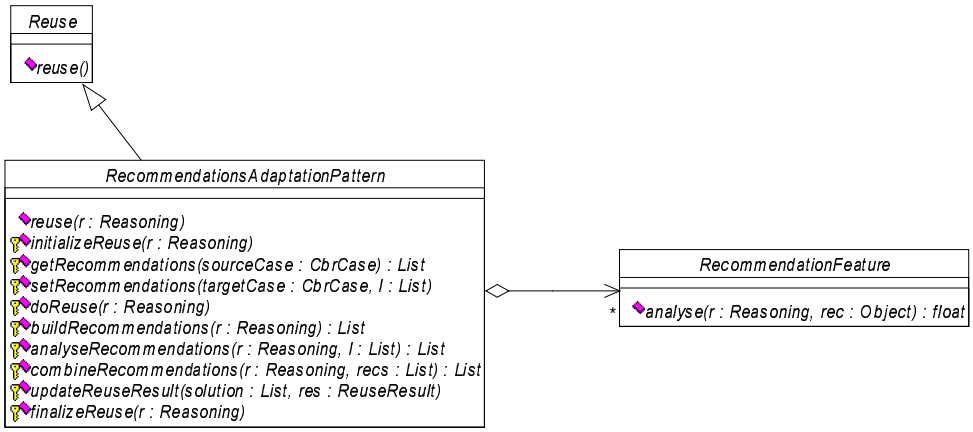


Figure V-13 : schéma de réutilisation de recommandations

Le protocole de réutilisation se décompose alors en trois étapes importantes (cf. Figure V-14) :

- constitution de la liste des éléments à étudier (méthode buildRecommendations) en réunissant les éléments donnés par les cas sources (méthode getRecommendations),
- analyse de chaque élément au regard d'une liste de caractéristiques (méthode analyseRecommendations) et agrégation des évaluations pour obtenir une évaluation globale,

- combinaison (sélection, tri) de la liste des éléments (méthode combineRecommendations) suivant les évaluations.

Cette modélisation explicite donc deux points d'ouverture spécifiques qui permettent d'une part de modifier ce protocole (constitution de la liste des éléments par union ou par intersection, analyse en combinant les caractéristiques suivant différentes fonctions d'agrégation, utilisation de différents critères de tri et de sélection pour former la liste finale) et d'autre part d'ajouter ou de retirer dynamiquement des critères d'analyse. Ainsi, nous pouvons spécialiser ce schéma dans le cadre de Broadway suivant l'algorithme défini (cf. §2.1.4.2) avec une modélisation plus stable et réutilisable.

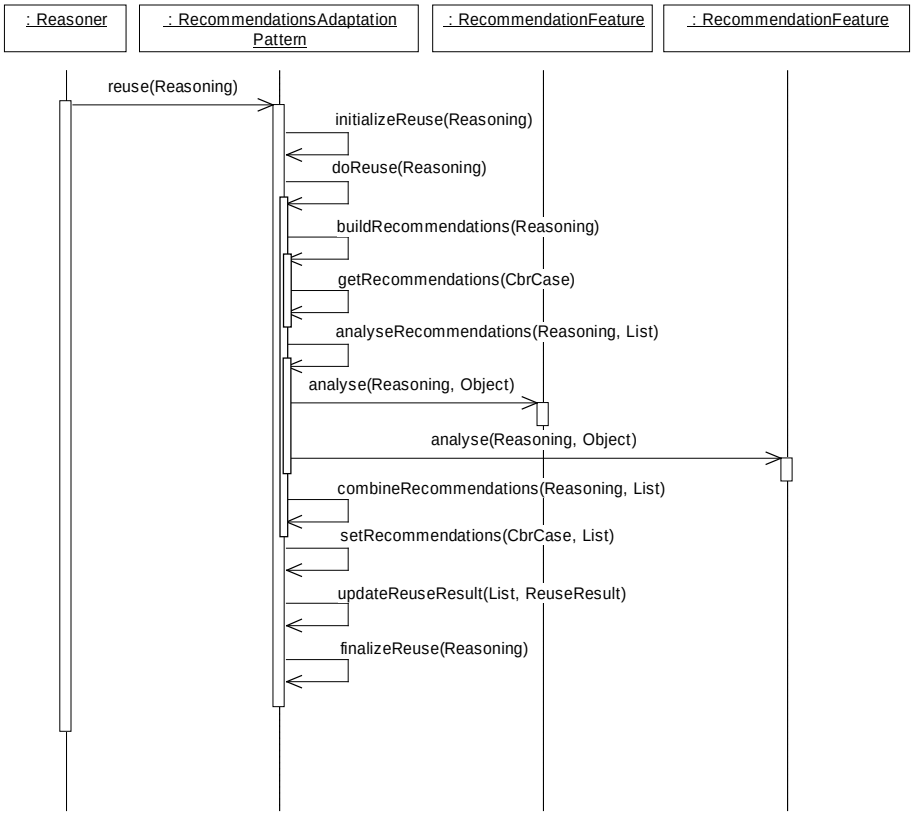


Figure V-14 : protocole du schéma de réutilisation de recommandations

2.2.2 Représentation des cas

La représentation des données d'observation, sur lesquelles repose la description des cas, nécessite uniquement la spécialisation de la représentation d'un enregistrement (cf. Figure V-15). Dans Broadway en effet, quatre variables ont été retenues (adresse, contenu, évaluation et ratio d'affichage). La spécialisation du point d'ouverture Record permet donc de déclarer ces variables. Les types standards des chroniques et des curseurs, permettant de manipuler des chroniques échantillonnées, sont alors directement utilisés.

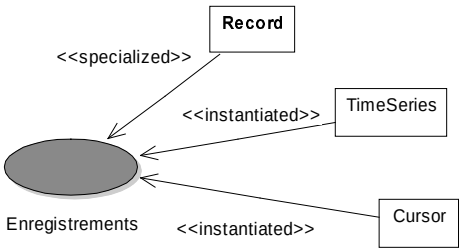


Figure V-15 : utilisation des points d'ouverture pour les données d'observation

Au-dessus de ces données d'observation, il s'agit de représenter les indices d'une situation comportementale (cf. Figure V-16). Pour la composante instantanée, le point d'ouverture CompoundIndice est utilisé pour modéliser les indices du contexte d'un enregistrement (liste de sites traversés et ensemble de mots clefs). Comme aucun autre indice instantané n'est utilisé, la structuration par défaut d'une situation instantanée qui ne comprend que le contexte, est utilisée. Pour la composante comportementale, le point d'ouverture BehaviouralSituation est spécialisé pour simplifier les opérations de manipulation des indices comportementaux (consultation du nombre de pages sélectionnées par exemple) étant donné la sémantique particulières des variables.

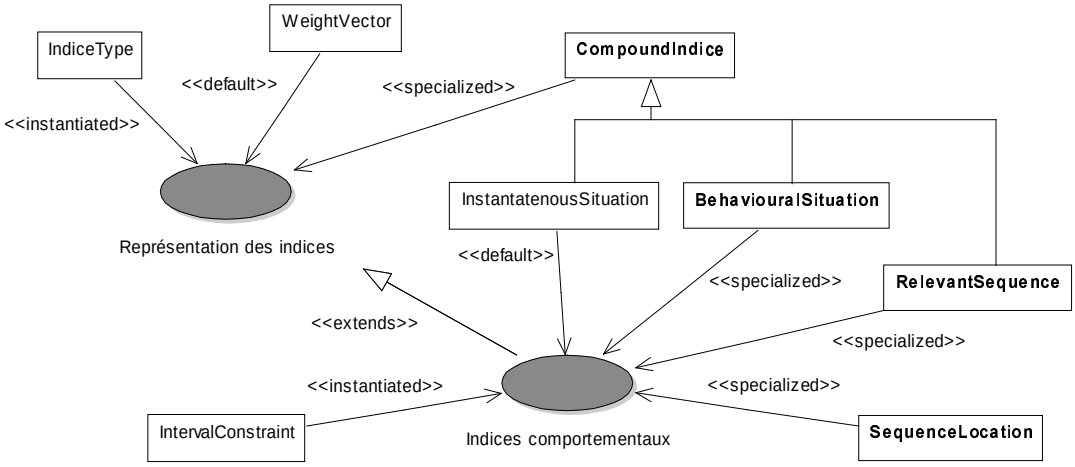


Figure V-16 : utilisation des points d'ouverture pour les indices

Le point d'ouverture RelevantSequence est également spécialisé pour déclarer les types de comportements élémentaires formés, suivant les variables, d'indices sur des entiers (évaluation), des réels (ratio d'affichage), des ensembles de mots (contenu) et des URI (adresse). Enfin, la gestion des situations comportementales nécessite également la réalisation d'une contrainte de localisation spécifique pour identifier les pages passées pertinentes. Le point d'ouverture SequenceLocation est alors spécialisé.

Les cas concrets sont alors représentés grâce à la spécialisation du point d'ouverture TimeExtendedCase qui permet de définir la partie solution (liste de pages recommandées) ainsi que des informations de gestion spécifiques à Broadway (nom de l'utilisateur, date de création, nombre de remémorations, nombre de confirmations) (cf. Figure V-17). Le patron de cas potentiels est également réalisé par la spécialisation du point d'ouverture PotentialCaseTemplate qui utilise une énumération spécifique permettant de n'instancier le patron qu'aux instants pertinents d'un enregistrement (cf. §2.1.4.1.2).

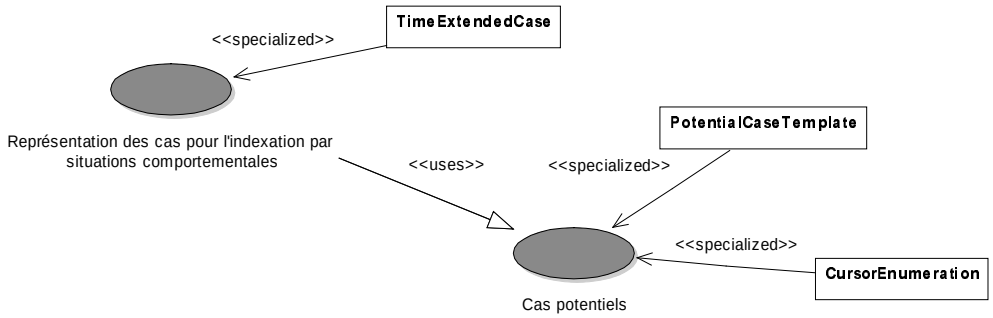


Figure V-17 : utilisation des points d'ouverture pour les cas

2.2.3 Organisation de la mémoire

L'organisation de la mémoire se décompose d'une part en l'organisation de l'indexation et l'organisation des cas. Pour l'organisation de l'indexation (cf. Figure V-19), une unique stratégie d'indexation est déclarée au niveau du point d'ouverture IndexBase qui est alors spécialisé. Cette stratégie utilise un certain nombre d'index simples en les combinant (cf. Figure V-18).

Tout d'abord, l'index s (la racine de l'arbre de composition) permet de déclencher la recherche des cas concrets, et si elle échoue, la phase de recherche de cas potentiels est effectuée. La composition des index pour la recherche des cas concrets permet de combiner en séquence les index simples c1, c2, c3 et c4. Tout d'abord, l'index c1 retourne les cas concrets filtrés à partir du contexte du cas cible. Ensuite, l'index c2 filtre ces cas sur la restriction, et l'index c3 filtre les cas restants sur les comportements élémentaires. Enfin, l'index c4 sélectionne les meilleurs cas en prenant en compte les contraintes temporelles et les similarités intermédiaires données par les index c2 et c3.

La recherche de cas potentiels est effectuée de manière similaire avec les index p1, p2, p3 et p4. Les navigations identifiées par l'index p1 sont parcourues par l'index p2 qui instancie et sélectionne les cas potentiels sur leur restriction. Les cas potentiels retournés, complètement instanciés, sont alors filtrés et sélectionnés (index p3 et p4) de manière identique aux cas concrets.

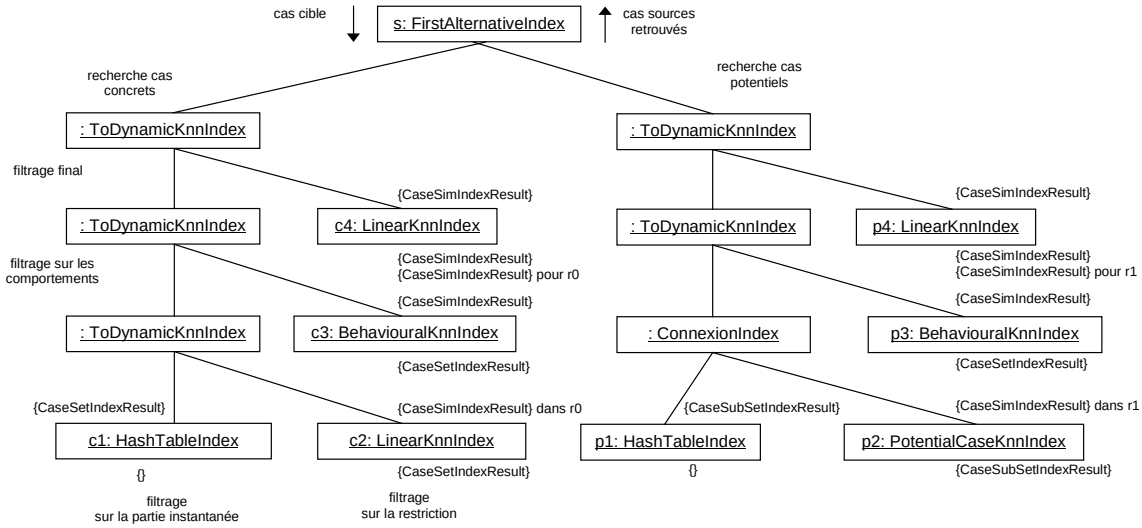


Figure V-18 : utilisation des index lors d'une recherche des cas sources

La réalisation de cette stratégie utilise tout d'abord l'index standard de classe FirstAlternativeIndex pour réunir l'indexation des cas concrets et des cas potentiels. Puis il est nécessaire de spécialiser des index à base de table de hachage (HashTableIndex) pour définir les notions spécifiques de clefs et

d'éléments (index c1 et p1). Les index c2 et p2 utilisent un filtre d'indice qui construit dynamiquement la représentation de la restriction utilisée pour les comparaisons. Les index c3 et p3 sont standards et permettent de construire les composantes comportementales homologues de chaque cas à comparer. Enfin, les index c4 et p4 sont des versions spécialisées de l'index par plus proches voisins pour prendre en compte les résultats intermédiaires de mise en correspondance.

Les similarités utilisées par ces index doivent présenter une structure identique aux indices (spécialisation du point d'ouverture Similarity). Ces mesures utilisent des fonctions élémentaires standards de similarité sauf pour l'adresse de pages Web et le ratio d'affichage (spécialisation du point d'ouverture ValueSimilarityFct pour réaliser les fonctions s_{Adresse} et $s_{\text{Affichage}}$).

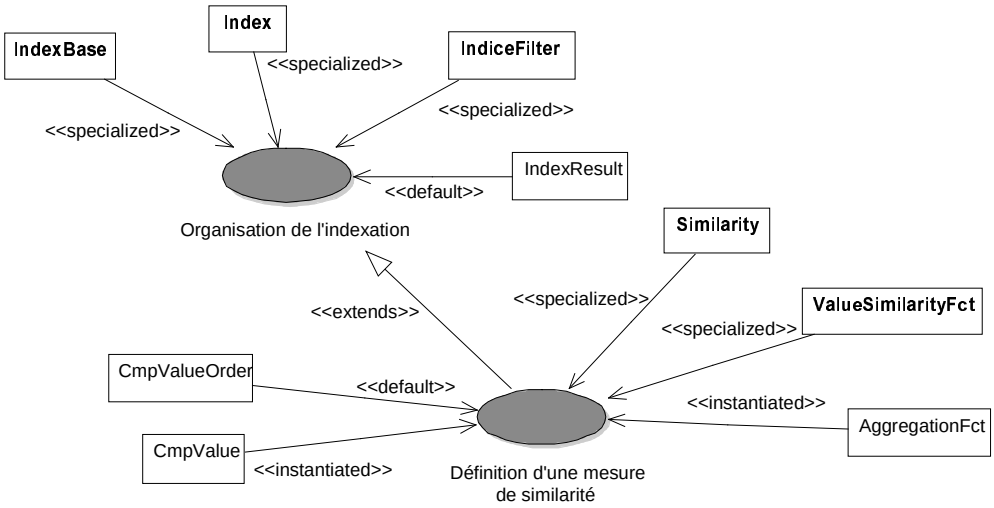


Figure V-19 : utilisation des points d'ouverture pour l'organisation de l'indexation

Enfin, pour l'organisation des cas (cf. Figure V-20), le point d'ouverture CaseBase est instancié pour utiliser la base de cas composée, formée d'une première base de cas pour les cas potentiels et d'une seconde pour les cas concrets (cf. chapitre IV, §2.2). Le point d'ouverture TemplateBase est finalement spécialisé dans l'unique but de déclarer le patron de cas potentiels utilisé par Broadway.

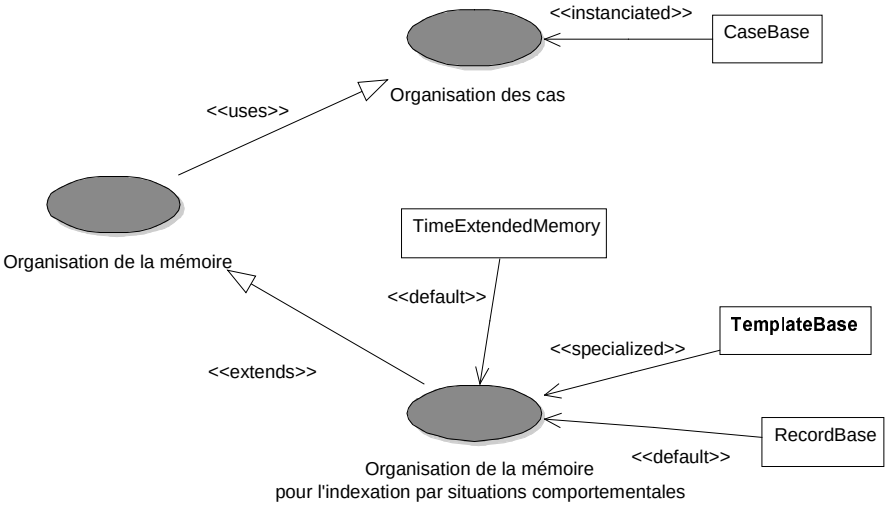


Figure V-20 : utilisation des points d'ouverture pour l'organisation des cas et de la mémoire

2.3 Bilan

La modélisation et la réalisation de Broadway nous a permis d'évaluer concrètement nos propositions du modèle d'indexation par situations comportementales, et de la plate-forme à objets CBR*Tools.

2.3.1 Modèle d'indexation par situations comportementales

L'utilisation de notre modèle de représentation dans le cadre de Broadway montre premièrement la généricité des concepts introduits. Deuxièmement, nous montrons l'apport de notre modèle pour la mise en évidence de connaissances.

2.3.1.1 *Généricité du modèle de représentation*

Pour les données d'observation, nous avons représenté les navigations des utilisateurs (enregistrements) qui sont construites avec quatre variables : adresse, contenu, évaluation et ratio d'affichage. L'évolution de ces variables est représentée par des chroniques échantillonnées pour lesquelles l'unité de temps est le changement d'une page. Enfin, une navigation possède un contexte donnant les mots clefs et les sites traversés. Nous donnons dans le Tableau V-2 le récapitulatif de ces choix de représentation tout en rappelant l'utilisation du modèle dans le cadre de la nutrition des plantes.

Concepts du modèle	*Exemple pour la nutrition des plantes*	*Utilisation dans Broadway*
Variables	rayonnement solaire, température, humidité, écart du drainage, irrigation	adresse, contenu, évaluation, ratio d'affichage
Chronique	échantillonnées	échantillonnées
Enregistrement	expérimentation	navigation
Contexte	consigne (objectif)	mots clefs et sites Web (synthèse de la navigation)

Tableau V-2 : utilisation des concepts pour la représentation des données d'observation

Concepts du modèle	*Exemple pour la nutrition des plantes*	*Utilisation dans Broadway*
composante instantanée	contexte	contexte
comportement élémentaire	séquence de valeurs (évolution)	caractéristique des pages visitées
contrainte de localisation	Ψ_P (par position), Ψ_V (par valeur)	Ψ_B (par valeur sur l'adresse ou le contenu), Ψ_R(par référence)
contrainte temporelle	relation d'occurrence des événements et des séquences de valeurs	précédence des pages visitées
patron de cas potentiels	deux patrons suivant les conditions climatiques	un seul patron sélectionnant des pages passées comme comportements et des pages futures comme solution

Tableau V-3 : utilisation des concepts pour la représentation des cas

Pour la représentation des cas et des situations comportementales (cf. Tableau V-3), nous avons mis en pratique le concept de situation instantanée qui représente le contexte de la navigation. Pour la composante comportementale, nous avons pu sélectionner les pages passées caractérisant le

comportement des utilisateurs avec des comportements élémentaires. Le formalisme que nous avons utilisé nous a également permis de définir une nouvelle contrainte de localisation (Ψ_B) pour identifier de manière spécifique des pages homologues dans un cas cible. Les contraintes temporelles ont été utilisées pour ordonner les pages sélectionnées. Enfin, nous avons défini un patron de cas potentiels pour l'extraction automatique d'expériences pertinentes passées pour guider le raisonnement.

2.3.1.2 *Mise en évidence de cas : vers la découverte des connaissances*

La réalisation concrète de Broadway nous a permis de valider les mécanismes proposés pour les deux premières étapes du processus de découverte des connaissances (cf. chapitre II, §1.1.2.2 et §3) :

1. *Extraction de cas potentiels à partir des données brutes*. L'extraction d'enseignements est effectuée grâce à l'instanciation d'un *patron de cas potentiels*. Ce patron permet d'extraire à partir des navigations, des enseignements liés à un groupe d'utilisateurs. Cette extraction utilise deux hypothèses principales pour caractériser les comportements des utilisateurs : importance des toutes dernières pages visitées, et importance des pages passées évaluées explicitement ou implicitement sur un critère lié au temps d'affichage. Le patron permet alors de structurer en comportements élémentaires et en contraintes temporelles les indices effectifs des cas potentiels identifiés.
2. *Mise à jour des cas concrets*. Un fois identifiés, les cas potentiels jugés utiles par les décisions d'apprentissage sont stockés en mémoire sous la forme de cas concrets. Cette mémorisation n'est effectuée que lorsque les cas concrets déjà identifiés ne sont pas suffisants. Les cas concrets sont alors mis à jour suite aux différents raisonnements qui les réutilisent : nombre de remémorations, nombre de confirmations. Ainsi, des informations liées à la qualité de chaque cas sont calculées et sont utilisées pour guider les réutilisations.

Nous n'avons pas abordé l'interprétation des cas par analyse des indices. Toutefois, nous pouvons aisément supposer que l'étude des cas concrets est facilitée par leur structure explicite associant une liste de pages passées sélectionnées, à un ensemble de pages recommandées. Ainsi, l'interprétation des cas montrant un fort degré de qualité (nombre de confirmations sur le nombre de remémorations) pourrait mener à l'identification de navigations typiques pour mieux comprendre le processus de navigation.

Toutefois les mécanismes proposés impliquent la création de nombreux cas concrets. Ces créations sont rendues possibles car chaque cas concret demande peu de mémoire pour sa représentation, étant donné que les informations coûteuses sont partagées dans les navigations. De plus, comme nous l'avons montré, la recherche s'effectue par filtrages successifs pour maintenir l'efficacité du système. Enfin, ces mécanismes ont été conçus dans la perspective de l'analyse des cas concrets pour créer des cas abstraits. Ces derniers permettraient de généraliser et de remplacer des cas concrets pour réduire la taille de la base de cas. Grâce à notre modèle d'indexation, la création de ces cas abstraits bénéficierait du travail réalisé dans l'identification et la mise à jour des cas concrets.

2.3.2 Plate-forme à objets CBR*Tools

Nous évaluons l'utilisation de CBR*Tools suivant deux critères. Dans le premier critère, nous estimons le degré d'expertise nécessaire de la plate-forme pour réaliser Broadway. De manière générale, une plate-forme demandant peu d'expertise est plus facile à utiliser. Dans le second critère, nous estimons plus concrètement la réutilisation de la modélisation et du code pour Broadway. L'objectif d'une plate-forme est de maximiser cette réutilisation et de limiter la création de nouvelles classes.

2.3.2.1 *Evaluation du degré d'expertise nécessaire de CBR*Tools pour Broadway*

L'estimation du degré requis d'expertise, relatif au fonctionnement de notre plate-forme pour la réalisation de Broadway, repose sur la complexité des connaissances nécessaires à chaque type d'utilisation d'un point d'ouverture :

- une spécialisation demande en général une compréhension approfondie du point d'ouverture,
- une instanciation demande uniquement la connaissance des paramètres et du comportement global du composant réalisant un point d'ouverture,
- une utilisation par défaut reste transparente pour l'utilisateur.

Nous rappelons que, si un point d'ouverture est utilisé suivant différentes méthodes, nous ne retenons que celle qui demande le plus d'expertise. Cette estimation montre que la réalisation de Broadway nécessite une connaissance approfondie d'uniquement 50% des points d'ouverture de la plate-forme (cf. Tableau V-4).

Type d'utilisation	*Axe « gestion du raisonnement »*	*Axe « Représentation des cas »*	*Axe « Organisation de la mémoire »*	*Total*
Spécialisation	5	8	6	*19 (50%)*
Instanciation	3	4	3	*10 (26%)*
Par défaut	3	2	4	*9 (24%)*
Total	*11*	*14*	*13*	*38 (100%)*

Tableau V-4 : utilisation des points d'ouverture de CBR*Tools

Parmi les spécialisations effectuées, cinq d'entre elles ne sont pas considérées obligatoires (cf. chapitre IV, §3.2.2.1) mais ont été nécessaires car il n'existait pas de valeurs par défaut ou de composants utilisables par instanciation :

- chaînage des états des raisonnements associés à une navigation (Reasoning),
- contrainte de localisation spécifique Ψ_B (SequenceLocation),
- parcours des enregistrements adapté au patron de cas potentiels (CursorEnumeration),
- configuration des index (index),
- ajout de fonctions élémentaires de similarité (ValueSimilarityFct).

2.3.2.2 *Evaluation de l'aide apportée par CBR*Tools pour la réalisation de Broadway*

Sur le plan de la réalisation, nous estimons l'apport de CBR*Tools en termes de modélisation et de réutilisation de code. Dans ce but, nous partageons les classes nécessaires à Broadway en quatre types (cf. Tableau V-5) :

1. les classes créées uniquement par spécialisation d'interfaces de CBR*Tools,
2. les classes créées par spécialisation de classes abstraites ou concrètes de CBR*Tools,
3. les classes existantes de CBR*Tools utilisées uniquement par instanciation, et
4. les classes créées spécifiquement qui n'ont pas de lien de spécialisation avec des classes existantes.

Le nombre de spécialisations (1+2) reste toujours supérieur ou égal au nombre de points d'ouverture spécialisés, car un même point d'ouverture peut être spécialisé à plusieurs reprises. Par exemple dans Broadway, la configuration des index nécessite trois spécialisations (configuration des deux tables de hachage et de l'algorithme par plus proches voisins pour le filtrage final). Le nombre de classes différentes utilisées par instanciation directe est également supérieur ou égal au nombre de

points d'ouverture instanciés. En effet, un même point d'ouverture peut être instancié pour des raisons différentes en sélectionnant plusieurs sous-classes. De plus, ce groupe de classes recouvre aussi : les classes des points d'ouverture spécialisés mais aussi instanciés (6 classes différentes sont également instanciées pour le point d'ouverture Index), les classes annexes utilisées pour les paramètres des points d'ouverture, et les classes des points d'ouverture utilisés par défaut (si la valeur par défaut n'est pas null). Enfin, dans Broadway, seules les classes nécessaires à la représentation du schéma de réutilisation sont nouvelles.

Types de classes	***Axe « gestion du raisonnement »***	***Axe « Représentation des cas »***	***Axe « Organisation de la mémoire »***	***Total***
Spécialisation d'interfaces	2	3	3	*8 (11%)*
Spécialisation de classes abstraites ou concrètes	3	11	8	*22 (30%)*
Instanciation	5	13	17	*35 (48%)*
Création	8	0	0	*8 (11%)*
Total	*18*	*27*	*28*	*73 (100%)*

Tableau V-5 : types de classes utilisées dans Broadway

En définitive, CBR*Tools facilite la modélisation d'une application comme Broadway grâce à son architecture dans laquelle les comportements ont déjà été analysés et assignés à différentes classes (89% des classes sont issues de spécialisation ou d'instanciation de classes existantes). De plus, des parties algorithmiques plus lourdes sont réutilisées, principalement dans les index et les similarités (78% des classes réutilisent du code à travers des spécialisations de classes concrètes ou abstraites, et des instanciations).

3 Proposition d'un nouveau type d'assistants à la navigation

Avec notre système Broadway, nous introduisons un nouveau type d'assistants à la navigation. Nous proposons en effet la réutilisation de *comportements observés par variables* pour calculer les recommandations. Les assistants de navigation existants ne permettent pas une observation aussi détaillée et flexible. De plus, le cadre générique formé par notre modèle d'indexation nous permet d'aborder des problèmes d'assistance dans des contextes différents. Dans cette section, nous détaillons tout d'abord une évaluation expérimentale de notre prototype Broadway, menée avec des utilisateurs dans un contexte précis de recherche d'informations. Puis, nous présentons l'originalité de Broadway par rapport aux approches existantes.

3.1 Evaluation expérimentale de Broadway

Une expérimentation avec des utilisateurs réels a été menée par des étudiants en DESS d'ergonomie pour évaluer si Broadway assiste effectivement les utilisateurs dans leurs tâches de recherche d'informations sur le Web (Hébraud *et al.*, 1998). Nous introduisons tout d'abord les objectifs et la procédure d'expérimentation puis nous présentons les résultats de cette expérimentation. Enfin, nous donnons la configuration et l'adaptation de Broadway qui a été nécessaire pour cette expérimentation.

3.1.1 Objectifs et procédure d'expérimentation

La procédure d'expérimentation a été déterminée en toute autonomie par des élèves ergonomes qui ont reçu au préalable une formation d'une journée à l'utilisation et aux concepts de Broadway. Pour cette expérimentation, dix étudiants en psychologie ont été choisis pour jouer le rôle d'utilisateurs de Broadway qui veulent rechercher des informations sur le Web. Deux groupes dont les connaissances du Web et des navigateurs sont équivalentes, ont alors été formés : le premier groupe aura accès aux recommandations de Broadway contrairement au deuxième. Toutefois, tous les sujets devront systématiquement évaluer les pages visitées avec la barre d'outils de Broadway[55]. Tous les sujets auront alors le même objectif de recherche :

« *Vous devez rechercher des informations sur la vie de Patrick Sommier. Vous savez qu'il s'agit d'un artiste contemporain qui travaille probablement dans le monde du théâtre* »

Cet objectif a été choisi avec soin : aucun des sujets n'avait de connaissance préalable sur cet artiste et sa biographie n'était pas directement accessible depuis les cinq premiers liens donnés par le moteur de recherche Altavista pour le mot clef « spectacles ». Ces cinq liens constitueront les points de départ des navigations des sujets. Des tests préliminaires ont montré que l'accès à l'information demandée était accessible sans assistance depuis ces liens en 30 minutes environ. Ainsi, le dépassement de ce délai sera considéré comme un échec de la recherche. Une base de cas a également été construite au préalable avec des exemples de navigations réussies. L'objectif principal de cette expérimentation est de comparer la qualité des navigations entre le groupe utilisant Broadway et celui sans. Plus précisément, cinq hypothèses sont à évaluer (Hébraud *et al.*, 1998) :

1. Les sujets de la condition « Avec Broadway » ont un taux de réussite à la tâche supérieur à ceux de la condition « Sans Broadway » (indicateur : nombre de réussites).
2. Les sujets de la condition « Avec Broadway » retrouve l'information plus rapidement que ceux de la condition « Sans Broadway » (indicateur : temps total).
3. Les sujets de la condition « Avec Broadway » visitent moins de pages Web que ceux de la condition « Sans Broadway » (indicateur : nombre de pages visitées).
4. Les évaluations portées sur les pages Web sont homogènes pour les deux groupes (indicateur : évaluations affectées à une même page par tous les sujets).
5. Les sujets de la condition « Avec Broadway » sont globalement plus satisfaits que ceux de la condition « Sans Broadway » (indicateur : somme des évaluations sur le nombre de pages évaluées).

3.1.2 Résultats de l'expérimentation

Les résultats de l'expérimentation sur la vérification des trois premières hypothèses sont donnés dans le Tableau V-6. Nous constatons que, dans le contexte choisi, l'utilisation de Broadway a effectivement facilité la tâche de recherche d'information : il y a plus de succès, la durée des navigations est réduite et leur longueur également.

Pour l'hypothèse 4, l'expérimentation a permis de montrer que les évaluations ne sont pas cohérentes dans certains cas. En effet, bien que l'objectif de recherche ait été précisément défini, il semble que certains sujets aient évalué une page en se basant sur le thème général abordé, alors que d'autres ont évalué positivement uniquement les pages ayant directement un rapport avec la requête. Toutefois, il y a un consensus sur l'évaluation la plus positive. Pour pallier ce problème, nous pensons étudier certaines modifications pour une version future de Broadway : réduction du nombre de degrés d'évaluation, définition plus claire de la sémantique des évaluations, ou normalisation des évaluations

[55] L'évaluation systématique n'est pas requise dans une utilisation courante de Broadway.

en tenant compte des pages évaluées le plus positivement. Enfin, l'expérimentation n'a pu valider l'hypothèse 5 au vu du nombre restreint d'utilisateurs, bien que les résultats bruts montrent une satisfaction globale plus grande pour les sujets ayant bénéficié de Broadway.

Critères d'évaluation	***Avec Broadway (4 sujets)***	***Sans Broadway (6 sujets)***
Nombre de succès	75% (3/4)	33% (2/6)
Durée des navigations réussies (moyenne)	18 min	24 min
Longueur des navigations réussies (moyenne)	19 pages	39 pages

Tableau V-6 : synthèse des résultats pour les trois premières hypothèses

Cette expérimentation a permis une première évaluation de Broadway par des utilisateurs qui n'étaient pas liés au développement du système. Les résultats confirment la validité de notre approche dans les conditions restreintes de l'expérimentation, mais d'autres tests restent nécessaires pour les généraliser.

3.1.3 Configuration de Broadway utilisée

Durant l'expérimentation, les sujets localisés à l'université de Provence à Aix-en-Provence ont accédé pendant plusieurs jours au système Broadway situé à l'INRIA Sophia Antipolis au travers du réseau Internet. Même si deux groupes avaient été créés et que seulement l'un d'entre eux eut accès aux recommandations, Broadway fut nécessaire dans les deux cas pour observer les comportements des sujets et pour leur permettre d'évaluer les pages. Pour ne pas engendrer des conflits entrent les deux groupes au niveau des données enregistrées et de la base de cas générée, deux systèmes Broadway avaient été lancés simultanément. Au moment de l'expérimentation, le prototype de Broadway n'intégrait pas l'ensemble des mécanismes que nous avons décrits dans ce chapitre. Notamment, le serveur de recommandations utilisait uniquement des cas potentiels, et les fonctions d'analyse de la base de cas pour gérer l'obsolescence des navigations et des cas n'avaient pas été implantées. L'analyse des pages Web était également plus simple et ne disposait pas des fonctions de filtrage des mots non significatifs et de la réduction des mots à leur racine.

Pour réaliser l'expérimentation, les ergonomes chargés de l'étude avaient exprimé le besoin de pouvoir analyser finement les navigations des sujets mais aussi l'activité du serveur de recommandations et les choix de recommandations effectués. Aucun module n'avait jusqu'alors été prévu pour ce type d'analyse, cependant l'architecture ouverte de Broadway nous a facilement permis d'ajouter de tels modules. Deux modules ont alors été conçus et implantés : un *traceur* et un *enregistreur*. Ces deux modules sont des processus distants qui se connectent tout simplement au répartiteur d'événements lors du lancement de Broadway.

Le traceur permet de sauvegarder dans un fichier les événements au fur et à mesure qu'ils se produisent, dans un format brut qui peut être analysé directement sous le tableur Excel par exemple. Chaque ligne du fichier donne le nom du module émetteur, la date, le nom de l'utilisateur, la classe de l'événement et les informations caractérisant le contenu de l'événement. L'extrait présenté dans la Figure V-21 permet de retracer l'activité de l'utilisateur et du système de la façon suivante. Tout d'abord, la barre d'outils d'un utilisateur nommée baseErgo commence par envoyer une évaluation. Puis, suite à la sélection d'un hyperlien dans le navigateur, le proxy charge une nouvelle page.

L'ancienne page dans le navigateur est alors effacée et remplacée par une nouvelle : l'observateur (*matcher*) envoie alors deux événements pour l'indiquer. Le serveur de recommandations (*recommender*) activé par l'affichage d'une nouvelle page commence son raisonnement puis envoie un événement indiquant les résultats. Enfin, l'utilisateur sélectionne une des recommandations dans le contrôleur (*manager*).

```
toolbar;13/03/1998 15:22:09;baseErgo;broadway.events.PageEvaluationEvent;
        url=http://www.grandtheatre.qc.ca/pour_enfants.html, eval=1
proxy;13/03/1998 15:27:51;baseErgo;broadway.events.PageEvent;
        url=http://www.grandtheatre.qc.ca/programmation_chrono.html, state=LOADED
watcher;13/03/1998 15:24:16;baseErgo;broadway.events.PageEvent;
        url=http://www.grandtheatre.qc.ca/pour_enfants.html, state=STOP
watcher;13/03/1998 15:24:22;baseErgo;broadway.events.PageEvent;
        url=http://www.grandtheatre.qc.ca/programmation_chrono.html, state=START
recommender;13/03/1998 15:28:00;baseErgo;broadway.events.AdviceStartEvent;state=START
recommender;13/03/1998 15:28:03;baseErgo;broadway.events.AdviceResultEvent;
        state=RESULT, duration=2667, nb=2,
        url0=http://www.brookes.ac.uk/schools/sol/home/index.htm, rate0=0.44375,
        url1=http://www.brookes.ac.uk/~p0030818/cal/1674.html, rate1=0.44375
manager;13/03/1998 15:25:04;baseErgo;broadway.events.AdviceSelectionEvent;
        url=http://www.brookes.ac.uk/schools/sol/home/index.htm
```

Figure V-21 : extrait de l'enregistrement des événements[56]

```
Navigation from baseErgo

size: 20
start date: 13/03/1998 15:20:23
end date: 13/03/1998 15:34:09

1. URL = http://www.pariserve.tm.fr/sortir/dinersp.htm

        ○ keywords = DINERS, SPECTACLES
        ○ size = 13714
        ○ display = 1920
        ○ ratio = 0.14000292
        ○ eval = 0
        ○ selected = –
        ○ advice :

[....]

5. URL = http://www.grandtheatre.qc.ca/programmation_chrono.html

        ○ keywords = PROGRAMMATION, CHRONOLOGIQUE
        ○ size = 31111
        ○ display = 22360
        ○ ratio = 0.71871686
        ○ eval = 0
        ○ selected = http://www.brookes.ac.uk/schools/sol/home/index.htm
        ○ advice :
            □ 0.44375 http://www.brookes.ac.uk/schools/sol/home/index.htm
            □ 0.44375 http://www.brookes.ac.uk/~p0030818/cal/1674.html
```

Figure V-22 : extrait du fichier généré par l'enregistreur pour les événements de la Figure V-21

Contrairement au traceur, l'enregistreur effectue un traitement des événements en les regroupant par utilisateur et en les présentant de manière plus claire. L'enregistreur permet en effet de générer la synthèse d'une navigation dans un fichier HTML lorsqu'une navigation est fermée (cf. Figure V-22). Ainsi, la taille de la navigation, ses dates de début et de fin, et toutes les pages visitées sont facilement

[56] Les informations sont ici formatées avec des retours à la ligne pour plus de clarté, mais en réalité chaque événement n'est représenté que par une seule ligne.

accessibles. De plus, pour chaque page, différentes informations sont disponibles : URL, mots clefs, taille en octets, temps d'affichage en millisecondes, ratio d'affichage, évaluation (de –1 à 2), recommandation sélectionnée et liste des recommandations obtenues.

Ces deux modules sont complémentaires : alors que l'enregistreur est dépendant de la création d'une navigation et propose une vue synthétique mais réduite d'une navigation, le traceur enregistre tous les événements même lorsqu'une navigation n'a pas été démarrée. Ces deux modules ont été nécessaires au dépouillement des données[57]. Pour l'expérimentation, tous les fichiers créés par le traceur et l'enregistreur ont été placés automatiquement dans l'espace public du proxy utilisé dans Broadway et qui opère dans ce cas comme un serveur HTTP classique. Ainsi, les personnes chargées du dépouillement ont pu librement accéder aux données distantes avec leur navigateur Web.

La réalisation de ces deux modules montre enfin la stabilité de l'architecture ouverte choisie au regard de spécifications qui évoluent. Ces évolutions sont naturelles dans un contexte réel d'expérimentation et correspondent exactement aux contraintes que nous voulions prendre en compte.

3.2 Comparaison avec les systèmes existants d'aide à la navigation

Nous classons les différents assistants existants de navigation en trois grands groupes suivant les techniques utilisées :

- raisonnement à partir de cas : Radix (Corvaisier *et al.,* 1997), Hypercase (Micarelli & Sciarrone, 1996) et Hospitext (Elkassar & Charlet, 1997).
- autres techniques d'apprentissage automatique : WebWatcher (Armstrong *et al.,* 1997), PersonnalWebWatcher (Mladenic, 1996), Letizia (Lieberman, 1995), WebMate (Chen & Sycara, 1998).
- statistiques et comparaisons : approche de Yan *et al*. (Yan *et al.,* 1996), WBI (Barrett *et al.,* 1997), FindMe (Burke *et al.,* 1997) et Footprints (Wexelblat & Maes, 1997).

Ces assistants sont récents (les premières publications datent de 1995) et la plupart ont été réalisés de manière concourante à nos travaux. Nous montrons que Broadway met en œuvre une approche originale permettant à la fois la prise en compte de comportements structurés par variables dans la situation de recommandation, et l'apprentissage à partir d'un groupe d'utilisateurs, tout en proposant une architecture ouverte et adaptée au Web. La synthèse de cette comparaison est donnée dans le Tableau V-7.

3.2.1 Situation de recommandation

Un assistant de navigation observe l'utilisateur et peut générer des recommandations, à un instant précis. Trois approches sont possibles pour constituer la situation servant à déterminer les recommandations : la page Web courante et/ou l'intérêt exprimé par l'utilisateur (*contexte local*), un résumé automatique de la navigation et de l'intérêt de l'utilisateur (*profil utilisateur*), ou un ensemble d'observations issues des navigations (*comportements*).

Six assistants utilisent un contexte local : WebWatcher, PersonnalWebWatcher, Footprints, FindMe, WBI et Hospitext. WebWatcher et PersonnalWebWatcher utilisent la page courante ainsi qu'un ensemble de mots clefs donné par l'utilisateur au début de sa recherche. WebWatcher met en valeur les hyperliens recommandés de la page courante. Footprints permet de visualiser un graphe

[57] Le traceur intervient en fait en tant que sauvegarde des événements émis. Cette précaution a été très utile dans la mesure où, suite à des erreurs de manipulation qui ont conduit à ne pas démarrer les navigations à observer, l'enregistreur n'a pu générer les fichiers des navigations.

dans lequel chaque nœud représente une page et les nœuds sont reliés entre eux par des liens représentant les déplacements passés des utilisateurs. De plus, les liens sont dessinés avec différentes couleurs pour montrer leur fréquence d'utilisation. Ainsi, un utilisateur peut visualiser ce graphe pour repérer sa position courante et choisir de suivre un de ces liens en cliquant directement sur le graphe. L'idée soutenue par ce système est qu'un lien est intéressant si un grand nombre d'utilisateurs l'ont suivi. FindMe est un type d'approche permettant de guider l'utilisateur dans la recherche d'éléments précis. Cette technique a été appliquée pour la sélection de voitures, de vidéos, de restaurants ou d'appartements. Lorsqu'un utilisateur débute sa navigation, il indique ses premières contraintes et le système peut alors lui proposer une solution initiale. Puis, l'utilisateur navigue par modifications successives de ces contraintes par rapport à l'élément courant sélectionné. Par exemple, l'utilisateur peut indiquer qu'il cherche un appartement moins cher, plus grand, plus beau… Le système génère alors une nouvelle proposition en effectuant une similarité entre la proposition courante et les éléments existants, tout en prenant en compte la modification demandée. WBI utilise un ensemble de séquences types issues des navigations d'un utilisateur. Il propose alors la page finale d'une séquence dès que l'utilisateur visualise la première page. Enfin, l'objectif d'Hospitext est d'assister différents types d'utilisateurs durant leur consultation de dossiers médicaux de patients qui sont disponibles dans un hypermédia. A partir d'un objectif de navigation, il propose une séquence de documents à visiter.

Deux assistants, Letizia et WebMate, construisent un profil utilisateur à partir des navigations observées et des interactions avec le système. Letizia est un agent qui assiste un utilisateur en navigant en parallèle et de manière autonome depuis la page courante en cours de visualisation. Letizia utilise le temps passé par l'utilisateur à lire la page pour explorer les environs et anticiper ses besoins. Cet assistant construit une description instantanée de l'intérêt de l'utilisateur à partir de ses actions durant la navigation courante (profil utilisateur formé d'un vecteur de mots clefs). WebMate est un agent qui apprend le profil d'un utilisateur suivant un ensemble de pages appréciées lors de ses navigations sur le Web. Un profil est dans ce système constitué d'un ensemble de vecteurs de mots clefs pondérés. Letizia et WebMate sont alors capables de filtrer et de recommander des pages qui dépassent un seuil de similarité avec le profil.

Enfin, trois derniers assistants prennent en compte un ensemble d'événements issus de la navigation de l'utilisateur : Radix, Hypercase et l'approche de Yan *et al*. Radix, que nous avons déjà présenté (cf. chapitre II, §1.1.1.3.5), est basé sur une description structurée des sessions de navigation et utilise une fonction de similarité prenant en compte la succession ordonnée des actions de l'utilisateur (changement de pages) pour calculer les recommandations. Hypercase utilise également l'ensemble des pages déjà visitées comme indice de son raisonnement pour déterminer les recommandations. Enfin, l'approche de Yan *et al*. propose l'utilisation de navigations typiques construites statistiquement. Ainsi, la navigation d'un utilisateur (ensemble de pages visitées) est tout d'abord analysée et classée dans l'une de ces navigations typiques. Puis, le serveur propose les pages de cette navigation qui n'ont pas encore été visitées. Cependant, comme dans Hypercase, l'ordre d'accès aux pages n'est pas pris en compte[58].

Broadway se situe dans cette troisième classe d'assistants et propose plus particulièrement la prise en compte de *comportements observés par variables*. Broadway est en effet basé sur l'observation des comportements structurés en variables : adresse, contenu des pages, évaluation explicite et évaluation implicite. Notre approche est originale car elle ne se limite pas à ces quatre variables, et grâce à notre modèle d'indexation, elle offre un cadre générique pour la prise en compte de tout type de variables. Par exemple, il est possible d'ajouter simplement dans le prototype actuel des variables portant sur

[58] Les auteurs reconnaissent l'importance de prendre en compte cet ordre et l'envisagent dans leurs travaux futurs.

des saisies de formulaire, ou sur l'envoi de requêtes à des moteurs de recherche. De plus, notre approche est également applicable dans d'autres contextes d'assistance à la navigation dans lesquels les variables prennent des sémantiques spécifiques :

- aide à la navigation sur un site de commerce électronique : descriptif textuel d'un article, sélection d'un article dans son panier d'achat,
- aide à la navigation dans une hiérarchie thématique (par exemple Yahoo) : descriptif textuel d'un thème, type de lien hypertexte sélectionné (catégorie transversale, sous-catégorie ou super-catégorie), sélection d'un site.

Toutefois, la prise en compte de ces variables est plus coûteuse qu'une approche par contexte local, par profil utilisateur ou par comportements. Nous mettons alors en œuvre notre modèle d'indexation qui permet de gérer efficacement les situations comportementales (filtrage progressif lors de la recherche et partage des données brutes entre les cas concrets).

Assistants de navigation	***Situation de recommandation***	***Apprentissage***	***Architecture***		
			indépendance navigateur	*assistance multi-sites*	*ouverture*
FindMe (Burke *et al.*, 1997)	contexte local	pas d'apprentissage	✓	-	✓
Footprints (Wexelblat & Maes, 1997)	contexte local	groupe	✓	-	-
Hospitext (Elkassar & Charlet, 1997)	contexte local	groupe (RàPC)	✓	-	-
WBI (Barrett *et al.*, 1997)	contexte local	unique utilisateur	✓	✓ (proxy)	✓
WebWatcher (Armstrong *et al.*, 1997)	contexte local	groupe	✓	✓ (redirection des liens)	✓
PersonalWebWatcher (Mladenic, 1996)	contexte local	unique utilisateur	✓	✓ (proxy)	✓
Letizia (Lieberman, 1995)	profil utilisateur	unique utilisateur	✓	✓	-
WebMate (Chen & Sycara, 1998)	profil utilisateur	unique utilisateur	✓	✓ (proxy)	-
Hypercase (Micarelli & Sciarrone, 1996)	comportements	pas d'apprentissage (RàPC)	✓	-	-
Radix (Corvaisier *et al.*, 1997)	comportements	groupe (RàPC)	-	✓	-
Yan *et al* (Yan *et al.*, 1996)	comportements	groupe	✓	-	-
Broadway	**comportements observés par variables**	**groupe (RàPC)**	✓	✓ (proxy)	✓

Tableau V-7 : comparatif des assistants de navigation existants avec Broadway

3.2.2 Apprentissage

Un assistant de navigation met à jour ses connaissances grâce à un mécanisme d'apprentissage. Trois types d'apprentissage existent : le cas limite où il n'y a *pas d'apprentissage*, l'apprentissage centré sur un *unique utilisateur* et l'apprentissage à partir d'un *groupe* d'utilisateurs.

Hypercase est un assistant qui utilise des techniques de raisonnement à partir de cas. Cependant, cet assistant ne peut pas apprendre et s'enrichir à partir des expériences des utilisateurs, puisqu'il utilise uniquement un ensemble de cas prédéfinis construits par des experts. FindMe ne met pas non plus à jour ses connaissances suivant les comportements des utilisateurs, seuls de nouveaux éléments (voitures, vidéos, appartements…) peuvent être ajoutés dans sa base.

Letizia, WebMate, PersonnalWebWatcher et WBI ne prennent en compte qu'un seul utilisateur. Letizia et WebMate apprennent le profil de leur unique utilisateur. Contrairement à WebMate, Letizia n'apprend pas à améliorer ses recommandations au cours des différentes navigations, puisque le profil n'est pas sauvé ni mis à jour. PersonnalWebWatcher a pour but explicite d'assister un unique utilisateur. WBI enregistre les navigations d'un utilisateur et les examine pour en extraire des séquences qui se produisent fréquemment. Ce fonctionnement permet alors de mettre en évidence des raccourcis dans les navigations d'un utilisateur.

Enfin, les autres approches mettent à profit les navigations d'un groupe d'utilisateurs pour améliorer leur processus de recommandation. Radix enregistre de nouveaux cas dans sa base. Hospitext nécessite le recueil de navigations commentées par les utilisateurs pour connaître le type des informations recherchées à chaque choix d'un lien. Puis, ce système est doté d'un module d'analyse pour extraire automatiquement, à partir de ces observations, des navigations abstraites liant des types de documents et des types d'éléments structurels, plutôt que des documents en absolu. Les navigations sont ainsi réutilisables dans différents contextes. WebWatcher suit un groupe d'utilisateurs au cours de leurs navigations et tente d'apprendre la fonction suivante avec différentes méthodes :

$$\text{QualitéLien} : \text{Page} \times \text{Intérêt} \times \text{Lien} \rightarrow [0 \ldots 1]$$

Footprints analyse les logs HTTP d'un serveur pour former le graphe de synthèse des navigations effectuées par les utilisateurs. Enfin, l'approche de Yan *et al*. analyse également les logs d'un serveur HTTP pour construire statistiquement des navigations typiques (*clustering*).

Comme ces derniers travaux, Broadway met à jour ses connaissances à partir des comportements d'un groupe d'utilisateurs (enregistrement et réutilisation de cas provenant d'utilisateurs différents, cf. §2.1.4.4). Les cas représentent des connaissances issues d'utilisateurs différents et sont réutilisés durant le raisonnement.

3.2.3 Architecture : indépendance des navigateurs, assistance multi-sites et ouverture

Sur le plan de l'architecture des assistants, trois critères sont importants dans le cadre de nos contraintes d'application : *l'indépendance* vis-à-vis des navigateurs utilisés, *l'assistance multi-sites* sur le Web et *l'ouverture* du système pour modifier son comportement.

Dans Radix, l'observation des actions de l'utilisateur, comme la sélection d'un signet, la spécification manuelle d'une adresse, l'utilisation des boutons de navigation (« précédant » et « suivant »), est nécessaire pour représenter une navigation et ses composants. C'est pourquoi ce système utilise un navigateur avec des fonctions personnalisées : son utilisation est alors restreinte à une plate-forme logicielle spécifique. Les autres assistants présentés peuvent être utilisés avec les navigateurs classiques.

Les systèmes existants visent, soit une assistance générale du côté de l'utilisateur, soit une assistance limitée à un serveur spécifique. Radix, Letizia, WebWatcher, PersonalWebWatcher, WBI et WebMate font partie de la première catégorie et permettent une assistance multi-sites. Ces assistants reposent alors généralement sur une architecture de serveur proxy (PersonalWebWatcher, WBI et WebMate). WebWatcher est implanté sous la forme d'un serveur HTTP qui analyse et modifie tous les liens des pages visitées pour les rediriger vers le serveur lui-même. Ce procédé permet de suivre les utilisateurs durant leurs navigations tout en présentant un fonctionnement similaire à un proxy (mais plus coûteux). D'un autre côté, Hypercase, Hospitext, FindMe, Footprints et l'approche de Yan *et al*. sont liés à un hypermédia particulier. Hypercase nécessite en effet la définition préalable de chaque page, ce qui n'est pas adapté au Web qui est par nature dynamique et sans bornes. Hospitext utilise des connaissances du domaine et plus particulièrement une taxonomie des pages pour effectuer son apprentissage. Cette approche est difficile à mettre en pratique sur le Web en général puisqu'une telle hiérarchie n'existe pas[59]. FindMe est une approche servant à concevoir un serveur dans un domaine bien spécifique (recherche de vidéos par exemple). Footprints et l'approche de Yan *et al*. utilisent les logs d'un serveur, et sont donc fortement liés à celui-ci.

Enfin, certains assistants mettent en avant une architecture permettant de changer et d'évaluer différentes stratégies d'assistance ou de fonctionnement. WebWatcher et PersonalWebWatcher peuvent utiliser différentes techniques d'apprentissage. FindMe définit une approche applicable dans différents contextes. WBI offre également une architecture modulaire dans laquelle différents agents peuvent collaborer.

Broadway est indépendant du navigateur utilisé, et peut assister les utilisateurs sur n'importe quel site du Web grâce à une architecture de proxy. Il est enfin très ouvert : il repose sur une architecture répartie intégrant un ensemble extensible de modules et il utilise notre plate-forme CBR*Tools pour réaliser le serveur de recommandations.

4 Conclusion

Broadway est un système d'assistance à la navigation sur le Web, motivé par la difficulté de la recherche d'informations sur ce vaste hypermédia. Broadway est tout d'abord une application concrète qui nous a permis d'évaluer notre modèle d'indexation par situations comportementales ainsi que notre plate-forme à objets CBR*Tools. Nous montrons ainsi que notre modèle d'indexation permet la représentation des navigations ainsi que l'extraction des cas potentiels et leur transformation en cas concrets. Sur le plan de la réalisation, CBR*Tools nous a offert un cadre cohérent dans lequel nous avons spécialisé les objets gérant notre modèle d'indexation puis assemblé les phases du raisonnent pour former le système complet.

Grâce à Broadway, nous introduisons également un nouveau type d'assistants à la navigation. Nous proposons en effet la réutilisation de *comportements observés par variables* pour calculer les recommandations. Les assistants de navigation existants utilisent, soit le contexte local (dernière page visitée et parfois les objectifs de recherche), soit un profil utilisateur (résumé de la navigation en termes de mots clefs), soit des comportements simples (séquence ou ensemble de pages). Dans Broadway, quatre variables sont utilisées : adresse, contenu, évaluation explicite et évaluation implicite. Cette observation par variables, combinée avec notre modèle d'indexation, permet la gestion de comportements plus détaillés dans un cadre flexible et générique. L'évaluation

[59] Ce type d'approche pourrait être viable sur le Web si l'utilisation des métadonnées (informations liées aux pages) était normalisée et généralisée.

expérimentale de Broadway réalisée avec des utilisateurs réels montre la faisabilité et l'intérêt d'une telle approche.

Broadway est enfin un premier prototype démontrant notre approche. D'autres systèmes similaires mais basés sur des variables différentes, sont en cours d'étude avec notre participation : assistance d'un grand groupe d'utilisateurs sur un hypermédia délimité[60] et assistance dans une tâche de recherche combinant des requêtes et des navigations[61].

[60] Cf. contrat de recherche avec le CTI-CNET (centre de recherche de France Télécom) de 1998 à 2000.
[61] Cf. collaboration entre l'INRIA et le centre de recherche de Xerox en 1998.

Conclusion

Dans notre étude, nous avons traité trois thèmes de recherche encore très peu abordés :

- la représentation d'indices pour le RàPC dans le cadre d'une classe de problèmes,
- la modélisation d'outils ouverts pour le RàPC,
- la réutilisation de comportements pour l'assistance à la navigation sur le Web.

Tout d'abord, nous présentons notre contribution pour chaque thème en faisant le bilan de la démarche suivie. Nous ébauchons alors les trois principales perspectives qui montrent la portée de nos travaux.

1 Bilan

1.1 Modèle d'indexation par situations comportementales

Durant toute notre étude, nous avons été motivés par une classe de problèmes dont la principale caractéristique est de nécessiter la prise en considération d'historiques dans la formulation du problème à résoudre. Nous avons alors étudié le problème de l'indexation qui est grandement affecté par cette spécificité au sein du cycle de raisonnement à partir de cas. Nous avons présenté dans le chapitre II, l'état de l'art des approches d'indexation par historiques et nous avons proposé le *modèle d'indexation par situations comportementales*. Aucun des travaux existants n'aborde de manière générale l'indexation par historiques dans le cadre d'un cycle de raisonnement à partir de cas. Notre modèle d'indexation isole cette problématique en deux parties : modèle de représentation des indices et des cas, guide d'utilisation. Dans ce modèle, nous offrons une modélisation générique pour faciliter l'analyse du RàPC dans le cadre de cette classe de problèmes, et pour constituer une compétence métier nécessaire pour aborder des problèmes complexes.

Notre modèle d'indexation permet également de dépasser les deux principales limites des approches existantes afin de permettre une meilleure indexation des cas dans la classe de problèmes visée. Premièrement, nous offrons la possibilité d'exprimer des indices à la fois sur des historiques échantillonnés et à événements. Cette possibilité est importante d'une part pour gérer des processus hybrides, et d'autre part, pour représenter des indices calculés sous forme d'événements, alors que d'autres variables restent échantillonnées par manque de connaissances.

Deuxièmement, nous proposons des solutions originales pour permettre une meilleure gestion du processus de découverte des connaissances. Ce processus est important dans notre classe de problèmes car il n'est pas possible de représenter de manière explicite tous les cas utiles issus des observations, étant donné le manque de connaissances et la quantité importante des données disponibles. L'introduction du concept de cas potentiel permet alors d'extraire des cas à partir des données d'observation. Les cas potentiels sont définis en compréhension au travers de patrons qui peuvent être instanciés. Ainsi, contrairement aux travaux existants, nous permettons l'utilisation d'hypothèses préétablies, représentées dans les patrons de cas potentiels, pour structurer cette extraction. Puis, pour pallier le manque d'apprentissage de la plupart des travaux existants, nous permettons de sauver ces instanciations sous forme de cas concrets. La nature explicite des cas concrets permet de faire évoluer leur représentation en fonction des raisonnements : par exemple, association d'informations sur leur utilisation ou modification de leurs indices. Les indices des cas concrets peuvent en effet être ajustés au cours des raisonnements par la sélection de comportements élémentaires et par l'expression de contraintes temporelles. Nous introduisons ainsi une

représentation qui permet l'expression de situations simples ou complexes, proches de scénarios temporels. Enfin, les cas concrets référencent toujours les données d'observation structurées en variables et en enregistrements. Cette séparation en deux niveaux permet la création de nombreux cas et leur modification, tout en facilitant leur interprétation.

1.2 Plate-forme à objets pour le RàPC

Dans les chapitres III et IV, nous avons détaillé notre approche qui a mené à la réalisation d'un outil ouvert permettant de faciliter l'utilisation du modèle d'indexation par situations comportementales, que nous avons proposé. Plus précisément, cet outil permet la spécialisation du modèle, son intégration cohérente dans un système complet de RàPC, et la capitalisation d'objets-métier. Aucun des outils existants en RàPC ne présente une ouverture comparable. Ces outils se limitent en effet à des boîtes noires dans lesquelles uniquement certains aspects sont modifiables, et ne facilitent pas l'intégration de nouveaux composants. C'est pourquoi nous avons proposé une nouvelle approche mettant en œuvre le concept de *plate-forme à objets*. La plate-forme à objets réalisée repose sur une architecture abstraite de collaboration entre les concepts du RàPC et de notre modèle d'indexation (index, similarité, base de cas, indice, situation comportementale). Cette architecture intègre un ensemble de *points d'ouverture* qui peuvent être configurés pour obtenir les comportements adaptés à une application particulière. La conception d'une telle architecture est réputée difficile pour garantir l'objectif de réutilisabilité systématique. Nous proposons alors une structuration des points d'ouverture identifiés en trois axes de variabilité et en deux niveaux de spécificité (RàPC et indexation par situations comportementales). Cette structuration est alors détaillée sous la forme de *cas d'utilisation*. Nous proposons également des modèles à objets ouverts utilisant des *patrons de conception*. Les patrons de conception présentent des solutions typiques à des problèmes récurrents de conception de logiciels à objets, et permettent de justifier les nombreuses décisions de conception.

Sur un plan théorique, nous apportons ainsi une modélisation à objets flexible du RàPC et de l'indexation par situations comportementales, pour faciliter la réalisation de différents systèmes. Sur un plan plus pratique, nous avons réalisé la plate-forme à objets CBR*Tools dans le langage à objets Java. CBR*Tools a été utilisée dans la réalisation d'une application classique en RàPC (cf. annexe C), dans un prototype de recherche des cas pour la nutrition des plantes, et dans Broadway. Cette plate-forme offre un ensemble d'objets-métier et permet l'intégration de nouveaux composants que l'on peut assembler : indice, index, similarité, phase de raisonnement… L'utilisation de cet outil est alors facilitée par la structuration des points d'ouverture en cas d'utilisation et par l'utilisation du vocabulaire formé par les patrons de conception utilisés. Sur le plan du génie logiciel, nous donnons l'exemple de la conception d'une plate-forme modélisant les fondements du raisonnement à partir de cas. La recherche d'architectures ouvertes et réutilisables est une démarche d'actualité, étant donné le besoin de rendre interopérables les logiciels pour assembler des composants spécialisés provenant d'origines diverses. Toutefois, la conception de plates-formes à objets est une tâche difficile pour laquelle il n'existe pas encore de méthode dédiée. Cependant, nous montrons concrètement comment la notation UML ainsi que les concepts de cas d'utilisation et de patrons de conception sont utilisés à la fois pour une meilleure conception mais aussi pour une aide à l'utilisation.

1.3 Broadway : un nouveau type d'assistants à la navigation sur le Web

Dans le chapitre V, nous avons premièrement proposé une nouvelle approche d'assistance à la navigation dans un hypermédia. Nous proposons en effet d'aider un utilisateur en réutilisant, à partir d'un groupe d'utilisateurs, des *comportements observés par variables*. Cette observation par

variables, combinée avec notre modèle d'indexation, permet la gestion de comportements détaillés dans un cadre flexible et générique. Le prototype Broadway est un exemple de cette approche et permet le calcul de recommandations en fonction d'observations sur quatre variables (adresse et contenu des pages visitées, évaluations explicite et implicite). Aucun des assistants de navigation existants fondés sur les comportements des utilisateurs ne prend en compte un ensemble de variables, ni ne permet l'ajout et le traitement unifié de nouvelles variables. Notre modèle d'indexation et notre plate-forme à objets sont ainsi mis à profit dans notre approche.

Deuxièmement, la réalisation concrète de Broadway ainsi que son évaluation expérimentale montrent la faisabilité et l'intérêt de notre approche. Broadway effectue un cycle de raisonnement à partir de cas en suivant les quatre phases classiques : recherche, réutilisation, révision et apprentissage. A partir des navigations observées, Broadway extrait des cas utiles grâce aux cas potentiels et permet de les sauvegarder sous la forme de cas concrets. Un cas représente une association issue d'une navigation, entre un comportement et un ensemble de pages proposées. A partir de la navigation courante d'un utilisateur, Broadway identifie tout d'abord un ensemble de cas qui présentent des comportements similaires. La réutilisation de ces cas consiste alors à faire la synthèse des pages proposées pour former la liste des recommandations présentées à l'utilisateur. Puis, l'utilisateur révise la solution suggérée au cours de sa navigation en évaluant, à son tour, les pages visitées. Enfin, Broadway apprend de cette interaction une nouvelle navigation et met à jour son indexation.

Broadway est également le résultat d'un travail important pour la réalisation des modules annexes au moteur de raisonnement à partir de cas. En effet, la construction d'un prototype réaliste a principalement nécessité : l'intégration de l'interface graphique dans un navigateur Web, la répartition des fonctions en huit processus communiquant par RMI (*Remote Method Invocation*), la spécialisation d'un proxy HTTP (serveur Jigsaw du W3C), et l'analyse de pages au format HTML pour l'extraction des mots clefs.

2 Perspectives

A partir de nos travaux, nous avons mis en évidence trois principales perspectives portant sur :

- l'extension et la généralisation de notre modèle d'indexation,
- l'intégration des caractéristiques d'une plate-forme à objets dans des méthodes de gestion de projet RàPC,
- la conception de systèmes d'aide basés sur notre approche de réutilisation de comportements utilisateurs.

2.1 Vers l'extension et la généralisation de notre modèle d'indexation

Nous avons appliqué notre modèle d'indexation dans le cadre de la régulation de la nutrition des plantes et celui de l'assistance à la navigation sur le Web. Pour poursuivre l'évaluation du modèle, d'autres applications sont envisageables dans cette classe de problèmes. Par exemple, une étude a déjà été menée pour l'aide à l'argumentation sur Internet dans un contexte de décision collective (Karacapilidis *et al.*, 1997). Il serait de plus intéressant d'étudier l'utilisation de notre modèle pour des systèmes existants comme REBECAS pour les feux de forêts ou RADIX pour la recherche d'information.

Aux deux types de cas proposés (concrets et potentiels), il serait également intéressant de combiner des *cas abstraits* (Bergman & Wilke, 1996) et de définir des techniques pour gérer le cycle

de vie des cas : extraction de cas potentiels, transformation en cas concrets et abstraits, élimination des cas. Notamment, des mécanismes de création de cas abstraits permettraient de généraliser et de remplacer les cas concrets : la taille de la base de cas serait alors plus réduite. Des méthodes d'abstraction de chroniques (Shahar, 1997) et de modification automatique des indices des cas sont alors nécessaires. Dans le cadre plus précis de la navigation sur le Web, des approches de recherche de séquences récurrentes de pages (Chen & Park, 1998 ; Schechter *et al.*, 1998) sont envisageables en s'appuyant sur les pages sélectionnées comme indices dans les cas concrets.

Enfin, avec notre modèle d'indexation, nous rejoignons les préoccupations du modèle d'indexation UIF adapté à une autre classe de problèmes. Selon Janet Kolodner (Kolodner, 1993, pages 219–220), cette démarche est prometteuse et permettra à terme de réutiliser des cas issus de domaines différents puisqu'ils utilisent une description basée sur les mêmes principes. Cette réutilisation pourra alors mener à de meilleurs raisonnements et à l'introduction d'un facteur de créativité. Nous pensons également que notre effort doit être poursuivi sur d'autres modèles d'indexation pour aboutir à une méthodologie visant à faciliter la définition de tels modèles. Le modèle d'indexation par situations comportementales et sa structuration présentent alors un point de départ novateur. Notre approche met également en valeur l'importance de l'indexation au sein du cycle de raisonnement à partir de cas, et nous pensons que les principes suivants, proposés dans notre modèle, pourraient être généralisés à d'autres classes de problèmes :

- séparation des données brutes et des cas dont la représentation est dépendante de la tâche du système et doit nécessairement intégrer des informations supplémentaires (sélection des indices, poids d'importance, qualité du cas).
- extraction de cas potentiels guidée par les raisonnements et leur sauvegarde sous la forme de cas concrets pour modifier leurs indices et leurs informations.

2.2 Vers des méthodes d'utilisation d'objets-métier pour le RàPC

Notre approche basée sur le concept de plate-forme à objets est une première étape vers la constitution d'un ensemble de modèles et d'objets-métier, voire des patrons-métier, facilitant la conception de systèmes de RàPC. De manière plus générale, nous pensons que ce type d'outils devrait permettre d'améliorer le processus de mise en place de systèmes RàPC. Tout d'abord, une plate-forme offre déjà un ensemble de classes abstraites et peut ainsi permettre de faciliter le passage des modèles au niveau connaissance jusqu'au niveau symbolique. Puis, une plate-forme amène des atouts qui pourraient être utilisés dans la gestion d'un projet de RàPC : prototypage rapide grâce à la réutilisation directe des composants de base, bonne stabilité aux modifications des exigences et configuration suivant les besoins des points d'ouverture. En définitive, un tel outil permet de concentrer les efforts sur les points importants et spécifiques à une application : identification des indices, acquisition des cas et des connaissances nécessaires à l'organisation de la mémoire, à l'adaptation et à l'apprentissage. C'est pourquoi l'étude de méthodes de gestion de projet RàPC (Bergmann *et al.*, 1997 ; Bergmann *et al.*, 1998) devrait s'enrichir d'une telle approche.

Puis, nous pensons améliorer notre plate-forme à objets en enrichissant les composants que cet outil contient. L'ajout de composants (schémas de phase de raisonnement, index) serait intéressant, comme par exemple : index par MOP (*Memory Organisation Package*, Kolodner, 1983 ; Schank, 1982) permettant d'indexer les cas dans une structure discriminante évolutive, index spécifiques pour les situations comportementales prenant en compte des techniques de mise en correspondance optimisées des historiques (Faloutsos *et al.*, 1994), schémas de recherche et d'adaptation par classification hiérarchique (Lieber, 1997). Le couplage effectif avec l'outil TEMPOS permettrait finalement de stocker les historiques dans une base de données à objets. Enfin, il s'agit de poursuivre notre effort pour la description de nos composants dans le modèle standard des JavaBeans (Beans,

1997). Ce modèle permet un couplage faible entre les objets (communication par événements) et amène ainsi une plus grande flexibilité. De plus, les composants peuvent être manipulés au sein d'environnements graphiques évolués, comme dans l'outil VisualAge[62] de IBM. Ces composants peuvent alors être combinés avec d'autres objets qui ne sont pas liés au RàPC (base de données, interface graphique, autres objets-métier).

2.3 Vers des assistants basés sur les comportements

Une première évaluation de Broadway a été menée dans un cadre précis et montre l'intérêt de notre approche. Toutefois, les expérimentations doivent continuer pour généraliser les résultats obtenus. Dans ce but, l'ensemble des paramètres identifiés (cf. annexe D) ainsi que les outils construits (Broadway*Tools) sont des apports importants. Nous avons également commencé l'étude de l'intégration de robots de recherche dans notre architecture. Ces robots peuvent remplacer totalement ou partiellement les utilisateurs dans leurs navigations sur le Web pour leur éviter des parcours fastidieux. La réutilisation des navigations se ferait alors au niveau des robots (Golinelli, 1998 ; Taheri, 1998). Notre approche de coopération indirecte peut également être enrichie de mécanismes de coopération directe dans le cadre d'une tâche de recherche d'information (Trousse *et al.,* 1998), et nécessite une gestion des notions de session et de profil utilisateur (Elhaddad, 1998).

Plus généralement, nous sommes à l'étude de deux applications importantes de notre approche visant des systèmes d'aide basés sur la réutilisation de comportements observés sur différentes variables :

- Dans le cadre d'une collaboration entre l'INRIA et le centre de recherche européen de Xerox (Meylan, France), nous étudions la réutilisation de comportements utilisateurs alliant des étapes d'exécution de requêtes dans un moteur de recherche, et des étapes de navigation sur le Web. Les objectifs sont de guider l'utilisateur dans son processus de reformulation de requête et de lui proposer des documents ou des bases documentaires à consulter en se basant sur des comportements passés.
- Dans le cadre d'un contrat de recherche avec le CNET (centre de recherche de France Télécom), nous visons l'assistance à un important groupe d'utilisateurs au sein d'un hypermédia délimité.

Dans ces deux applications, il s'agit d'identifier les variables d'observation des utilisateurs et de construire, grâce à notre modèle d'indexation et à CBR*Tools, des systèmes d'aides adaptés. Enfin, nous pensons que notre approche serait intéressante dans le cadre du commerce électronique pour guider des consommateurs vers les produits souhaités mais difficiles à exprimer *a priori* (configuration d'un micro-ordinateur par exemple).

[62] http://www.software.ibm.com/ad/vajava/.

Références bibliographiques

1 Liste des références

(Aamodt & Plaza, 1994) A. Aamodt et E. Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and Systems. *AI Communications*, vol. 7, n°1, pages 36–59, 1994.

(Allen, 1984) J. F. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, vol. 23, n°2, pages 123–154, 1984.

(Althoff *et al.,* 1995) K. D. Althoff, E. Auriol, R. Barletta et M. Manago. *A Review of Industrial Case-Based Reasoning Tools*. AI Perspectives Report, AI Intelligence, 148 pages, 1995.

(Andreoli *et al.,* 1996) J. M. Andreoli, U. Borghoff et R. Pareschi. Constraint-Based Knowledge Broker Model: Semantics, Implementation and Analysis. *Journal of Symbolic Computation*, vol. 21, n°4, pages 635–667, 1996.

(Armstrong *et al.,* 1997) R. Armstrong, T. Joachims et T. Mitchell. WebWatcher: A Tour Guide for the World Wide Web. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'97),* pages 770–775, Morgan Kaufmann, 1997.

(Auriol *et al.,* 1994) E. Auriol, M. Manago, K. D. Althoff, S. Wess et S. Dittrich. Integrating Induction and Case-Based Reasoning: Methodological Approach and First Evaluations. In *Proceedings of the 2nd European Workshop on CBR (EWCBR'94)*, pages 145–155, AcknoSoft press, 1994.

(Bachiochi *et al.,* 1997) D. Bachiochi *et al*. Usability Studies and Designing Navigational Aids for the World Wide Web. In *Proceedings of the 6th International World Wide Web Conference (WWW6)*, volume 29 de *Computer Network and ISDN Systems*, pages 1489–1496, Elsevier, 1997.

(Baille *et al.,* 1990) A. Baille, T. Boulard et C. Gary. Les critères d'optimisation dans la gestion du climat et de la production sous serre. *Séances de l'Académie d'Agriculture*, vol. 76, n°5, 12 pages, 1990.

(Baille, 1991) A. Baille. L'aide à la decision dans la gestion du climat sous serre : de nouvelles perspectives. *P. H. M. Revue Horticole*, n°315, pages 51–57, 1991.

(Baird-Smith *et al.,* 1997) A. Baird-Smith, Y. Lafon et B. Mahé. Jigsaw: An Object Oriented Server. W3C, http://www.w3.org/Jigsaw/User/Introduction/wp.html, 1997.

(Barletta, 1994) R. Barletta. A Hybrid Indexing and Retrieval Strategy for Advisor CBR Systems Built with REMIND. In *Proceedings of the 2nd European Workshop on CBR (EWCBR'94)*, pages 49–58, AcknoSoft press, 1994.

(Barrett *et al.*, 1997) R. Barrett, P. P. Maglio et D. C. Kellem. How to Personalize the Web. In *Proceedings of the Conference on Human-computer Interaction (CHI'97)*, pages 75–82, Addison-Wesley, 1997.

(Bassaville, 1988) M. Basseville. *Distance Measures for Signal Processing and Pattern Recognition*. Rapport de recherche n°899, INRIA, 47 pages, 1988.

(Beans, 1997) *Java Beans*. Sun Microsystems, http://java.sun.beans, 114 pages, 1997.

(Bergman & Wilke, 1996) R. Bergmann, W. Wilke. On the Role of Abstraction in Case-Based Reasoning. In *Advances in Case-Based Reasoning, Proceedings of the 3rd European Workshop on Case-Based Reasoning (EWCBR'96)*, volume 1168 de *Lecture Notes in Artificial Intelligence*, pages 28–43, Springer, 1996.

(Bergmann *et al.,* 1997) R. Bergmann, W. Wilke et J. Schumacher. Using Software Process Modeling for Building a Case-Based Reasoning Methodology: Basic Approach and Case Study. In *Case-Based Reasoning Research and Development, Proceedings of the 2nd International Conference on Case-Based Reasoning (ICCBR'97)*, volume 1266 de *Lecture Notes in Artificial Intelligence*, pages 509–518, Springer, 1997.

(Bergmann *et al.*, 1998). R. Bergmann *et al*. Collecting Experience on the Systematic Development of CBR Applications Using the INRECA Methodology. In *Proceedings of 4th European Workshop on Case-Based Reasoning (EWCBR'98)*, volume 1488 de *Lecture Notes in Artificial Intelligence*, pages 460–470, Springer, 1998.

(Berners-Lee *et al.,* 1994) T.B. Lee, R. Cailliau, A. Luotonen, H. F. Nielsen et A. Secret. The World Wide Web. *Communications of the ACM*, vol. 37, n°8, pages 76–82, 1994.

(Bird & Wadler, 1988) R. Bird et P. Wadler. *Introduction to Functional Programming*. Prentice Hall, 293 pages 1988.

(Box & Jenkins, 1976) G. E. P. Box et G. M. Jenkins. *Time Series Analysis: Forcasting and Control*. Holden Day, 575 pages, 1976.

(Brandau *et al.,* 1991) R. Brandau, A. Lemmon, et C. Lafond. Experience with Extended Episodes: Cases with Complex Temporal Structure. In *Proceedings of the 3rd DARPA Workshop on Case-Based Reasoning*, pages 1–12, Morgan Kaufmann, 1991.

(Bray, 1996) T. Bray. Measuring the Web. In *Proceedings of the 5th International World Wide Web Conference,* volume 28 de *Computer Network and ISDN Systems*, pages 993–1005, Elsevier, 1996.

(Breiman *et al.,* 1984) L. Breiman, J. H. Freidman, R. A. Olshen et C. J. Stone. *Classification and Regression Tree*. Wadsworth and Brooks, 358 pages, 1984.

(Brooks *et al.,* 1995) C. Brooks, M. S. Mazer, S. Meeks et J. Miller. Application-Specific Proxy Servers as HTTP Stream Transducers. In *Proceedings of the 4th International World Wide Web Conference*, numéro 1 de *World wide web journal*, pages 539–548, 1995.

(Brown *et al.,* 1995) M. Brown, I. Watson et N. Filer. Separating the Cases from the Data: Towards More Flexible Case-Based Reasoning. In *Case-Based Reasoning Research and Development, Proceedings of the 1st International Conference on Case-Based Reasoning (ICCBR'95)*, volume 1010 de *Lecture Notes in Artificial Intelligence*, pages 157–168, Springer, 1995.

(Brun & Settenbrino, 1993) R. Brun et A. Settenbrino. Roses hors-sol sous serre : gestion de la nutrition minérale et hydrique. *Revue horticole*, nº345, pages 25–29, 1993.

(Brun *et al.,* 1993) R. Brun, B. Paris et I. Hammelin. Fertigation Management of Rose Plants Grown in Greenhouse on Rockwool. *Advanced Horticulture Science*, vol. 7, pages 109–111, 1993.

(Bull *et al.,* 1997) M. Bull, G. Kundt et L. Gierl. Case-Based Risk Detection and Forecasting in a Geographic-Medical System. In *Proceedings of the 5th German Workshop on Case-Based Reasoning (GWCBR'97)*, rapport LSA-97-01E, Centre for Learning Systems and Applications (LSA), Université de Kaiserslautern, pages 59–64, 1997.

(Burke *et al*., 1997) D. Burke, K. J. Hammond et B. C. Young. The FindMe Approach to Assisted Browsing. *IEEE Expert*, vol. 12, n°4, pages 32–39, 1997

(Bushmann *et al.,* 1996) F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad et M. Stal. *Pattern-Oriented Software Architecture*: *A System of Patterns*. Wiley & Sons, 457 pages, 1996.

(Canavaggio, 1997) J. F. Canavaggio. *TEMPOS : un modèle d'historiques pour un SGBD temporel*. Thèse de Doctorat, Université Joseph Fourier, Grenoble, 176 pages, 1997.

(Carbonell, 1986) J. G. Carbonell. Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. *Machine Learning*, vol. 2, chapitre 14, pages 317–392, 1986.

(Charles, 1975) G. Charles. *Reconnaissance automatique de forme d'un signal transitoire rapide*. Thèse de Doctorat, Université de Paris sud, 130 pages, 1975.

(Chartron, 1996) G. Chartron. Recherche d'information sur Internet. In *la recherche d'information sur les réseaux*, cours INRIA, pages 43–101, ADBS editions, 1996.

(Chen & Park, 1998) M-S. Chen et J. S. Park. Efficient Data Mining for Path Traversal Patterns. *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, n°2, pages 209–221, 1988.

(Chen & Sycara, 1998) L. Chen et K. Sycara. WebMate: A Personal Agent for Browsing and Searching. In *Proceedings of of the 2nd International Conference on Autonomous Agents*, pages 132–139, ACM press, 1998.

(Chiriatti & Plant, 1993) K. C. Chiriatti et R. E. Plant. Case-Based Reasoning: Application to the Agricultural Domain, a Prototype. In *Proceedings of the 1st European Workshop on CBR (EWCBR'93)*, rapport SEKI SR-93-12, pages 174–177, Université de Kaiserslautern, 1993.

(Codenie *et al.,* 1997) W. Codenie, K. De Hondt, P. Steyaert et A. Vercammen. From Custom Applications to Domain-Specific Frameworks. *Communications of the ACM*, vol. 40, n°10, pages 71–77, 1997.

(Corvaisier *et al.,* 1997) F. Corvaisier, A. Mille et J.M. Pinon. Information Retrieval on the World Wide Web Using a Decison Making System. In *Actes de la 5ème conférence sur la Recherche d'Information Assistée par Ordinateur sur Internet (RIAO'97)*, Centre des hautes études internationales d'Informatique, Montréal, pages 285–295, 1997.

(Corvaisier *et al.,* 1998) F. Corvaisier, A. Mille et JM. Pinon. Recherche Assistée de documents Indexés sur l'eXpérience (RADIX) : mesures de similarités des épisodes de recherche sur le Web. In *Actes de Ingénierie des Connaissances (IC'98)*, pages 153–163, INRIA, 1998.

(Cutler *et al.,* 1997) M. Cutler, Y. Shih et W. Meng. Using the Structure of HTML Documents to Improve Retrieval. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (ITS'97)*, pages 241–252, USENIX Association, 1997.

(Demeyer *et al.,* 1997) S. Demeyer, T.D. Meijler, O. Nierstrasz et P. Steyaert. Design Guidelines for 'Tailorable' Framework. *Communications of the ACM*, vol. 40, n°10, pages 60–64, 1997.

(Deter, 1994) R. Deter. CBR for Maintenance of Telecommunication Network. In *Proceedings of the 2nd European Workshop on Case-Based Reasoning (EWCBR'94)*, pages 23–31, AcknoSoft press, 1994.

(Dillon *et al.,* 1990) A. Dillon, C. McKnight et J. Richardson. Navigation in Hypertext: A critical Review of the Concept. In *Proceedings of the 13rd International Conference on Human-Computer Interaction (INTERACT'90)*, pages 587–592, Elsevier, 1990.

(Dousson & Ghallab, 1994) C. Dousson et M. Ghallab. Suivi et reconnaissance de chroniques. *Revue d'Intelligence Artificielle*, vol. 8, n°1, pages 29–61, 1994.

(Dreyer *et al.,* 1994) W. Dreyer, A. K. Dittrich et D. Schimdt. Research Perspectives for Time Series Management Systems. *SIGMOD Record*, vol. 23, n°1, pages 10–15, 1994.

(Elhaddad, 1998) M. Elhaddad. *Conception et réalisation d'un serveur de profils utilisateurs, utilisation dans Broadway*. Mémoire de projet de fin d'études, Ecole Nationale Supérieure d'Informatique et d'Analyse des Systemes, Maroc, et INRIA Sophia Antipolis, Action AID, 63 pages, 1998.

(Elkassar & Charlet, 1997) S. Elkassar et J. Charlet. Représentation de connaissances et aide à la navigation hypertextuelle à partir de cas : application au dossier médical. In *Actes de la journée Ingénierie des Connaissances et Apprentissage Automatique (JICAA'97)*, pages 387–401, INRIA, 1997.

(Faloutsos *et al.,* 1994) C. Faloutsos, M. Ranganathan et Y. Manolopoulos. Fast Subsequence Matching in Time-Series Database. *SIGMOD Record*, vol. 23, n°1, pages 419–429, 1994.

(Faurre & Robin, 1986) P. Faurre et M. Robin. *Elément d'automatique*. Dunod, 443 pages, 1986.

(Fauvet *et al.,* 1997) M. C. Fauvet, J. F. Canavaggio et P. C. Scholl. TEMPOS : un modèle d'historiques pour un SGBD temporel à objets. In *Actes des 13eme journées Bases de Données Avancées (BDA'97)*, IMAG, pages 67–87, 1997.

(Fayad & Schmidt, 1997) M. E. Fayad et D. C. Schmidt. Object-Oriented Application Frameworks. *Communications of the ACM*, vol. 40, No 10, pages 32–38, 1997.

(Fontaine, 1996) D. Fontaine. Une approche par graphes pour la reconnaissance de scénarios temporels. *Revue d'Intelligence Artificielle*, vol. 10, n°4, pages 439–468, 1996.

(Foss, 1989) C. L. Foss. *Detecting Lost Users: Empirical Studies on Browsing Hypertext*. Rapport de Recherche n°972, INRIA, 39 pages, 1989.

(Froedlish *et al.,* 1997) G. Froedlish, H. J. Hoover, L. Liu et P. Sorenson. Hooking into Object-Oriented Application Frameworks. In *Proceedings of International Conference on Software Engineering (ICSE'97)*, pages 491–501, IEEE Computer Society press, 1997.

(Fuchs *et al.,* 1995) B. Fuchs, A. Mille et B. Chiron. Operator Decision Aiding by Adaptation of Supervision Strategies. In *Case-Based Reasoning Research and Development, Proceedings of the 1st International Conference on Case-Based Reasoning (ICCBR'95),* volume 1010 de *Lecture Notes in Artificial Intelligence*, pages 23–32, Springer, 1995.

(Fuchs, 1997) B. Fuchs. *Représentation des connaissances pour le raisonnement à partir de cas – Le système ROCADE*. Thèse de Doctorat, Université Jean Monnet de Saint-Etienne, 196 pages, 1997.

(Fynn *et al.,* 1989) R. P. Fynn, W. L. Roller et H. M. Keener. A Decision Model for Nutrition Management in Controlled Environment Argiculture. *Agricultural Systems*, n°31, pages 35–53, 1989.

(Gamma *et al.,* 1995) E. Gamma, R. Helm, R. Johnson et J. Vlissides. *Design Patterns : Elements of Reusable Object Oriented Software*. Addison-Wesley, 395 pages, 1995.

(Gebhardt *et al.,* 1997) F. Gebhardt, A. Voss, W. Grather et B. Schmidt-Belz. *Reasoning with Complex Cases*. Kluwer Academic Publisher, 250 pages, 1997.

(Geib *et al.,* 1997) J. M. Geib, C. Gransart et P. Merle. *CORBA, des concepts à la pratique*. InterEditions, 293 pages, 1997.

(Ghallab & Mounir-Alaoui, 1989) M. Ghallab et A. Mounir-Alaoui. Managing Efficiently Temporal Relations through Indexed Spanning Tree. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 1297–1303, Morgan Kaufmann, 1989.

(Golinelli, 1997) M. Golinelli. Etude d'une implémentation objet dans CBR*Tools de l'algorithme de recherche des cas similaires à base d'arbre k-d appliqué au raisonnement par cas. Rapport interne, INRIA, Action AID, 35 pages, 1997.

(Golinelli, 1998) M. Golinelli. *Persistance et robots d'évaluation pour Broadway*. Rapport de projet de DESS Informatique, Université de Nice-Sophia Antipolis et INRIA Sophia Antipolis, Action AID, 30 pages, 1998.

(Gonzalez, 1978) R.C. Gonzalez et M.G. Thomason. *Syntactic Pattern Recognition*. Addison-Wesley, 377 pages, 1978.

(Haimowitz & Kohane, 1993a) I. J. Haimowitz et I. S. Kohane. Automated Trend Detection with Temporal Hypotheses. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 146–151, Morgan Kaufmann, 1993.

(Haimowitz & Kohane, 1993b) I. J. Haimowitz et I. S. Kohane. An Epistemology for Clinically Significant Trends. In *Proceedings of the AAAI National Conference on Artificial Intelligence (AAAI'93)*, pages 176–181, MIT press, 1993.

(Hall & Dowling, 1980) P. A. V. Hall et G. R. Dowling. Approximate String Matching. *Computing Survey*, vol. 12, n°4, pages 381–402, 1980.

(Hallam-Baker & Connolly, 1996) P. M. Hallam-Baker et D. Connolly. *Session Identification URI*. Document de travail du W3C, WD-session-id-960221, http://www.w3.org/pub/WWW/TR/WD-session-id.html, 1996.

(Hammond, 1986) K. Hammond. CHEF: A Model of Case-Based Planning. In *Proceedings of the AAAI National Conference on Artificial Intelligence (AAAI'86)*, pages 267–271, Morgan Kaufmann, 1986.

(Harmon, 1992) P. Harmon. Case-Based Reasoning, Part III: A Review of the CBR Products that are Currently being Marketed. *Intelligent Software Strategies*, vol. III, n°1, pages 1–12, 1992.

(Haton *et al.,* 1991) J.P. Haton *et al*. Raisonnement distribué et modèle de tableau noir. *Le raisonnement en Intelligence Artificielle*, chapitre 10, pages 387–429, InterEditions, 1991.

(Haton *et al.,* 1991) J.P. Haton *et al*. Raisonnement par analogie. *Le raisonnement en Intelligence Artificielle*, chapitre 9, pages 365–385, InterEditions, 1991.

(Hébraud *et al.,* 1998) C. Hébraud, M. Rech et A. Thébault. *Test de l'utilité de l'outil d'aide à la navigation Broadway*. Rapport interne, Université de Provence, UFR de psychologie, Aix en Provence, et INRIA Sophia Antipolis, Action AID, 19 pages, 1998.

(Hennessy & Hinkle, 1992) D. Hennessy et D. Hinkle. Applying Case-Based Reasoning to Autoclave Loading. *IEEE Experts*, vol. 7, n°5, pages 21–26, 1992.

(Hinrichs, 1989) T.R. Hinrichs. Strategies for Adaptation and Recovery in a Design Problem Solver. In *Proceedings of the 2nd DARPA Workshop on Case-Based Reasoning*, pages 115–118, Morgan Kaufmann, 1989.

(Huitema, 1995) C. Huitema. *Et Dieu créa l'Internet...* Eyrolles, 201 pages, 1995.

(Iyengar, 1997) A. Iyengar. Dynamic Argument Embedding: Preserving State on the World Wide Web. *IEEE Internet Computing*, pages 50–56, 1997.

(Jaczynski & Trousse, 1994) M. Jaczynski et B. Trousse. Fuzzy Logic for the Retrieval Step of a Case-Based Reasoner. In *Proceedings of the 2nd European Workshop on Case-Based Reasoning* (*EWCBR'94)*, pages 313–321, AcknoSoft press, 1994.

(Jaczynski & Trousse, 1997a) M. Jaczynski et B. Trousse. *CBR*Tools: une bibliothèque objet pour l'indexation des cas par situation comportementale*. Rapport de recherche n°3215, INRIA, 38 pages, 1997.

(Jaczynski & Trousse, 1997b) M. Jaczynski et B. Trousse. Broadway: a World Wide Web Advisor Reusing Past Navigations from a Group of Users. In *Proceedings of 3rd UK Workshop on Case-Based Reasoning (UKCBR3)*, 17 pages, Université de Manchester, 1997.

(Jaczynski & Trousse, 1998a) M. Jaczynski et B. Trousse. An Object-Oriented Framework for the Design and the Implementation of Case-Based Reasoners. In *Proceedings of 6th German Workshop on Case-Based Reasoning (GWCBR'98)*, IMIB Series vol. 7, pages 33–42, Université de Rostock, 1998.

(Jaczynski & Trousse, 1998b) M. Jaczynski et B. Trousse. WWW Assisted Browsing by Reusing Past Navigations from a Group of Users. In *Proceedings of 4th European Workshop on Case-Based Reasoning (EWCBR'98)*, volume 1488 de *Lecture Notes in Artificial Intelligence*, pages 160–171, Springer, 1998.

(Jaczynski & Trousse, 1998c) M. Jaczynski et B. Trousse. Broadway, a Case-Based Browsing Advisor for the Web. In *Research and Advanced Technology for Digital Libraries, Proceedings of the 2nd European Conference on Digital Libraries (ECDL'98)- poster*, volume 1513 de *Lecture Notes in Computer Science*, pages 697–698, Springer, 1998.

(Jaczynski & Trousse, 1999) M. Jaczynski et B. Trousse. Patrons de conception dans la modélisation d'une plate-forme à objets pour le raisonnement à partir de cas. *L'objet*, numéro spécial : les Patterns, vol. 5, n°2, 27 pages, Hermes, 1999. A paraître.

(Jaczynski, 1993) M. Jaczynski. *Recherche des cas similaires et utilisation des ensembles flous*. Rapport de DEA, Université de Nice Sophia-Antipolis et INRIA Sophia-Antipolis, Projet SECOIA, 123 pages, 1993.

(Jaczynski, 1997) M. Jaczynski. A Framework for the Management of Past Experiences with Time-Extended Situations. In *Proceedings of the 6th International Conference on Information and Knowledge Management (CIKM'97)*, pages 32–39, ACM press, 1997.

(Jakob & Suslenschi, 1990) F. Jakob et P. Suslenschi. Situation Assessment for Process Control. *IEEE Expert*, vol. 5, n°2, pages 49–59, 1990.

(Jerke *et al.*, 1990) K-H. Jerke, P. Szabo, A Lesh et H. Rossler. Combining Hypermedia Browsing with Formal Queries. In *Proceedings of the 13rd International Conference on Human-Computer Interaction (Interact'90)*, pages 593–598, Elsevier, 1990.

(Johnson & Foote, 1988) R. E. Johnson et B. Foote. Designing Reusable Classes. *Journal of Object-Oriented Programmin*g, vol. 1, n°2, pages 22–35, 1988.

(Johnson, 1992) R. E. Johnson. Documenting Frameworks using Patterns. In *Proceedings of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'92)*, volume 27 de *ACM SIGPLAN Notices*, n°10, pages 63–76, 1992.

(Johnson, 1997) R. E. Johnson. Frameworks = (Components + Patterns). *Communications of the ACM*, vol. 40, n°10, pages 39–42, 1997.

(Karacapilidis *et al.*, 1997) N.I. Karacapilidis, B. Trousse et D. Papadias. Using Case-Based Reasoning for Argumentation with Multiple Viewpoints. In *Case-Based Reasoning Research and Development, Proceedings of the 2nd International Conference on Case-Based Reasoning (ICCBR'97)*, volume 1266 de *Lecture Notes in Artificial Intelligence*, pages 541–552, Springer, 1997.

(Keravnou, 1995) E. T. Keravnou. Modelling Medical Concepts as Time-Objects. In *Artificial Intelligence in Medicine*, volume 934 de *Lecture Notes in Artificial Intelligence*, pages 67–90, Springer-Verlag, 1995.

(Kolodner & Leake, 1996) J. L. Kolodner et D. B. Leake. A Tutorial Introduction to Case-Based Reasoning. *Case-Based Reasoning: Experiences, Lessons & Future Directions*, AAAI press, MIT press, pages 31–65, 1996.

(Kolodner, 1983) J. Kolodner. Maintaining Organization in a Dynamic Long-Term Memory. *Cognitive Science*, vol. 7, n°4, pages 243–280, 1983.

(Kolodner, 1992) J. Kolodner. An Introduction to Case-Based Reasoning. *Artificial Intelligence Review*, vol. 6, n°1, pages 3–34, 1992.

(Kolodner, 1993) J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 668 pages, 1993.

(Kolodner, 1996) J. L. Kolodner. Making the Implicit Explicit: Clarifying the Principles of Case-Based Reasoning. *Case-Based Reasoning: Experiences, Lessons & Future Directions*, AAAI press, MIT press, pages 349–370, 1996.

(Kopeikina *et al.,* 1988) L. Kopeikina, R. Brandau, et A. Lemmon. Case-Based Reasoning for Continous Control. In *Proceedings of the 1st DARPA Workshop on Case-Based Reasoning*, pages 250–259, Morgan Kaufmann, 1988.

(Korfhage, 1997) R. R Korfhage. *Information Storage and Retrieval*. Wiley & Sons, 349 pages, 1997.

(Koton, 1989) P. Koton. *Using Experience in Learning and Problem Solving*. PhD thesis, Departement of Computer Science, MIT, 1989.

(Kurata, 1988) K. Kurata. Greenhouse Control by Machine Learning. *Acta Horticulturae*, n°230, pages 195–200, 1988.

(Lajoie & Keller, 1994) R. Lajoie et R. K. Keller. Design and Resuse in Object-Oriented Framework: Patterns, Contracts, and Motifs in Concert. In *Actes du 62ème Congrès de l'Association Française pour l'avancement des sciences*, 1994.

(Landau, 1988) I. D. Landau. *Identification et commande des systèmes*. Hermes, 306 pages, 1988.

(Lassila & Swick, 1998) O. Lassila et R. R. Swick. *Resource Description Framework (RDF) Model and Syntax*. Document de travail du W3C, WD-rdf-syntax-19980216, http://www.w3.org/TR/ WD-rdf-syntax, 1998.

(Leake *et al.,* 1997) D. B. Leake, A. Kinley et D. Wilson. Case-Based Similarity Assessment: Estimating Adaptability from Experience. In *Proceedings of the 14th AAAI National Conference on Artificial Intelligence (AAAI'97)*, pages 674–679, AAAI press, 1997.

(Leake, 1996) D. B. Leake. CBR in Context: The Present and Future. *Case-Based Reasoning: Experiences, Lessons & Future Directions*, AAAI press, MIT press, pages 2–30, 1996.

(Lieber, 1997) J. Lieber. *Raisonnement à partir de cas et classification hiérarchique – Application à la planification de synthèse en chimie organique*. Thèse de Doctorat, Université Henri Poincaré, Nancy 1, 163 pages, 1997.

(Lieberman, 1995) H. Lieberman. Letizia: An Agent that Assists Web Browsing. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 924–929, Morgan Kaufmann, 1995.

(Lissandre, 1990) M. Lissandre. *Maîtriser SADT*. Armand Colin, 219 pages, 1990.

(Luotonen & Altis, 1994) A. Luotonen et K. Altis. World-Wide Web Proxies. In *Proceedings of the 1st International World Wide Web Conference*, volume 27 de *Computer Network and ISDN Systems*, pages 147–154, Elsevier, 1994.

(Manago *et al.,* 1994) M. Manago *et al*. *CASUEL : A Common Case Representation Language*. Délivrable D1, tâche 1.1, projet Esprit 6322, 36 pages, 1994.

(Mark *et al.,* 1996) W. Mark, E. Simoudis et D. Hinkle. Case-Based Reasoning: Expectations and Results. *Case-Based Reasoning: Experiences, Lessons & Future Directions*, AAAI press, MIT press, pages 269–294, 1996.

(Mattsson & Bosch, 1997) M. Mattsson et J. Bosch. Framework Composition: Problems, Causes and Solutions. In *Proceedings of the 23rd International Conference in Technology of Object-Oriented Languages and Systems (TOOLS '97 USA)*, 1997.

(Meeks *et al.,* 1996) W. S Meeks, C. Brooks et M. S. Mazer. Transducers and Associates: Circumventing Limitations on the World Wide Web. In *Proceedings of the 1st Conference on Emerging Technologies and Applications in Communications (etaCOM'96)*, 1996.

(Meusel *et al.,* 1997) M. Meusel, K. Czarnecki et W. Köpf. A Model for Structuring User Documentation of Object-Oriented Frameworks Using Patterns and Hypertext. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP'97)*, volume 1241 de *Lecture Notes in Computer Science*, pages 496–510, Springer, 1997.

(Micarelli & Sciarrone, 1996) A. Micarelli et F. Sciarrone. A Case-Based System for Adaptive Hypermedia Navigation. In *Advances in Case-Based Reasoning, Proceedings of the 3rd European Workshop on Case-Based Reasoning (EWCBR'96),* volume 1168 de *Lecture Notes in Artificial Intelligence*, pages 266–279, Springer, 1996.

(Mladenic, 1996) D. Mladenic. *Personal WebWatcher: Design and Implementation*. Rapport technique IJS-DP-7472, School of Computer Science, Carnegie Mellon University, 18 pages, 1996.

(Moorman & Ram, 1992) K. Moorman et A. Ram. A Case-Based Approach to Reactive Control for Autonomous Robot. In *Proceedings of the AAAI Fall Symposium on AI for Real-World Autonomous Mobile Robots*, 1992.

(Morrill, 1998) J. P. Morrill. Distributed Recognition of Patterns in Time Series Data. *Communications of the ACM*, vol. 41, n°5, pages 45–51, 1998.

(Mounir-Alaoui, 1990) A. Mounir-Alaoui. *Raisonnement temporel pour la planification et la reconnaissance de situations*. Thèse de Doctorat, Université Paul Sabatier de Toulouse, 138 pages, 1990.

(Muller, 1997) P.A. Muller. *Modélisation objet avec UML*. Eyrolles, 421 pages, 1997.

(Nakhaeizadeh, 1993) G. Nakhaeizadeh. Learning Prediction of Time Series: A Theoretical and Empirical Comparison of CBR with some other Approaches. In *Proceedings of the 1st European Workshop on Case-Based Reasoning (EWCBR'93)*, rapport SEKI SR-93-12, pages 149–153, Université de Kaiserslautern, 1993.

(Nakhaeizadeh, 1994) G. Nakhaeizadeh. Learning Prediction of Time Series: A Theoretical and Empirical Comparison of CBR with some other Approaches. In *Topics in Case-Based Reasoning*, volume 837 de *Lecture Notes in Artificial Intelligence*, pages 65–76, Springer-Verlag, 1994.

(Odenthal & Quibeldey-Cirkel, 1997) G. Odenthal et K. Quibeldey-Cirkel. Using Patterns for Design and Documentation. In *Proceeding of the European Conference on Object-Oriented Programming (ECOOP'97)*, volume 1241 de *Lecture Notes in Computer Science*, pages 511–529, Springer, 1997.

(Oracle, 1992) ORACLE7 Server, SQL Language Reference Manual. Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065, USA, 1992.

(Porter, 1980) M. F. Porter. An Algorithm for Suffix Stripping. *Program*, vol. 14, pages 130–137, 1980.

(Pree, 1994) W. Pree. *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, 268 pages, 1994.

(Quatrani, 1998) T. Quatrani. *Visual Modeling with Rational Rose and UML*. Addison-Wesley, 222 pages, 1998.

(Quinlan, 1979) J. R. Quinlan. Discovering Rules by Induction form Large Collections of Examples. *Expert Systems in the Micro Electronic Age*, pages 168–201, Edinburgh University press, 1979.

(Raggett *et al.,* 1998) D. Raggett, A. Le Hors et I. Jacobs. HTML 4.0 Specification. Recommandations du W3C, REC-HTML40, http://www.w3.org/TR/REC-html40/, 1998.

(Raggett, 1997) D. Raggett. HTML 3.2 Reference Specification. Recommandations du W3C, REC-HTML32, http://www.w3.org/TR/REC-html32, 1997.

(Ram & Santamaria, 1993) A. Ram et J.C. Santamaria. Continous Case-Based Reasoning. In *Proceedings of the AAAI Case-Based Reasoning Workshop*, pages 86–93, AAAI press, 1993.

(Ram & Santamaria, 1997) A. Ram et J.C. Santamaria. Continuous Case-Based Reasoning. *Artificial Intelligence*, vol. 90, n°1-2, pages 25-77, 1997.

(Redmond, 1990) M. A. Redmond. Distributed Cases for Case-based Reasoning: Facilitating Use of Multiple Cases. In *Proccedings of the AAAI National Conference on Artificial Intelligence (AAAI'90)*, pages 304–309, MIT press, 1990.

(REMIND, 1992) *REMIND: Developer's Reference Manual*. Cognitive System, 200-230, Commercial St., Boston MA 02109, 562 pages, 1992.

(RFC 1738, 1994) T. Berners-Lee, L. Masinter et M. McCahill. *Uniform Resource Locators (URL)*. Request For Comments (RFC) 1738, 1994.

(RFC 1808, 1995) R. Fielding. *Relative Uniform Resource Locator*. Request For Comments (RFC) 1808, 1995.

(RFC 2046, 1996) N. Freed et N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. Request For Comments (RFC) 2046, 1996.

(RFC 2068, 1997) R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. Request For Comments (RFC) 2068, 1997.

(RFC 2109, 1997) D. Kristol et L. Montulli. *HTTP State Management Mechanism*. Request For Comments (RFC) 2109, 1997.

(Riehle, 1997) D. Riehle. Composite Design Patterns. In *Proceedings of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'97)*, volume 32 de *SIGPLAN Notices*, n°10, pages 218–228, 1997.

(Riesbeck & Schank, 1989) C. K. Riesbeck et R. C. Schank. *Inside Case-Based Reasoning*. Erlbaum, 423 pages, 1989.

(Ritter & Larkin, 1994) F. E. Ritter et J. H. Larkin. Developing Process Models as Summaries of HCI Action Sequences. *Human-Computer Interaction*, vol. 9, pages 345–383, 1994.

(RMI, 1997) Java Remote Method Invocation Specification. Sun Microsystem.s Inc, http://java.sun.com/products/jdk/1.1/download-pdf-ps.html, 96 pages, 1997.

(Rose, 1998) Rational Rose 98 - Using Rational Rose. Rational Software Corporation, http://www.rational.com, 267 pages, 1996.

(Rougegrez, 1994a) S. Rougegrez. Prédiction de processus à partir de comportement observés : le système REBECAS. Thèse de Doctorat, Institut Blaise Pascal, 174 pages, 1994.

(Rougegrez, 1994b) S. Rougegrez. Similarity Evaluation between Observed Behaviours for the Prediction of Processes. In *Topics in Case-Based Reasoning*, volume 837 de *Lecture Notes in Artificial Intelligence*, pages 155–166, Springer-Verlag, 1994.

(Rüping, 1996) A. Rüping. Framework Patterns. In *Proceedings of the European Conference on Pattern Langages of Programming (EuroPLoP'96)*, http://www.cs.wustl.edu/~schmidt/europlop-96/EuroPLoP-96.html, 10 pages, 1996.

(Sakagami & Kamba, 1997) H. Sakagami et T. Kamba. Learning Personal Preferences on line Newspaper Articles from User Behaviors. In *Proceedings of the 6th Internatinal World Wide Web Conference*, volume 29 de *Computer Network and ISDN Systems*, pages 1447–1455, Elsevier, 1997.

(Schank, 1982) R. C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University press, 234 pages, 1982.

(Schechter *et al*., 1998) S. Schechter, M. Krishman et M. D. Smith. Using Path Profiles to Predict HTTP Requests. In *Proceedings of the 7th International World Wide Web Conference (WWW7)*, volume 30 de *Computer Network and ISDN Systems*, pages 457–467, Elsevier, 1997.

(Schicker *et al.,* 1996) M. A. Schicker, M. S Mazer et C. Brooks. Pan-Browser Support for Annotations and other Meta-Information on the World Wide Web. In *Proceedings of the 5th International World Wide Web Conference*, volume 28 de *Computer Network and ISDN Systems*, pages 1063–1074, Elsevier, 1996.

(Schmid, 1997) H. A. Schmid. Systematic Framework Design by Generalization. *Communications of the ACM*, vol. 40, n°10, pages 48–51, 1997.

(Schmidt *et al.,* 1996) R. Schmidt, B. Heindl, B. Pollwein, et L. Gierl. Abstraction of Data and Time for Multiparametric Time Course Prognoses. In *Advances in Case-Based Reasoning, Proceedings of the 3rd European Workshop on Case-Based Reasoning (EWCBR'96),* volume 1168 de *Lecture Notes in Artificial Intelligence*, pages 377–391, Springer, 1996.

(Schmidt, 1995) D. C. Schmidt. Experiences Using Design Patterns to Develop Reusable Object-Oriented Communication Software. *Communications of the ACM,* vol. 38, n°10, pages 65-74, 1995.

(Scholl *et al.,* 1993) P.C. Scholl, M.C. Fauvet, F. Lagnier, F. Maraninchi. Cours d'informatiques: langages et programmation. Masson, 418 pages, 1993.

(Segev & Chandra, 1993) A. Segev et R. Chandra. A Data Model for Time-Series Analysis. In *Advanced Database Systems,* volume 759 de *Lecture Notes in Computer Science,* chapitre 10, pages 191–212, Springer, 1993.

(Segev & Shoshani, 1993) A. Segev et A. Shoshani. A Temporal Data Model based on Time Sequences. *Temporal Databases: Theory, Design and Implementation*, Benjamin/Cummings, chapitre 11, pages 248–270, Benjamin-Cummings, 1993.

(Segev *et al.,* 1995) A. Segev, C. S. Jensen et R. T. Snodgrass. Report on the 1995 International Workshop on Temporal Databases. *SIGMOD Record*, vol 24, n°4, pages 46–52, 1995.

(Shahabi *et al.,* 1997) C. Shahabi, A. M. Zarkesh, J. Adibi et V. Shah. Knowledge Discovery from Users Web-Page Navigation. In *Proceedings of the 7th International Workshop on Research Issue in Data Engineering, High Performance Database Management for Large-Scale Applications*, pages 20–29, IEEE Computer Society press, 1997.

(Shahar, 1997) Y. Shahar. A Framework for Knowledge-Based Temporal Abstraction. *Artificial Intelligence*, vol. 90, n°1-2, pages 79–133, 1997.

(Slade, 1991) S. Slade. Case-Based Reasoning: A Research Paradigm. *AI magazine*, vol. 12, n°1, pages 42–55, 1991.

(Smyth & Keane, 1995) B. Smyth et M. T. Keane. Remembering to Forget. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 377–382, Morgan Kaufmann, 1995.

(Taheri, 1998) Y. Taheri. *Modèle et composants logiciels pour un moteur de recherche sur le Web fondé sur des robots intelligents*. Mémoire de projet de fin d'études, Ecole Mohammadia d'Ingénieurs, Maroc, et INRIA Sophia Antipolis, Action AID, 37 pages, 1998.

(Trousse & Visser, 1993) B. Trousse et W. Visser. Use of Case-Based Reasoning Techniques for Intelligent Computer-Aided-Design Systems. In *Proccedings of IEEE International Conference on Systems, Man and Cybernetics - Systems Engineering in the Service of Humans*, pages 513–518, 1993.

(Trousse *et al.,* 1998) B. Trousse, M. Jaczynski et R. Kanawati. Towards a Framework for Building Collaborative Information Searching Systems. In *Research and Advanced Technology for Digital Libraries*, *Proceedings of the 2nd European Conference on Digital Libraries (ECDL'98)- poster*, volume 1513 de *Lecture Notes in Computer Science*, pages 659–660, Springer, 1998.

(UML, 1997) *UML Notation Guide*. Rational Software, 142 pages, 1997.

(Vanoirbeek, 1996) C. Vanoirbeek. Avantages et limites des hypertextes pour la gestion de l'information. *La recherche d'information sur les réseaux*, cours INRIA, pages 103–126, ADBS éditions, 1996.

(Veloso, 1996) M. Veloso. Flexible Strategy Learning: Analogical Replay of Problem Solving Episodes. *Case-Based Reasoning: Experiences, Lessons & Future Directions*, AAAI press, MIT press, pages 137–149, 1996.

(Visser & Trousse, 1993) W. Visser et B. Trousse. Reuse of Designs: An Interdisciplinary Cognitive Approach. *Proceedings of the IJCAI'93 Workshop on Reuse of Designs*, 171 pages, 1993.

(Watson, 1997) I. Watson. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann, 289 pages, 1997.

(Wess *et al.,* 1994) S. Wess, K.D. Althoff, et G. Derwand. Using K-d Trees to Improve the Retrieval Step in Case-Based Reasoning. In *Topics in Case-Based Reasoning*, volume 837 de *Lecture Notes in Artificial Intelligence*, pages 167–181, Springer-Verlag, 1994.

(Wexelblat & Maes, 1997) A. Wexelblat et P. Maes. Footprints: Visualizing Histories for Web Browsing. In *Actes de la 5ème conférence sur la Recherche d'Information Assistée par Ordinateur sur Internet (RIAO'97)*, Centre des hautes études internationales d'Informatique, Montréal, pages 75–84, 1997.

(Williams, 1986) B. C. Williams. Doing Time: Putting Qualitative Reasoning on Firmer Ground. In *Proceedings of the AAAI National Conference on Artificial Intelligence (AAAI'86)*, pages 105–112, Morgan Kaufmann, 1986.

(Yan *et al.,* 1996) T.W. Yan, M. Jacobsen, H.Garcia-Molina et U. Dayal. From User Access Patterns to Dynamic Hypertext Linking. In *Proceedings of the 5th International World Wide Web Conference*, volume 28 de *Computer Network and ISDN Systems*, pages 1007–1014, Elsevier, 1996.

2 Index des références

Aamodt & Plaza, 1994 .....18, 32, 33, 35, 107, 213
Allen, 1984 .....75, 213
Althoff *et al*., 1995 .....37, 89, 213, 257
Amstrong *et al*., 1995 .....31, 197
Andreoli *et al*., 1996.....30, 213
Armstrong *et al*., 1997.....31, 197, 199, 213
Auriol *et al*., 1994 .....94, 213
Bachiochi *et al*., 1997.....31, 213
Baille *et al*., 1990 .....25, 213
Baille, 1991 .....25, 213
Baird-Smith *et al*., 1997 .....167, 213
Barletta, 1994 .....90, 213
Barrett *et al.*, 1997.....197, 199, 213
Bassaville, 1988.....56, 213
Beans, 1997 .....151, 209, 213
Bergman & Wilke, 1996.....32, 207, 214
Bergmann *et al*., 1997 .....89, 208, 214
Bergmann *et al.*, 1998 .....89, 208, 214
Berners-Lee *et al*., 1994 .....28, 214
Bird & Wadler, 1988 .....70, 214
Box & Jenkins, 1976 .....27, 214
Brandau *et al*., 1991 .....45, 214
Bray, 1996 .....30, 171, 214
Breiman *et al*., 1984 .....91, 214
Brooks *et al*., 1995 .....165, 214
Brown *et al*., 1995 .....109, 214
Brun & Settenbrino, 1993 .....17, 25, 26, 27, 214
Brun *et al*., 1993.....17, 25, 26, 27, 214
Bull *et al*., 1997 .....18, 49, 50, 54, 214
Burke *et al.*, 1997 .....197, 199, 214
Bushmann *et al*., 1996.....119, 215, 243
Canavaggio, 1997 .....62, 63, 143, 215
Carbonell, 1986 .....36, 215
Charles, 1975.....56, 215
Chartron, 1996.....28, 30, 215
Chen & Park, 1998 .....162, 208, 215
Chen & Sycara, 1998.....197, 199, 215
Chiriatti & Plant, 1993 .....27, 215
Codenie *et al*., 1997.....102, 215
Corvaisier *et al*., 1997 .....51, 162, 197, 199, 215
Corvaisier *et al*., 1998 .....18, 49, 51, 54, 77, 215
Cutler *et al*., 1997.....173, 215
Demeyer *et al*., 1997.....103, 106, 215
Deter, 1994 .....45, 215
Dillon *et al*., 1990.....31, 215
Dousson & Ghallab, 1994 .....59, 215
Dreyer *et al*., 1994 .....62, 215
Elhaddad, 1998.....167, 209, 216
Elkassar & Charlet, 1997.....197, 199, 216
Faloutsos *et al*., 1994.....208, 216
Faurre & Robin, 1986.....27, 216
Fauvet *et al*., 1997 .....63, 64, 216
Fayad & Schmidt, 1997 .....100, 216
Fontaine, 1996.....59, 216
Foss, 1989 .....31, 216
Froedlish *et al*., 1997.....105, 156, 216
Fuchs *et al*., 1995 .....18, 49, 50, 54, 77, 216
Fuchs, 1997 .....152, 216
Fynn *et al*., 1989 .....27, 216
Gamma *et al*., 1995 .....101, 103, 105, 117, 216, 243, 244
Gebhardt *et al*., 1997.....32, 94, 95, 96, 100, 107, 216
Geib *et al*., 1997.....166, 168, 216
Ghallab & Mounir-Alaoui, 1989.....60, 216
Golinelli, 1997 .....135, 216
Golinelli, 1998 .....169, 209, 217
Gonzalez, 1978.....57, 58, 217
Haimowitz & Kohane, 1993a.....59, 60, 217
Haimowitz & Kohane, 1993b.....59, 60, 61, 217
Hall & Dowling, 1980.....57, 217
Hallam-Baker & Connolly, 1996 .....165, 217
Hammond, 1986.....40, 217
Harmon, 1992 .....89, 217
Haton *et al*., 1991.....119, 217
Hébraud *et al*., 1998.....20, 193, 194, 217
Hennessy & Hinkle, 1992 .....37, 217
Hinrichs, 1989.....40, 217
Huitema, 1995.....29, 217
Iyengar, 1997 .....165, 217
Jaczynski & Trousse, 1994.....107, 110, 138, 217
Jaczynski & Trousse, 1997a.....19, 20, 217
Jaczynski & Trousse, 1997b.....20, 218
Jaczynski & Trousse, 1998a.....20, 218
Jaczynski & Trousse, 1998b.....20, 218
Jaczynski & Trousse, 1998c.....20, 218
Jaczynski & Trousse, 1999.....19, 218
Jaczynski, 1993 .....138, 152, 218
Jaczynski, 1997 .....19, 218
Jakob & Suslenschi, 1990 .....39, 218
Jerke *et al*., 1990 .....30, 218
Johnson & Foote, 1988 .....100, 101, 118, 218
Johnson, 1992 .....105, 156, 218
Johnson, 1997 .....100, 101, 105, 218
Karacapilidis *et al*., 1997 .....207, 218
Keravnou, 1995.....38, 218
Kolodner & Leake, 1996.....32, 218
Kolodner, 1983 .....32, 208, 219
Kolodner, 1992 .....35, 219
Kolodner, 1993 .....18, 32, 40, 41, 68, 208, 219
Kolodner, 1996 .....32, 34, 36, 219
Kopeikina *et al*., 1988.....45, 219
Korfhage, 1997.....173, 219
Koton, 1989 .....40, 219
Kurata, 1988.....27, 219
Lajoie & Keller, 1994 .....105, 219
Landau, 1988.....27, 45, 219
Lassila & Swick, 1998 .....172, 219
Leake *et al*., 1997.....33, 219
Leake, 1996.....32, 35, 41, 218, 219
Lieber, 1997 .....208, 219
Lieberman, 1995 .....31, 197, 199, 219
Lissandre, 1990 .....41, 219
Luotonen & Altis, 1994 .....165, 219
Manago *et al*., 1994 .....94, 219
Mark *et al*., 1996.....35, 36, 37, 93, 219
Mattsson & Bosch, 1997.....105, 220
Meeks *et al*., 1996.....164, 220

Meusel *et al.*, 1997 .....104, 105, 156, 220
Micarelli & Sciarrone, 1996 .....197, 199, 220
Mladenic, 1996 .....165, 197, 199, 220
Moorman & Ram, 1992 .....48, 220
Morrill, 1998 .....57, 220
Mounir-Alaoui, 1990 .....59, 220
Muller, 1997 .....117, 220, 235, 236, 239
Nakhaeizadeh, 1993 .....49, 220
Nakhaeizadeh, 1994 .....18, 49, 54, 220
Odenthal & Quibeldey-Cirkel, 1997.....104, 105, 220
Oracle, 1992 .....132, 220
Porter, 1980 .....173, 220, 281
Pree, 1994 .....220, 243
Quatrani, 1998 .....152, 221, 235
Quinlan, 1979 .....91, 221
Raggett *et al*., 1998 .....29, 221
Raggett, 1997 .....29, 172, 221
Ram & Santamaria, 1993 .....18, 47, 221
Ram & Santamaria, 1997 .....18, 45, 47, 52, 54, 221
Redmond, 1990 .....68, 221
Remind, 1992 .....90, 179, 221
RFC 1738, 1994 .....29, 171, 221
RFC 1808, 1995 .....171, 221
RFC 2046, 1996 .....171, 221
RFC 2068, 1997 .....29, 171, 221
RFC 2109, 1997 .....165, 221
Riehle, 1997 .....103, 221
Riesbeck & Schank, 1989 .....32, 221
Ritter & Larkin, 1994 .....38, 221
RMI, 1997 .....166, 221
Rose, 1998 .....20, 152, 221
Rougegrez, 1994a 18, 45, 46, 52, 54, 68, 77, 221
Rougegrez, 1994b 46, 222
Rüping, 1996 .....104, 222
Sakagami & Kamba, 1997 .....174, 222
Schank, 1982 .....32, 208, 222
Schechter *et al.*, 1998 .....162, 208, 222
Schicker *et al*., 1996 .....165, 222
Schmid, 1997 .....102, 103, 222
Schmidt *et al*., 1996 .....18, 49, 50, 54, 77, 222
Schmidt, 1995 .....104, 222
Scholl *et al.*, 1993 .....70, 222
Segev & Chandra, 1993 .....62, 222
Segev & Shoshani, 1993 .....62, 63, 222
Segev *et al*., 1995 .....62, 222
Shahabi *et al*., 1997 .....162, 166, 222
Shahar, 1997 .....208, 222
Slade, 1991 .....32, 35, 223
Smyth & Keane, 1995 .....36, 223
Taheri, 1998 .....209, 223
Trousse & Visser, 1993 .....138, 223
Trousse *et al*., 1998 .....209, 223
UML, 1997 .....117, 223, 235
Vanoirbeek, 1996 .....28, 223
Veloso, 1996 .....36, 223
Visser & Trousse, 1993 .....138, 223
Watson, 1997 .....32, 89, 98, 223
Wess *et al*., 1994 .....83, 135, 223, 257
Wexelblat & Maes, 1997 .....197, 199, 223
Williams, 1986 .....69, 223
Yan *et al*., 1996 .....197, 199, 223

Table des illustrations

Chapitre I : le RàPC pour une classe de problèmes

Figure I-1 : régulateur de concentration 26
Figure I-2 : exemple simple d'hypertexte (Vanoirbeek, 1996) 28
Figure I-3 : affichage d'une page Web avec le navigateur Netscape 29
Figure I-4 : l'espace virtuel du Web pour Lycos 30
Figure I-5 : principe d'un assistant à la navigation 31
Figure I-6 : cycle du raisonnement à partir de cas 33
Figure I-7 : chargement de l'autoclave 37
Figure I-8 : tâche de réalisation d'un système de RàPC 41

Chapitre II : modèle d'indexation par situations comportementales

Figure II-1 : historiques d'un cas suivant une direction de propagation 46
Figure II-2 : choix du cas minimisant le décalage 47
Figure II-3 : architecture de SINS 48
Figure II-4 : principes de la recherche dans SINS 48
Figure II-5 : mise en correspondance dans SINS 49
Figure II-6 : mise en correspondance de deux séquences dans PADIM 51
Figure II-7 : exemple de cas avec une situation non explicite dans SINS 53
Figure II-8 : électrocardiogramme humain normal (Gonzalez, 1978) 57
Figure II-9 : automate fini pour la reconnaissance d'un électrocardiogramme normal (Gonzalez, 1978) 58
Figure II-10 : déplacement d'un convoyeur dans une usine automatisée (Mounir-Alaoui, 1990) 59
Figure II-11 : définition d'une situation dans IxTeT 59
Figure II-12 : extrait d'un patron de tendances (Haimowitz & Kohane, 1993b) 61
Figure II-13 : types de chroniques (Segev & Shoshani, 1993) 63
Figure II-14 : différentes unités de temps dans TEMPOS (Canavaggio, 1997, page 48) 63
Figure II-15 : hiérarchie de types d'historiques dans TEMPOS (Fauvet *et al.,* 1997) 64
Figure II-16 : exemple de situation complexe 68
Figure II-17 : exemple d'historique en indices potentiels 70
Figure II-18 : exemples de chroniques échantillonnées et à événements 71
Figure II-19 : exemples de contraintes de localisation 73
Figure II-20 : relations entre intervalles 75
Figure II-21 : données d'observation pour la nutrition des plantes 81
Figure II-22 : exemple d'enregistrement pour la nutrition des plantes 81
Figure II-23 : exemple de cas concret pour la nutrition des plantes 83

Tableau II-1 : gestion des cas et découverte de connaissances dans les approches existantes 54
Tableau II-2 : synthèse des approches d'indexation 65

Chapitre III : plate-forme à objets pour la réalisation de systèmes de RàPC

Figure III-1 : exemple d'une hiérarchie de prototypes 90
Figure III-2 : arbre d'induction 91
Figure III-3 : index hybride dans REMIND 92
Figure III-4 : architecture de REPRO (Mark *et al.,* 1996) 93
Figure III-5 : extrait de la représentation d'un cas en CASUEL 94
Figure III-6 : contexte du calcul d'une similarité dans Rabit (Gebhardt *et al.,* 1997, page 54) 95
Figure III-7 : exemple de configuration distribuée du prototype FABEL 96
Figure III-8 : résolution d'un problème technique avec CasePoint WebServer 97
Figure III-9 : représentation d'un cas dans Help!CPR 98
Figure III-10 : extrait de l'interface de programmation de CPR 99
Figure III-11 : cycle de développement d'une plate-forme 102
Figure III-12 : types de points d'ouverture (d'après Schmid, 1997) 103
Figure III-13 : représentation d'un point d'ouverture 103
Figure III-14 : patron de conception *Composite* et son application pour des objets graphiques 104

Figure III-15 : décomposition des points d'ouverture en trois axes de variabilité 107
Figure III-16 : points d'ouverture pour la gestion du raisonnement 108
Figure III-17 : points d'ouverture pour la représentation des cas 108
Figure III-18 : points d'ouverture pour l'organisation de la mémoire 109
Figure III-19 : remplacement du point d'ouverture ReasonerFactory 110
Figure III-20 : points d'ouverture pour la représentation des cas avec des indices comportementaux 111
Figure III-21 : points d'ouverture pour l'organisation de la mémoire dans l'indexation par situations comportementales 112

Tableau III-1 : structuration et dénombrement des points d'ouverture 106

Chapitre IV : patrons de conception et modèles à objets ouverts pour le RàPC

Figure IV-1 : représentation du contrôleur de raisonnement 118
Figure IV-2 : représentation de l'état d'un raisonnement 119
Figure IV-3 : protocole standard de raisonnement 120
Figure IV-4 : interfaces de base pour les résultats des phases de raisonnement 121
Figure IV-5 : adaptation d'interfaces pour l'assemblage 122
Figure IV-6 : la fabrique comme composant du contrôleur du raisonnement 123
Figure IV-7 : classes d'objets du schéma de base pour la phase de recherche 124
Figure IV-8 : protocole de configuration et d'exécution de la recherche 124
Figure IV-9 : schéma d'adaptation d'un cas par analyse des indices 125
Figure IV-10 : protocole de réutilisation par analyse d'indices 126
Figure IV-11 : schéma de réutilisation de plusieurs cas par formule 127
Figure IV-12 : schéma d'apprentissage par chaîne de responsabilité 127
Figure IV-13 : protocole d'apprentissage par chaîne de responsabilité 128
Figure IV-14 : cas et situation 129
Figure IV-15 : indices structurés et types d'indices 129
Figure IV-16 : hiérarchie de poids d'importance pour les indices 130
Figure IV-17 : extrait du diagramme d'objets représentant les indices d'un cas 130
Figure IV-18 : délégation du stockage et de l'indexation 131
Figure IV-19 : hiérarchie de bases de cas 132
Figure IV-20 : exemple d'une structuration de bases de cas 132
Figure IV-21 : index et filtre d'indices 133
Figure IV-22 : hiérarchie extensible d'index 134
Figure IV-23 : exemple de composition d'index 135
Figure IV-24 : hiérarchie d'interfaces des résultats d'un index (recherche et indexation) 136
Figure IV-25 : transmission du résultat courant par les index composés 137
Figure IV-26 : indication des profils associés à un index 137
Figure IV-27 : exemple d'assemblage valide des index 138
Figure IV-28 : valeurs des similarités et relations d'ordre 139
Figure IV-29 : mesures de similarité structurées et modifiables 139
Figure IV-30 : extrait de la hiérarchie des fonctions élémentaires de similarité 141
Figure IV-31 : extrait de la hiérarchie des fonctions d'agrégation 142
Figure IV-32 : hiérarchie des chroniques 144
Figure IV-33 : représentation des enregistrements 144
Figure IV-34 : représentation des cas et structuration des indices 145
Figure IV-35 : décomposition de la partie comportementale 146
Figure IV-36 : hiérarchie des contraintes temporelles 146
Figure IV-37 : hiérarchie des contraintes de localisation 147
Figure IV-38 : représentation des patrons de cas potentiels 147
Figure IV-39 : hiérarchie d'algorithmes d'itération sur les enregistrements 147
Figure IV-40 : extension de la mémoire 148
Figure IV-41 : structuration typique des bases de cas concrets et potentiels 148
Figure IV-42 : structuration typique des index pour l'instanciation des patrons de cas potentiels 149
Figure IV-43 : sous-systèmes principaux de CBR*Tools 150
Figure IV-44 : organisation du sous-système core 150

Figure IV-45 : interface d'utilisation de CBR*Tools avec Rational Rose 152
Figure IV-46 : dépendances de l'axe « gestion du raisonnement » 155
Figure IV-47 : dépendances de l'axe « représentation des cas » 155
Figure IV-48 : dépendances de l'axe « organisation de la mémoire » 156

Tableau IV-1 : exemples de contrôles d'assemblages 121
Tableau IV-2 : fonctions élémentaires de similarité dans CBR*Tools 141
Tableau IV-3 : fonctions d'agrégation de CBR*tools 142
Tableau IV-4 : points d'ouverture à spécialiser obligatoirement pour une application au niveau « RàPC » 153
Tableau IV-5 : points d'ouverture à configurer obligatoirement pour une application au niveau « Indexation par situations comportementales » 154

Chapitre V : application pour l'assistance à la navigation sur le Web

Figure V-1 : réutilisation d'expériences pour déterminer des recommandations 161
Figure V-2 : barre d'outils de Broadway 163
Figure V-3 : contrôleur de Broadway 164
Figure V-4 : architecture globale de Broadway 165
Figure V-5 : architecture détaillée de Broadway 166
Figure V-6 : abonnement d'un module à des événements 168
Figure V-7 : cycle du raisonnement dans Broadway 169
Figure V-8 : exemple de structuration d'une page Web 172
Figure V-9 : extrait d'une navigation 175
Figure V-10 : exemple d'un cas concret 178
Figure V-11 : décomposition hiérarchique des adresses 179
Figure V-12 : utilisation des points d'ouverture pour le raisonnement 184
Figure V-13 : schéma de réutilisation de recommandations 185
Figure V-14 : protocole du schéma de réutilisation de recommandations 186
Figure V-15 : utilisation des points d'ouverture pour les données d'observation 187
Figure V-16 : utilisation des points d'ouverture pour les indices 187
Figure V-17 : utilisation des points d'ouverture pour les cas 188
Figure V-18 : utilisation des index lors d'une recherche des cas sources 188
Figure V-19 : utilisation des points d'ouverture pour l'organisation de l'indexation 189
Figure V-20 : utilisation des points d'ouverture pour l'organisation des cas et de la mémoire 189
Figure V-21 : extrait de l'enregistrement des événements 196
Figure V-22 : extrait du fichier généré par l'enregistreur pour les événements de la Figure V-21 196

Tableau V-1 : URI de pages Web 171
Tableau V-2 : utilisation des concepts pour la représentation des données d'observation 190
Tableau V-3 : utilisation des concepts pour la représentation des cas 190
Tableau V-4 : utilisation des points d'ouverture de CBR*Tools 192
Tableau V-5 : types de classes utilisées dans Broadway 193
Tableau V-6 : synthèse des résultats pour les trois premières hypothèses 195
Tableau V-7 : comparatif des assistants de navigation existants avec Broadway 199

Annexes

Figure A-1 : cas d'utilisation 235
Figure A-2 : types de relations entre cas d'utilisation 235
Figure A-3 : exemple de cas d'utilisation (Muller, 1997, page 127) 236
Figure A-4 : points d'ouverture et cas d'utilisation 236
Figure A-5 : utilisation des points d'ouverture 236
Figure A-6 : attributs et opérations d'une classe 237
Figure A-7 : classe abstraite et interface 237
Figure A-8 : association binaire unidirectionnelle 237
Figure A-9 : type d'associations 238

Figure A-10 : contraintes et relations 238
Figure A-11 : relation d'héritage 238
Figure A-12 : représentations d'un objet 238
Figure A-13 : diagramme d'objets (Muller, 1997, page 136) 239
Figure A-14 : diagramme de séquence 239

Figure C-1 : cycle de raisonnement 257
Figure C-2 : utilisation des points d'ouverture pour la représentation des cas 259
Figure C-3 : représentation des cas 259
Figure C-4 : utilisation des points d'ouverture pour l'organisation de la mémoire 260
Figure C-5 : hiérarchie de prototypes de la deuxième stratégie d'indexation 262
Figure C-6 : structure de la deuxième stratégie d'indexation 264
Figure C-7 : utilisation des points d'ouverture pour le contrôle du raisonnement 264

Tableau C-1 : attributs d'une voiture 258
Tableau C-2 : rapport d'exécution du premier raisonnement 267
Tableau C-3 : rapport d'exécution du deuxième raisonnement 268

Tableau D-1 : paramètres d'observation 281
Tableau D-2 : paramètres de représentation 282
Tableau D-3 : paramètres de recherche 283
Tableau D-4 : paramètres de réutilisation 283
Tableau D-5 : paramètres d'apprentissage et de maintenance 284

Annexe A : éléments de notation UML

Dans cette annexe, nous donnons les éléments de la notation UML (*Unified Modelling Language*) nécessaires pour la compréhension des diagrammes présentés dans ce document. Pour une description exhaustive de la notation, le lecteur pourra se référer aux ouvrages spécialisés (UML, 1997 ; Muller, 1997 ; Quatrani, 1998). La notation UML est une notation conçue pour la modélisation objet d'applications et fait suite notamment aux notations des méthodes OMT et Booch. Nous décrivons plus précisément les principes des diagrammes de cas d'utilisation, de classes, d'objets et de séquence.

1 Diagramme de cas d'utilisation

Les cas d'utilisation permettent de modéliser et de structurer les interactions entre les utilisateurs au sens large, appelés acteurs, et un système. Les cas d'utilisation représentent un moyen d'analyse des besoins utilisateurs et permettent de relier les actions faites par un utilisateur avec les réactions attendues d'un système. Plus précisément, un cas d'utilisation unitaire est une abstraction d'un ensemble de scénarios concrets effectués sur l'initiative d'un type d'utilisateurs (cf. Figure A-1).

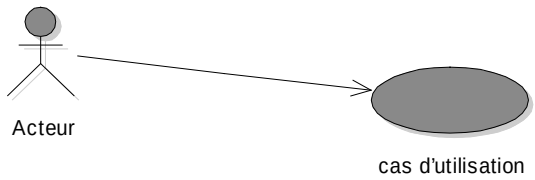


Figure A-1 : cas d'utilisation

Les cas d'utilisation peuvent être structurés et reliés par deux types principaux de relations : les relations d'utilisation (*uses*) et les relations d'extension (*extends*). Une relation d'utilisation permet de décomposer un cas d'utilisation en sous-cas d'utilisation. Une relation d'extension indique une spécialisation. Dans la Figure A-2, l'interaction définie par le cas d'utilisation A comprend celle du cas d'utilisation B, et le cas d'utilisation C est un cas particulier du cas d'utilisation B.

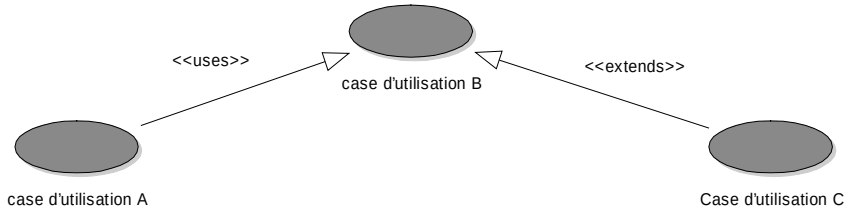


Figure A-2 : types de relations entre cas d'utilisation

Par exemple (cf. Figure A-3), si l'on désire modéliser les opérations de virements bancaires, on peut identifier deux types d'acteurs : les clients distants et les clients locaux. Les deux types de clients effectuent un virement, mais les clients distants utilisent un Minitel qui engendre une interaction spécifique (relation d'extension). Dans tous les cas, l'opération de virement comprend une phase d'identification (relation d'utilisation).

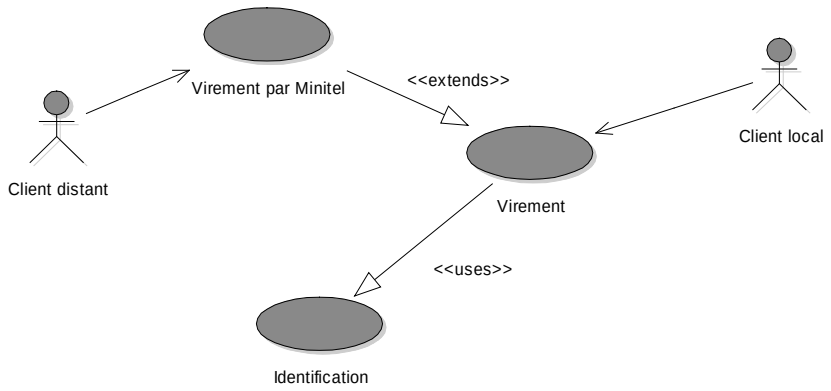


Figure A-3 : exemple de cas d'utilisation (Muller, 1997, page 127)

Dans le cadre de la plate-forme à objet CBR*Tools, nous avons utilisé les cas d'utilisation pour exprimer les interactions entre un utilisateur et le système formé par la plate-forme elle-même complétée de son environnement de manipulation. Un cas d'utilisation peut alors être relié à des points d'ouverture qui participent dans la réalisation du cas d'utilisation (cf. Figure A-4).

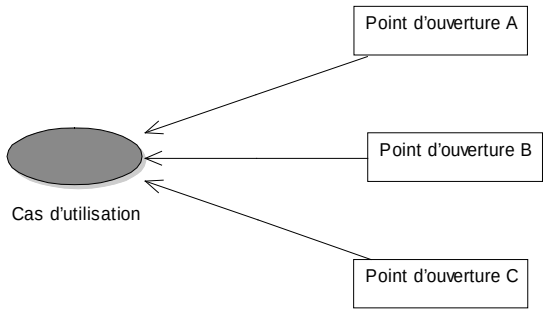


Figure A-4 : points d'ouverture et cas d'utilisation

Les cas d'utilisation nous permettent également de préciser le type de la configuration d'un point d'ouverture dans la réalisation d'une application faite avec CBR*Tools, tout en gardant la structure d'analyse initiale (cf. Figure A-5). Nous avons identifié trois types de configuration : spécialisation (point d'ouverture à boîte transparente), instanciation (point d'ouverture à boîte noire), ou utilisation du comportement par défaut. Si un même point d'ouverture est utilisé plusieurs fois avec différents types de configuration, nous ne retenons que le plus spécifique suivant cet ordre. Nous utilisons alors les stéréotypes de UML pour indiquer le type de la configuration sur les relations entre les points d'ouverture et les cas d'utilisation.

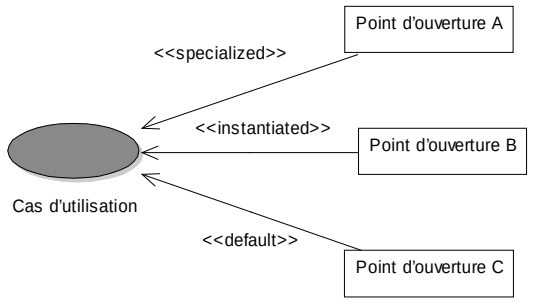


Figure A-5 : utilisation des points d'ouverture

2 Diagramme de classes

Une classe d'objets est représentée par un rectangle comprenant trois parties (cf. Figure A-6) : nom de la classe, attributs et opérations (ou méthodes). Les listes des attributs et des opérations sont toutefois optionnelles suivant le degré de détail recherché dans un diagramme : ces parties peuvent être vides ou même absentes. Les attributs et les opérations possèdent une visibilité (notamment publique ou protégée) qui est indiquée par un symbole précédent leur nom : si la forme d'une clef est dessinée, l'accès est dit protégé car il est réduit à la classe courante et à ses sous-classes.

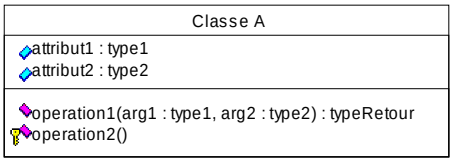


Figure A-6 : attributs et opérations d'une classe

Trois types de classes sont utilisés : les classes concrètes, les classes abstraites et les interfaces. Contrairement à une classe concrète, une classe abstraite dont la définition est en italique (cf. Figure A-7), ne peut pas être instanciée. Une interface est une classe ne définissant que des opérations et ne contient aucun code. Une interface est identifiée par l'indication du stéréotype « Interface » accolé à son nom.

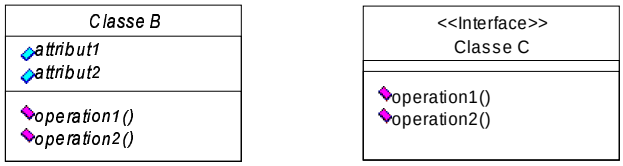


Figure A-7 : classe abstraite et interface

Une association représente une relation structurelle entre différentes classes. Une association est généralement bidirectionnelle mais nous précisons à chaque fois le sens de la navigation. L'association est alors dite unidirectionnelle (cf. Figure A-8) et indique également le sens de lecture de l'association. Chaque classe joue un rôle dans l'association qui porte une indication de multiplicité (1 pour un et un seul objet, et une étoile pour zéro à plusieurs objets). Par exemple, la Figure A-8 indique qu'un objet de classe C1 est associé à un ensemble d'objets de classe C2, sachant qu'un même objet de classe C2 est associé à un unique objet de classe C1.

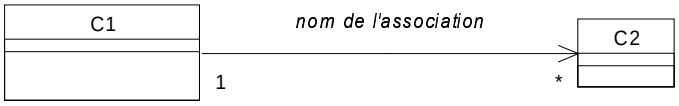


Figure A-8 : association binaire unidirectionnelle

Différents types d'associations sont également définis (cf. Figure A-9) : associations qualifiées, associations dérivées, associations avec classe et agrégations. Une association qualifiée est une association qui permet de restreindre les objets référencés dans une association grâce à une clef. Une association dérivée est une association calculée à partir d'une autre association. Une association avec classe est une association elle-même gérée par une classe qui peut ainsi fournir des attributs ou des opérations spécifiques pour l'association. Enfin, une agrégation est une association fortement asymétrique.

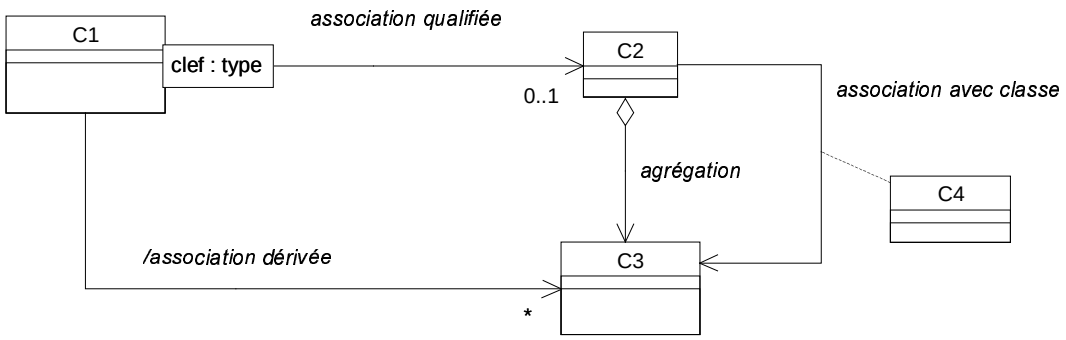


Figure A-9 : type d'associations

Des contraintes peuvent être posées sur les associations (cf. Figure A-10) : contrainte sur le rôle d'une association (par exemple, les objets sont ordonnés), ou entre associations (par exemple, une association est un sous-ensemble d'une autre).

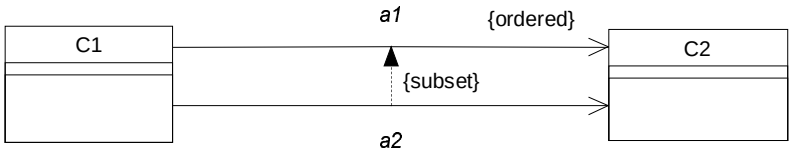


Figure A-10 : contraintes et relations

Enfin, les classes peuvent être reliées par la relation d'héritage simple ou multiple (cf. Figure A-11).

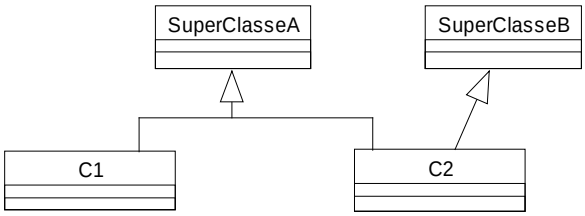


Figure A-11 : relation d'héritage

3 Diagramme d'objets

Un diagramme d'objets permet de montrer concrètement un assemblage entre objets. Un objet (ou instance) est représenté par un rectangle qui comporte généralement le nom de l'objet et sa classe (cf. Figure A-12). Toutefois, la classe ou le nom de l'objet peuvent être omis.

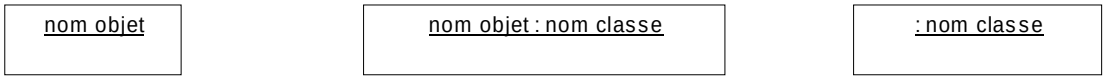


Figure A-12 : représentations d'un objet

La Figure A-13 donne l'exemple d'un diagramme d'objets montrant les relations d'agrégation entre une voiture, ses roues et son moteur.

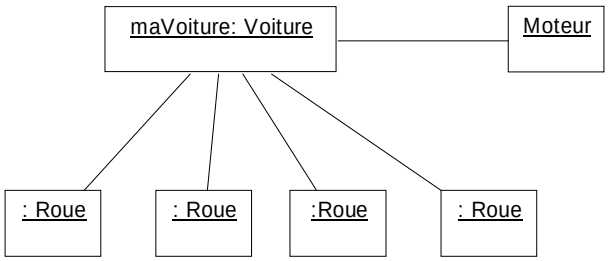


Figure A-13 : diagramme d'objets (Muller, 1997, page 136)

4 Diagramme de séquence

Un diagramme de séquence montre chronologiquement (de haut en bas) les interactions entre un ensemble d'objets (cf. Figure A-14). Chaque objet dispose d'une ligne de vie (ligne verticale). Sur ces lignes de vie, des périodes d'activité sont indiquées par des rectangles fins qui sont superposés en cas d'appel récursif.

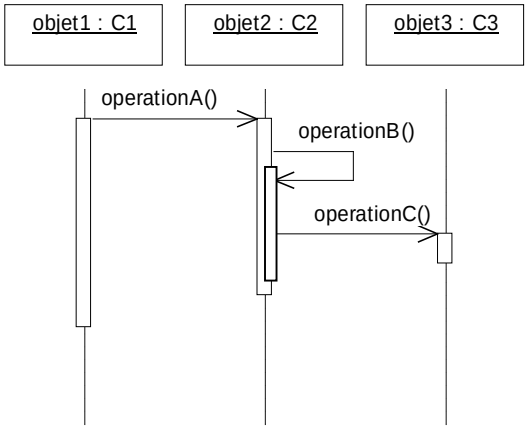


Figure A-14 : diagramme de séquence

Annexe B : patrons de conception et CBR*tools

Dans cette annexe, nous présentons un mémento des patrons de conception utilisés dans CBR*Tools. Notre objectif n'est donc pas de décrire chaque patron utilisé de manière exhaustive, mais seulement de faciliter la lecture du document en rappelant les caractéristiques fondamentales de ces patrons. Le lecteur pourra se référer à des ouvrages spécialisés pour de plus amples descriptions (Gamma *et al.*, 1995 ; Pree, 1994 ; Bushmann *et al.*, 1996). Chaque patron est présenté suivant une fiche qui comporte six parties :

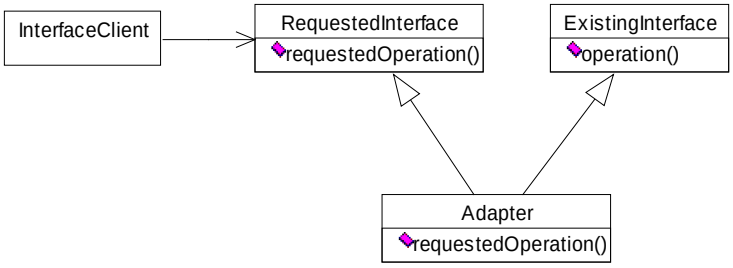
- *nom* : donne le nom du patron en français et en anglais tel qu'il est défini dans (Gamma *et al.*, 1995).
- *description* : décrit très brièvement le contexte, le problème et la solution abordés par ce patron,
- *structure et constituants* : liste les éléments typiques intervenant dans le patron avec des illustrations graphiques (diagramme de classes et si nécessaire diagramme d'objets).
- *collaborations* : définit les principales interactions entre les constituants avec, si nécessaire, la représentation graphique d'un diagramme de séquence.
- *forces et faiblesse*s : présente les principales forces (avantages) et faiblesses (inconvénients) du patron, soit dans sa mise en œuvre, soit dans son fonctionnement.
- *utilisation dans CBR*Tools* : référence les utilisations principales du patron dans CBR*Tools.

Les fiches sont classées dans l'ordre alphabétique : Adaptateur, Chaîne de responsabilité, Composite, Fabrication, Fabrique abstraite, Façade, Itérateur, Patron de méthode, Prototype, Stratégie.

Adaptateur (*Adapter*)

Description : ce patron est utile lorsque l'on désire utiliser une classe existante mais que l'interface proposée n'est pas satisfaisante ; il propose alors de définir par héritage multiple une classe effectuant les actions d'adaptation nécessaires.

Structure et constituants :



- RequestedInterface définit l'interface désirée par le client.
- ExistingInterface définit l'interface existante des objets à manipuler.
- Adpater effectue les actions nécessaires pour la réalisation des opérations suivant l'interface désirée en se basant sur les opérations existantes.
- InterfaceClient manipule l'objet suivant l'interface désirée.

Collaborations :

- le client manipule l'objet par l'interface désirée.
- l'adaptateur effectue les actions d'adaptation et utilise l'interface existante.

Forces et faiblesses :

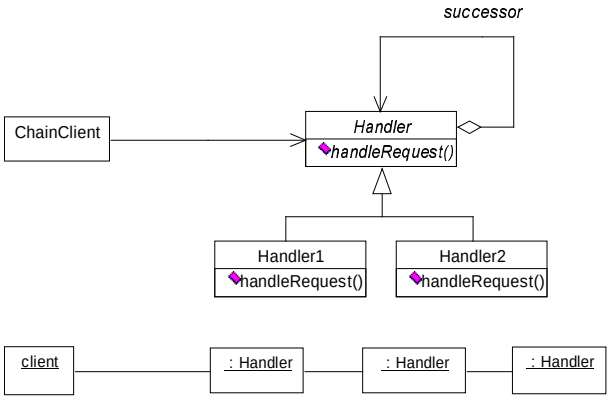
- 👍 permet la réutilisation d'une classe d'objets existante dans un autre contexte.
- 👎 il n'est pas toujours possible d'effectuer des actions d'adaptation pour réaliser ce patron.

***Utilisation dans CBR*Tool*s :** ce patron est utilisé pour connecter deux phases de raisonnement dont la première fournit un type de résultats qui n'est pas conforme à l'attente de la seconde. Ce patron est également utilisé de la même manière pour connecter des index. Toutefois, dans CBR*Tools, l'application de ce patron est facilitée par la définition de hiérarchies de classes interfaces (i.e. ne définissant aucun code). L'adaptation évite ainsi l'appel à du code existant (qui n'est pas toujours adaptable) et se présente de manière symétrique entre l'interface d'origine et celle qui est désirée. Ce type d'adaptateur est dit *bidirectionnel* (Gamma *et al.*, 1995, p. 168).

Chaîne de Responsabilité (*Chain of Responsibility*)

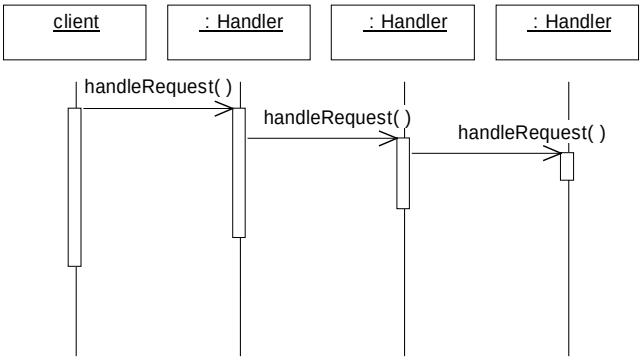
Description : lorsque plusieurs objets sont susceptibles de pouvoir traiter une requête suivant le contexte, ce patron permet d'éviter le couplage direct entre l'émetteur de la requête et son traitement par l'un des récepteurs potentiels. Une chaîne d'objets récepteurs est alors constituée et parcourue jusqu'à ce que l'un d'entre eux traite effectivement la requête.

Structure et constituants :



- Handler définit l'interface d'un récepteur de requêtes.
- Handler1 et Handler2 traitent concrètement une requête ou la passent à leur successeur.
- ChainClient transmet une requête.

Collaborations : la requête émise par le client est transmise le long de la chaîne jusqu'à son traitement effectif.



Forces et faiblesses :

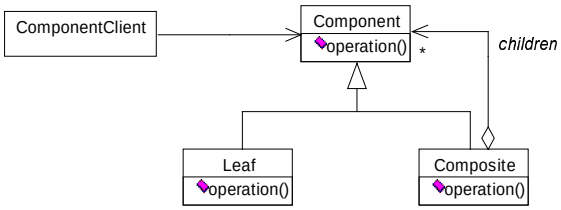
- 👍 réduction du couplage entre le client et le traitement d'une requête par un ensemble d'objets dont on ne sait pas au préalable lequel pourra effectivement traiter la requête.
- 👍 possibilité de modifier dynamiquement la façon de traiter une requête.
- 👍 souplesse dans l'attribution des responsabilités de traitement aux objets de traitement.
- 👎 suivant les contraintes de conception, le traitement peut ne pas être garanti.

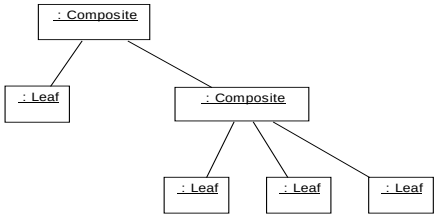
***Utilisation dans CBR*Tool*s :** un schéma d'apprentissage utilise ce patron pour prendre la décision de l'ajout d'un cas et de son indexation dans la mémoire. Une chaîne de décideurs est alors formée, dans laquelle chaque élément étudie suivant un critère la réponse à cette requête.

Composite (*Composite*)

Description : lorsque l'on souhaite représenter une arborescence ou masquer les différences entre des objets composites et simples de natures similaires, le patron composite propose la définition d'une interface unifiée d'objets simples et composites.

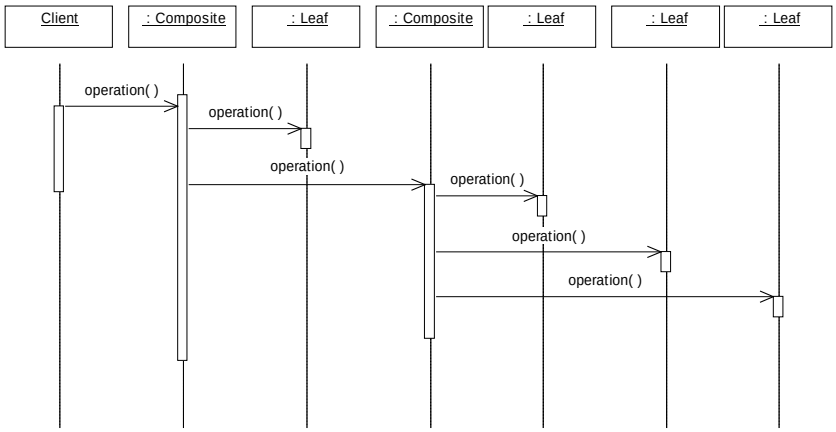
Structure et constituants :





- Component définit l'interface abstraite d'un composant simple ou composite.
- Leaf définit le comportement d'un composant feuille, c'est-à-dire qui ne peut pas avoir de fils.
- Composite définit le comportement d'un composite en stockant ses fils et en répercutant le traitement des méthodes à ses fils de manière appropriée.
- ComponentClient manipule des composants.

Collaborations : les clients manipulent un composant au travers de son interface générale. Une feuille exécute directement les opérations demandées alors qu'un composite transfère les requêtes à ses fils en opérant des actions supplémentaires avant ou après ce transfert.



Forces et faiblesses :

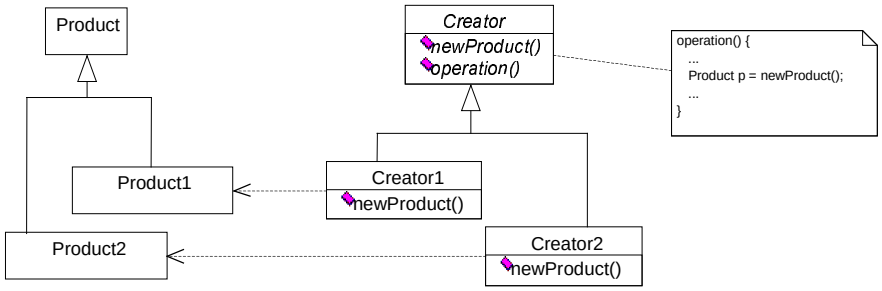
- structuration arborescente d'objets : un composite qui réunit des sous-composants peut être à son tour intégré dans un composite.
- encapsulation de l'hétérogénéité d'objets simples et composites : un objet composite peut être utilisé à la place d'un objet simple et inversement.
- intégration facilitée de nouvelles classes d'objets composites ou simples qui pourront être combinés de manière stable.
- nécessité de contrôler dynamiquement des contraintes particulières dans la composition d'objets au sein d'un composite.
- la maximisation de l'interface générale d'un composant (nécessaire pour l'encapsulation) peut amener à déclarer des méthodes qui n'ont pas de sens pour des objets simples.

***Utilisation dans CBR*Tool*s :** ce patron est utilisé pour représenter de manière structurée les indices, les similarités, les bases de cas et les index.

Fabrication (*Factory Method*)

Description : lorsqu'un algorithme a besoin de créer un objet dont sa classe concrète est variable suivant le contexte, ce patron propose la délégation à une sous-classe de l'instanciation de cet objet.

Structure et constituants :



- Creator définit une méthode dans laquelle un objet doit être créé et délègue cette instanciation à une de ses sous-classes par la définition de méthode à spécialiser (méthode newProduct).
- Creator1 et Creator2 implantent de manière concrète l'instanciation de l'objet requis.
- Product définit l'interface abstraite de l'objet qui est manipulé par Creator.
- Product1 et Product2 implantent les comportements des objets à utiliser.

Collaborations :

- Creator délègue à ses sous-classes l'instanciation de l'objet requis.

Forces et faiblesses :

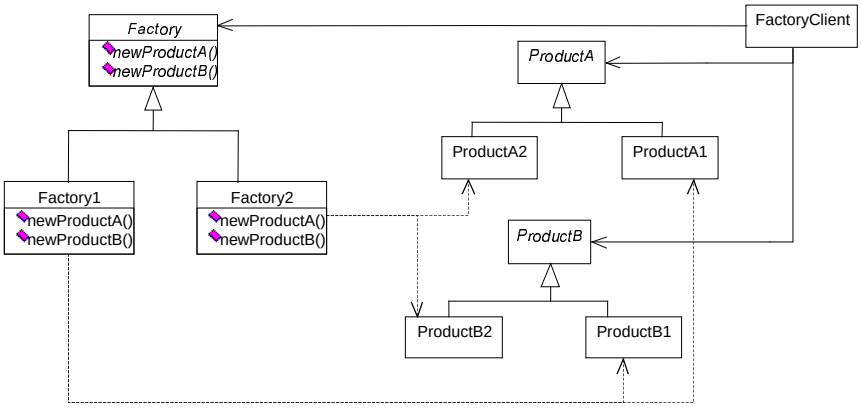
- 👍 dispense d'avoir à spécifier directement l'objet à instancier et permet à une sous-classe de choisir l'objet concret voulu.
- 👎 dans certains cas, la spécialisation de la classe mère est effectuée uniquement pour spécifier l'instanciation (opération plus lourde par exemple que de passer en paramètre l'objet requis). Il est ainsi préférable d'utiliser ce patron lorsqu'il faut également pour d'autres raisons spécialiser la classe mère.

***Utilisation dans CBR*Tool*s :** ce patron intervient dans différents algorithmes de CBR*Tools et notamment dans les protocoles internes des index pour la création des objets sauvegardant le résultat d'une recherche (de classe IndexResult) de manière à instancier les objets adéquats.

Fabrique Abstraite (*Abstract Factory*)

Description : lorsque différentes familles d'objets apparentés doivent être représentées tout en masquant leurs différences aux clients, ce patron propose la définition d'une interface abstraite centralisée de création d'un ensemble d'objets qui peut être spécialisée.

Structure et constituants :



- Factory définit l'interface abstraite de création d'une famille d'objets.
- Factory1 et Factory2 réalisent la création des objets en encapsulant le choix des classes concrètes.
- ProductA et ProductB définissent les interfaces de manipulation des objets à manipuler dans une famille.
- ProductA1, ProductA2, ProductB1 et ProductB2 définissent les comportements concrets des objets.
- FactoryClient crée les objets d'une famille et les manipule au travers de leur interface abstraite.

Collaborations :

- une fabrique délègue à ses sous-classes la création effective des objets d'une famille.
- le client n'accède aux objets ainsi créés que par leur interface abstraite.
- les objets d'une même famille peuvent coopérer directement entre eux si nécessaire.

Forces et faiblesses :

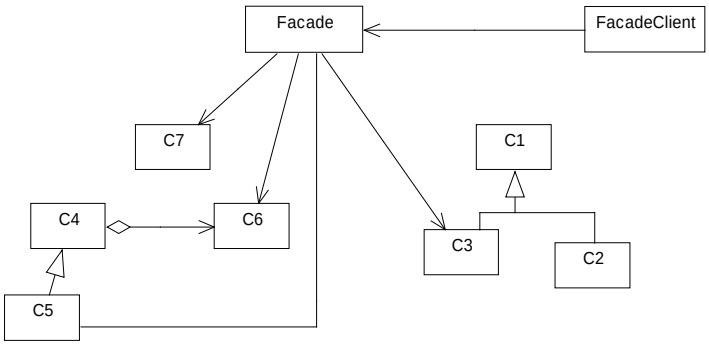
- 👍 le maintien de la cohérence entre objets est favorisé.
- 👍 la substitution d'une classe concrète par une autre est facilitée et reste masquée pour le client.
- 👎 si la coopération directe entre les objets concrets est nécessaire, ces objets sont rendus moins réutilisables et interchangeables.

Utilisation dans CBR*Tools : les objets nécessaires au raisonnement (phase de raisonnement, cas cible, objets sauvegardant l'état du raisonnement, etc.) représentent une famille dont la cohérence doit être assurée. C'est pourquoi, une fabrique abstraite (classe ReasonerFactory) est utilisée pour réunir les opérations de création de ces objets. Cette fabrique est spécialisée (classe TimeExtendedReasonerFactory) de manière à déclarer les opérations de création d'objets spécifiques pour l'indexation par situations comportementales.

Façade (*Facade*)

Description : lorsque l'on désire faciliter l'utilisation d'un sous-système, ce patron permet de fournir une interface simplifiée qui évite le couplage direct entre les clients et les éléments du sous-système.

Structure et constituants :



- Facade définit une interface unifiée pour le sous-système et donne accès à certains composants publics.
- C1 à C7 implantent des fonctions du sous-système.
- FacadeClient manipule le sous-système en passant par la façade.

Collaborations :

- le client manipule le sous-système en s'adressant à la façade ou aux éléments du sous-système rendus publics par la façade.
- la façade transmet les requêtes au sous-système après transformation si nécessaire.

Forces et faiblesses :

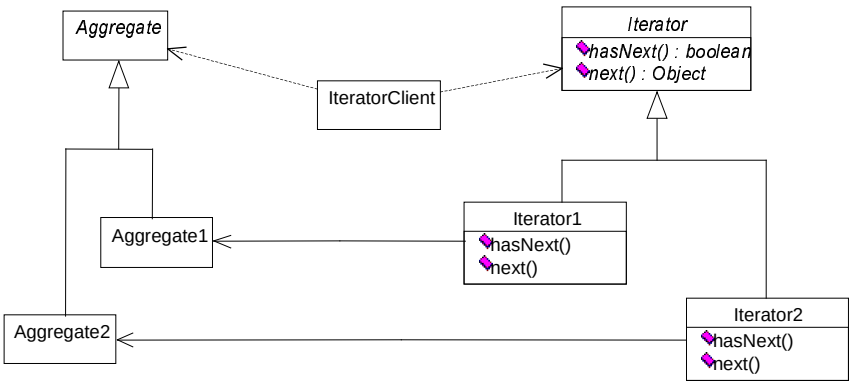
- diminue le couplage entre le client et le sous-système.
- masque des éléments privés du sous-système.
- l'interface unifiée présentée par la façade peut être trop restrictive pour utiliser l'ensemble des fonctions du sous-système.

***Utilisation dans CBR*Tool*s :** le contrôleur du raisonnement (classe Reasoner) est une façade pour l'ensemble du système de raisonnement, la mémoire (classe Memory) est une façade simplifiant la gestion de la base des cas et les index, enfin la partie comportementale (classe BehaviouralSituation) d'une situation comportementale est une façade des indices comportementaux pour simplifier leur gestion.

Itérateur (*Iterator*)

Description : lorsque l'on désire masquer au client la mécanique de parcours d'un agrégat, ce patron permet l'encapsulation de l'itération dans une hiérarchie de classe, soit pour des itérations polymorphes, soit pour offrir différents types de parcours pour un même type d'agrégat.

Structure et constituants :



- Iterator définit l'interface d'itération.
- Iterator1 et Iterator2 implantent deux mécaniques de parcours.
- Aggregate définit l'interface d'un agrégat.
- Aggregate1 et Aggregate2 implantent deux types d'agrégats concrets.
- IteratorClient lance l'itération sur l'agrégat.

Collaborations :

- le client demande la création d'un objet d'itération à l'agrégat ou à un autre objet responsable de l'instanciation de l'itérateur.
- le client contrôle l'itération (itérateur externe) en manipulant l'itérateur.

Forces et faiblesses :

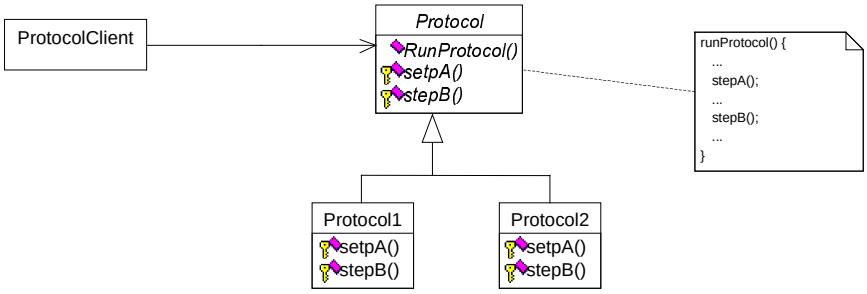
- permet de parcourir un agrégat sans connaître sa structure interne.
- permet de parcourir de différentes manières un même agrégat.
- l'agrégat peut être obligé d'exporter des méthodes uniquement pour que les itérateurs puissent fonctionner.

Utilisation dans CBR*Tools : le patron Itérateur est utilisé pour parcourir les enregistrements (classe Record) dans l'indexation par situations comportementales. Chaque type d'enregistrements définit un parcours par défaut adapté à sa représentation interne (propriétés des chroniques dont il est composé). De plus, les patrons de cas potentiels (classe PotentialCaseTemplate) définissent leur propre parcours d'un enregistrement suivant leurs conditions d'instanciation.

Patron de méthode (*Template Method*)

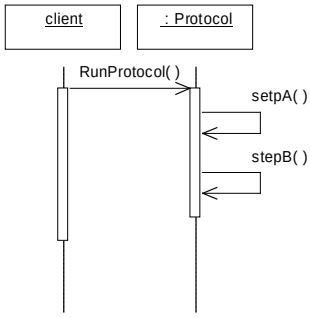
Description : lorsqu'un algorithme possède des étapes variables suivant le contexte, ce patron propose de déléguer à des sous-classes la réalisation concrète de ces étapes.

Structure et constituants :



- Protocol définit l'interface de l'algorithme et appelle des méthodes pour la réalisation de certaines étapes variables.
- Protocol1 et Protocol2 implantent les étapes manquantes dans l'algorithme.
- ProtocolClient lance l'exécution de l'algorithme.

Collaborations : le client lance l'algorithme dont la partie invariante est exécutée directement alors que les étapes variables sont réalisées de manière spécifique par les opérations définies dans une classe concrète.



Forces et faiblesses :

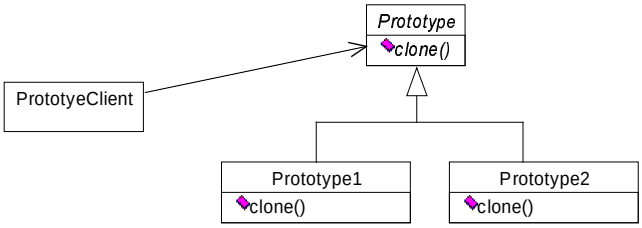
- permet d'implanter une fois pour toutes la structure d'un algorithme et des parties invariables qui seront partagées.
- permet de personnaliser certaines parties d'un algorithme dans des sous-classes.
- les étapes déléguées aux sous-classes doivent être choisies avec soin pour être vraiment utiles.
- problème de vérification des hypothèses faites par la classe mère qui peuvent être violées par l'implantation dans une sous-classe.

Utilisation dans CBR*Tool*s : ce patron est utilisé dans les phases de raisonnement et dans le protocole d'indexation, pour offrir des algorithmes et des protocoles ouverts.

Prototype (*Prototype*)

Description : lorsque l'on désire créer un objet à partir d'un objet existant, ce patron propose la définition d'une méthode de clonage qui masque au client les opérations effectives de duplication.

Structure et constituants :



- Prototype définit l'interface de l'opération de clonage avec si besoin une implantation par défaut.
- Prototype1 et Prototype2 implantent l'opération de clonage appropriée.
- PrototypeClient crée de nouveaux objets en demandant à un objet primaire de se cloner.

Collaborations :

- un client demande à un objet de se cloner.
- l'opération de clonage est déléguée aux sous-classes.

Forces et faiblesses :

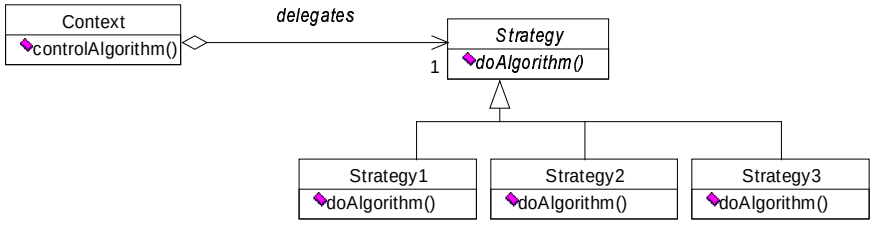
- encapsule la création d'un objet et le client n'a pas de connaissance explicite sur la classe de cet objet.
- permet la définition dynamique de classes d'objets plutôt qu'une définition statique faite à la compilation.
- permet de constituer des objets qui ne diffèrent que par des modifications faites incrémentalement.
- l'implantation d'une opération de clonage peut être difficile s'il existe des composants qui ne peuvent être clonés (opération de clonage non préalablement définie) ou s'il y a des références circulaires.
- la profondeur du clonage des composants à effectuer peut dépendre du contexte d'utilisation.

Utilisation dans CBR*Tools : ce patron est utilisé pour la constitution par clonage d'une bibliothèque de mesures de similarité, pour la copie de vecteurs de poids et pour la configuration du parcours des index. Pour réaliser ces clonages, tous les objets accessibles depuis une similarité implantent eux-mêmes une opération de clonage. Cette opération retourne un nouvel objet identique à l'objet primaire ou le même objet, si celui-ci est non modifiable (il sera donc partagé par tous les clones).

Stratégie (*Strategy*)

Description : lorsqu'il existe plusieurs implantations possibles d'un algorithme et qu'il peut être nécessaire d'en choisir une dynamiquement, ce patron propose la délégation de l'exécution de c*et al*gorithme à des objets interchangeables qui respectent une interface commune.

Structure et constituants :



- Strategy définit l'interface d'appel d'un algorithme.
- Strategy1, Strategy2 et Strategy3 implantent des variantes d'un même algorithme.
- Context détient le contexte d'exécution.

Collaborations :

- Context lance l'exécution de l'algorithme.
- Strategy peut avoir besoin de données contextuelles présentes dans Context.

Forces et faiblesses :

- 👍 permet la définition de famille d'algorithmes.
- 👍 permet le choix de l'algorithme de manière dynamique et adaptée au contexte.
- 👎 il peut être difficile de prévoir la représentation des paramètres issus du contexte d'exécution qui sont transmis de manière identique à tous les algorithmes concrets.

***Utilisation dans CBR*Tool*s :** ce patron est utilisé pour chacune des phases du raisonnement (classes Retrieve, Resuse, Revise et Retain) ainsi que la réalisation des phases elles-mêmes (classes CaseSetAdpatationFormula et AnalysisAdpatationFormula). Il est également utilisé pour les filtres d'indices (classe IndiceFilter) et dans les similarités (classes ValueSimilarityFct et AggregationFct).

Annexe C : exemple d'utilisation de CBR*Tools

Pour illustrer l'aide apportée par CBR*Tools dans la réalisation d'un système de RàPC, nous donnons l'exemple de notre prototype dans le cadre du calcul du facteur de risque d'une voiture pour un assureur. L'implantation de ce système nécessite uniquement la création de dix classes dont le code reste très déclaratif. Nous présentons tout d'abord le cadre de l'application, puis nous expliquons la construction du système de raisonnement avec CBR*Tools. Enfin, nous donnons des exemples d'exécution ainsi que l'intégralité du code source.

1 Présentation de l'application

L'objectif de cette application est de déterminer le facteur de risque (nombre entier compris entre -3 et 3) pour une voiture donnée (cas cible) en fonction des assignations déjà effectuées sur d'autres voitures (cas sources). L'approche RàPC repose sur l'utilisation d'une base de cas dans laquelle un cas associe la description des caractéristiques d'une voiture (problème formé d'indices) au facteur de risque (solution). Nous proposons alors d'effectuer un raisonnement en trois étapes dans lequel la phase de révision n'est pas utilisée (cf. Figure C-1). La phase de recherche permet d'identifier un ensemble de cas dont la description est proche de celle de la voiture courante pour laquelle il s'agit de déterminer le facteur de risque. Deux stratégies de recherche sont mises en œuvre pour les évaluer : recherche par plus proches voisins, et recherche combinée d'un filtrage par hiérarchie de prototypes et sélection par plus proches voisins. La phase de réutilisation prend en compte l'ensemble des cas retrouvés pour calculer le facteur de risque, en effectuant une moyenne des facteurs de risque assignés aux cas sources pondérés par leur similarité. Enfin, la phase d'apprentissage ajoute le nouveau cas à la mémoire, s'il n'existe pas déjà un cas trop proche du cas cible courant, et si la base de cas ne dépasse pas une taille donnée.

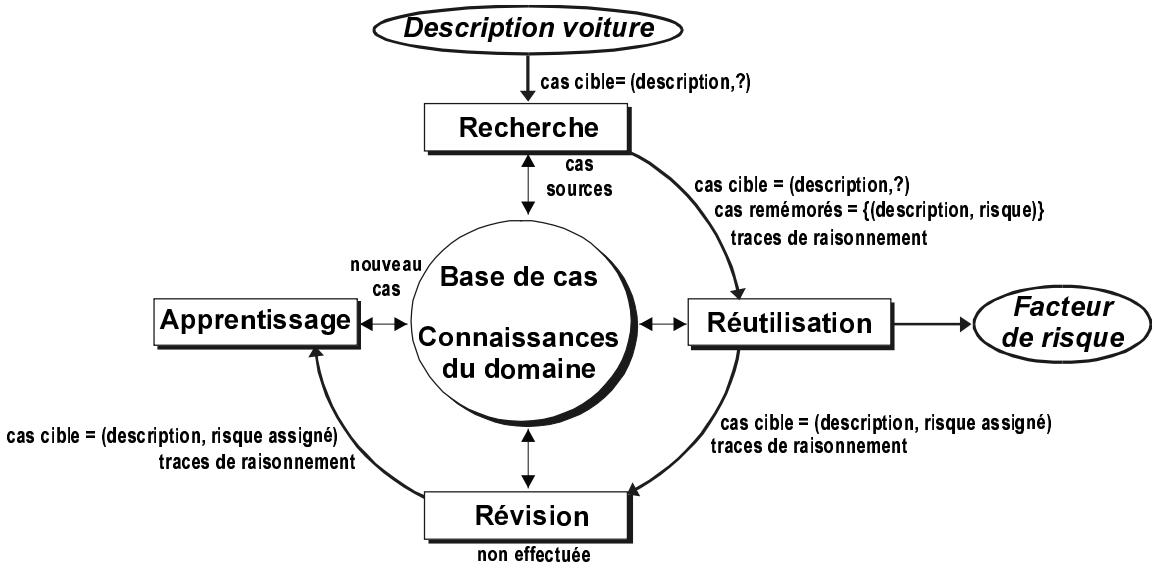


Figure C-1 : cycle de raisonnement

Bien que le raisonnement proposé soit réaliste, il ne s'agit pas de valider les résultats obtenus. Le prototype réalisé a pour but uniquement de montrer comment CBR*Tools facilite le développement d'un tel cycle de raisonnement. La base de cas utilisée dans ce prototype a déjà été utilisée pour effectuer des comparaisons et des expérimentations en RàPC (Althoff *et al.*, 1995 ; Wess *et al.*, 1994). Elle est librement accessible sur le serveur FTP :

ftp://ftp.ics.uci.edu/pub/ml-repos/machine-learning-databases/autos

2 Construction du système de raisonnement

Nous expliquons la construction de l'application et l'utilisation des ressources de CBR*Tools suivant les trois axes de variabilité : représentation des cas (2 classes), organisation de la mémoire (3 classes), gestion du raisonnement (4 classes).

2.1 Représentation des cas

La représentation des cas repose sur des données brutes décrivant 26 attributs pour chaque voiture selon le Tableau C-1.

Attribut	*Description*	*Domaine de valeur*
risk	Facteur de risque	entier de -3 à 3
normalizedLosses	Perte moyenne due aux sinistres	entier de 65 à 256
make	Marque de la voiture	symbole (exemple alfa-romero, audi, bmw, chevrolet…)
fuelType	Type de carburant	entier {1=diesel, 2=gas}.
aspiration	Turbo	entier {1=std, 2=turbo}
numOfDoors	Nombre de portes	entier {4=four, 2=two}
bodyStyle	Style	symbole {hardtop, wagon, sedan, hatchback, convertible}
driveWheels	Motorisation	entier {1=4wd, 2=fwd, 3=rwd}
engineLocation	Propulsion	entier {1=front, 2=rear}
wheelBase	Empattement	réel de 86.6 à 120.9
length	Longueur	réel de 141.1 à 208.1
width	Largeur	réel de 60.3 à 72.3
height	Hauteur	réel de 47.8 à 59.8
curbWeight	Poids carrosserie	réel de 1488 à 4066
engineType	Type de moteur	symbole {dohc, dohcv, l, ohc, ohcf, ohcv, rotor}
numOfCylinders	Nombre de cylindres	entier {2,3,4,5,6,8,12}
engineSize	Taille du moteur	réel de 61 to 326
fuelSystem	Type d'alimentation	symbole {bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi}
bore	Largeur des pistons	réel de 2.54 à 3.94
stroke	Temps du moteur	réel de 2.07 à 4.17
compressionRatio	Compression	réel de 7 à 23
horsepower	Puissance	réel de 48 à 288
peakRpm	Vitesse maximum	réel de 4150 à 6600
cityMpg	Consommation en ville	réel de 13 à 49
highwayMpg	Consommation sur autoroute	réel de 16 à 54
price	Prix	réel de 5118 à 45400

Tableau C-1 : attributs d'une voiture

Ces attributs de base sont alors séparés en deux groupes : la solution (attribut risk) et les indices potentiels (tous les autres attributs). Suivant les points d'ouverture définis dans CBR*Tools (cf. Figure C-2), nous définissons deux classes pour la représentation des cas : la classe CarCase (spécialisation du point d'ouverture CbrCase) et la classe CarCaseSituation (spécialisation du point d'ouverture CompoundIndice).

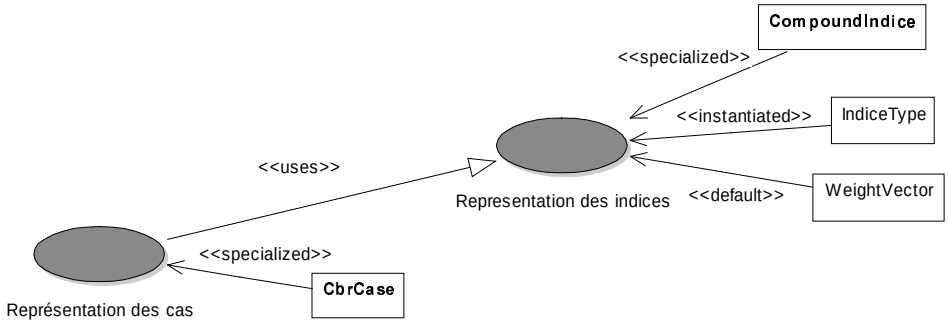


Figure C-2 : utilisation des points d'ouverture pour la représentation des cas

La classe CarCase hérite indirectement de CbrCase par la classe abstraire AbstractCbrCase. Cette classe utilitaire permet d'alléger le codage d'un cas en gérant les opérations de base (affectation d'un identificateur au cas, lien avec la base de cas). De même, la classe CarCaseSituation hérite de CompoundIndice au travers de la classe abstraite JavaClassIndice. Cette dernière implante l'interface d'un indice composite en utilisant une classe Java comme conteneur d'indices grâce aux mécanismes de réflexivité que possède le langage Java.

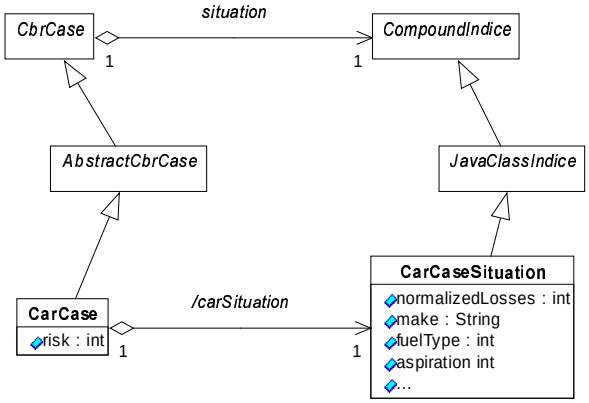


Figure C-3 : représentation des cas

Le code de la classe CarCase comprend la déclaration de deux attributs : l'un sauvegardant le facteur de risque et l'autre donnant les indices.

```
public class CarCase extends AbstractCbrCase {
   public int risk = 0;
   public CarCaseSituation indices = new CarCaseSituation();
...
}
```

Grâce à l'utilisation de la classe JavaClassIndice de CBR*Tools, la représentation des indices est grandement facilitée et s'intègre dans le langage Java. Les sous-indices d'un indice composite sont alors identifiés par des attributs d'une classe Java. Ces attributs doivent être publics pour qu'ils puissent être manipulés directement et efficacement. Toutefois, l'utilisation d'un accesseur (méthode

publique d'accès avec un attribut privé ou virtuel) est tout à fait possible et même nécessaire pour représenter des indices calculés (non utilisés dans cet exemple).

```
public class CarCaseSituation extends JavaClassIndice {
   public int normalizedLosses;
   public String make;
   public float height;
   public int numOfCylinders;
...
}
```

De plus, l'initialisation de la classe permet de déclarer effectivement les indices et de préciser les informations de typage : type (entier, réel, symbole), intervalle pour les attributs entier et réel, domaine de valeurs pour les types liste d'entiers.

```
static private  int _cylinders []  = {2,3,4,5,6,8,12};
static {
      ArrayIndiceDescriptor desc =
        new ArrayIndiceDescriptor("aid.cbr.samples.cars.CarCaseSituation");
     desc.addIndiceDescriptor(new IndiceDescriptor("normalizedLosses", new RangeIntType(65,256)));
     desc.addIndiceDescriptor(new IndiceDescriptor("make", StringType.STRING));
     desc.addIndiceDescriptor(new IndiceDescriptor("height", new RangeFloatType(60,75)));
     desc.addIndiceDescriptor(new IndiceDescriptor("numOfCylinders",new ListIntType(_cylinders)) );
     ...
 }
```

2.2 Organisation de la mémoire

L'organisation de la mémoire consiste à stocker les cas dans une base de cas et à les indexer. Dans ce but, trois points d'ouverture sont spécialisés (cf. Figure C-4) : base de cas (avec la classe CarSimpleFileCaseBase), définition d'une mesure de similarité (classe CarSimilarity) et constitution de la base d'index (classe CarIndexBase).

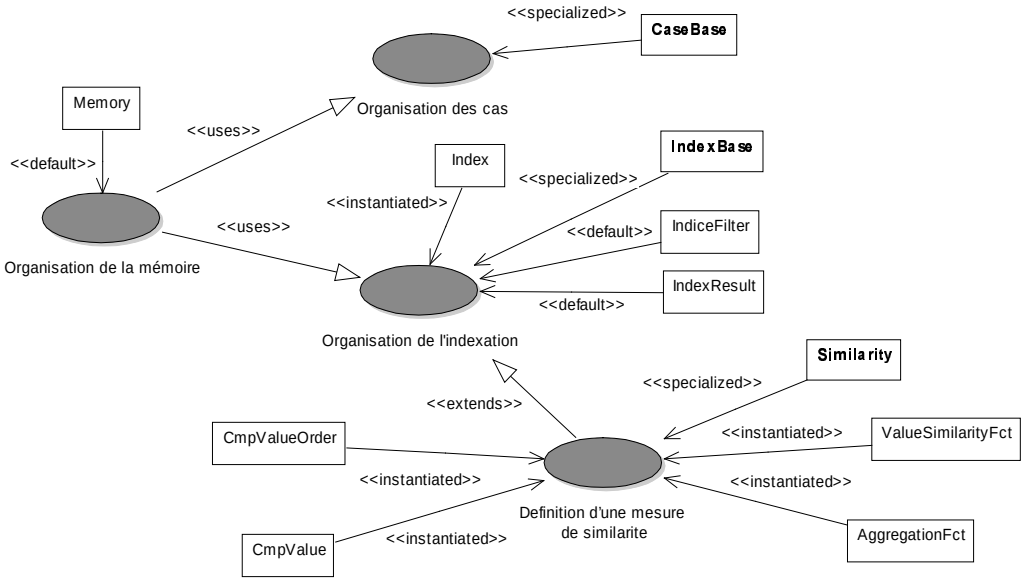


Figure C-4 : utilisation des points d'ouverture pour l'organisation de la mémoire

2.2.1 Base de cas

Dans cet exemple simple, nous ne cherchons pas à structurer la base de cas en sous-bases. Nous utilisons directement la classe abstraite FileSimpleCaseBase de CBR*Tools pour stocker les 205 cas disponibles dans un fichier. Le fichier de la base de cas est alors de la forme suivante :

```
CSV, Jeffrey C. Schlimmer, 19.05.1987, 1985 Auto Imports Database
2,1,168,toyota,2,1,2,hatchback,3,1,94.5,168.7,64.0,52.6,2204.0,ohc,4,98.0,2bbl,3.19,3.03,9.0,70.0,4800.0,29.0,34.0,8238.0
3,1,168,toyota,2,1,2,sedan,3,1,94.5,168.7,64.0,52.6,2169.0,ohc,4,98.0,2bbl,3.19,3.03,9.0,70.0,4800.0,29.0,34.0,8058.0
37,1,?,porsche,2,1,2,hatchback,3,1,98.4,175.7,72.3,50.5,3366.0,dohcv,8,203.0,mpfi,3.94,3.11,10.0,288.0,5750.0,17.0,28.0,?
```

La première ligne du fichier décrit les métadonnées de la base de cas (format, nom du créateur de la base, date de création, description). Ensuite, chaque ligne représente un cas suivant des données séparées par des virgules. La première donnée représente l'identificateur unique du cas, et les suivantes donnent les valeurs des attributs dans l'ordre du Tableau C-1. Les attributs dont les valeurs sont inconnues dans un cas sont marqués par le signe "?". La gestion de cette base de cas se fait très simplement en définissant deux méthodes déclarées abstraites dans la classe FileSimpleCaseBase : transformation d'une ligne en un cas (méthode stringToCase) pour la lecture, et transformation d'un cas en une ligne (méthode caseToString) pour l'écriture. Les autres opérations (ouverture, écriture, accès aux cas) sont prises en charge automatiquement par les super-classes.

2.2.2 Mesure de similarité

La mesure de similarité qui sera utilisée dans l'indexation est définie par la classe CarSimilarity qui hérite indirectement de CompoundIndiceSimilarity par la classe abstraite CompoundArraySimilarity. Cette dernière implante les opérations de base d'une similarité composite en utilisant un tableau de sous-similarités. Le code de la classe CarSimilarity est complètement déclaratif et consiste en l'instanciation de différents objets associant aux indices les fonctions élémentaires de similarité.

Tout d'abord, le constructeur de cette classe permet l'initialisation de la similarité avec une fonction d'agrégation par moyenne pondérée.

```
public class CarSimilarity extends CompoundIndiceArraySimilarity {
   public CarSimilarity () {
      super(new MeanAggregationFct(),25);
```

Ensuite, la structure de la mesure de similarité ainsi que les fonctions élémentaires de similarité sont instanciées et associées aux indices.

```
Class c = JavaClassIndice.getClass("aid.cbr.samples.cars.CarCaseSituation");
SimpleIndiceSimilarity floatAtt = new SimpleIndiceSimilarity(new FloatDomainSimilarityFct());
setSubSimilarity(CarCaseSituation.indexOf(c,"height"),floatAtt);
...
SimpleIndiceSimilarity intAtt = new SimpleIndiceSimilarity(new IntDomainSimilarityFct());
setSubSimilarity(CarCaseSituation.indexOf(c,"normalizedLosses"),intAtt);
setSubSimilarity(CarCaseSituation.indexOf(c,"numOfCylinders"),intAtt);
...
SimpleIndiceSimilarity eqsim = new SimpleIndiceSimilarity(new IdentitySimilarityFct(    new CmpFactorValue(1),
                                                                                         new CmpFactorValue(0)));
setSubSimilarity(CarCaseSituation.indexOf(c,"make"),eqsim);
```

Par exemple, une fonction élémentaire de similarité par matrice symétrique est associée à l'indice bodyStyle. Cette fonction est définie pour retourner la valeur 1 lorsque les valeurs des indices à comparer sont identiques, 0 si l'une d'entre elles n'est pas spécifiée. Puis, chaque couple de valeurs est explicitement défini avec la valeur de similarité correspondante.

```
SymetricExtensionSimilarityFct bodyStyleSim = new SymetricExtensionSimilarityFct(    new CmpFactorValue(0),
                                                                                     new CmpFactorValue(1));
bodyStyleSim.define("hardtop","convertible", new CmpFactorValue((float)0.9));
bodyStyleSim.define("hardtop","hatchback", new CmpFactorValue((float)0.5));
```

```
        bodyStyleSim.define("hardtop","sedan", new CmpFactorValue((float)0.5));
        setSubSimilarity(CarCaseSituation.indexOf(c,"bodyStyle"),new SimpleIndiceSimilarity(bodyStyleSim));
      }
}
```

2.2.3 Base d'index

Pour l'indexation, nous désirons mettre en place deux stratégies : la première utilise un index par plus proches voisins et la deuxième combine un index par hiérarchie de prototypes (filtrage) avec un index par plus proches voisins (sélection). La réalisation de ces deux stratégies est effectuée dans la classe CarIndexBase héritant indirectement de IndexBase par la classe abstraite BasicIndexBase qui implante la gestion de base des différentes stratégies. Le code de la classe CarIndexBase est alors réduit à la déclaration et à la configuration des index nécessaires.

```
public class CarIndexBase extends BasicIndexBase {
  CarIndexBase() {
      // First strategy: knn
      buildAndDeclareKnn();

      // Second strategy: prototype + knn
      declarePrototype();
      declareKnnForPrototype();
      buildProtoKnn();
  }
  ....
}
```

Plus précisément, la construction de la première stratégie est effectuée par l'instanciation de l'index LinearKnnIndex avec en paramètres la fonction d'ordre, la mesure de similarité, et le nombre de cas à retourner lors d'une recherche. D'autres paramètres sont également spécifiés : nom de l'index, commentaires, et valeur à utiliser en cas d'indices inconnus. Enfin, l'index est ajouté à la base.

```
protected void buildAndDeclareKnn() {
      LinearKnnIndex knnIndex = new LinearKnnIndex(new CmpFactorValueOrder(),new CarSimilarity(),5);
      knnIndex.setName("CarCaseKnn");
      knnIndex.setComments("This index uses Knn method to retrieve car cases");
      knnIndex.setUnknownCmpValue(new CmpFactorValue((float)0.5));
      addTopLevelIndex(knnIndex);
  }
```

La deuxième stratégie, plus complexe, s'exprime encore par de simples déclarations. Tout d'abord, l'index par hiérarchie de prototypes est construit par instanciation de l'index PrototypeIndex en indiquant que chaque prototype de la hiérarchie (feuilles et nœuds) doit sauvegarder les cas qui lui sont compatibles lors de l'indexation (classification).

```
protected void declarePrototype() {
      PrototypeIndex protoIndex = new PrototypeIndex();
      protoIndex.setName("ProtoIndex");
      protoIndex.setAddCaseToAcceptedPrototypes(true);
```

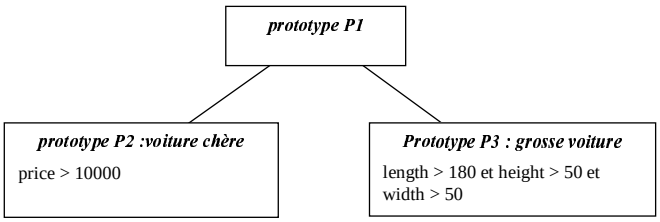


Figure C-5 : hiérarchie de prototypes de la deuxième stratégie d'indexation

Puis les prototypes (cf. Figure C-5) sont déclarés en utilisant les classes anonymes (déclaration de la classe au niveau de l'instanciation) de Java qui sont dans ce cas très pratiques.

```
Prototype p1 = new BooleanPrototype () { // root prototype
   public CmpValue computeMatching(CompoundIndice indices) {
            return CmpBoolValue.TRUE;
   }
};
p1.setName("Root");
protoIndex.addPrototype(p1);

Prototype p2 = new BooleanPrototype () { // expensive cars prototype
   public CmpValue computeMatching(CompoundIndice indices) {
            int priceIndex = ((JavaClassIndice)indices).indexOf("price");
            if (!indices.isUnknown(priceIndex)) {
               float price = indices.getFloatIndice(priceIndex);
               return (price > 10000)?CmpBoolValue.TRUE:CmpBoolValue.FALSE;
            }
            return CmpBoolValue.TRUE;
   }
};
p2.setName("ExpensiveCars");
protoIndex.addPrototype(p2);

Prototype p3 = new BooleanPrototype () { // big cars prototype
   public CmpValue computeMatching(CompoundIndice indices) {
            int lengthIndex = ((JavaClassIndice)indices).indexOf("length");
            int widthIndex = ((JavaClassIndice)indices).indexOf("width");
            int heightIndex = ((JavaClassIndice)indices).indexOf("height");
            if ( !indices.isUnknown(lengthIndex) &&
               !indices.isUnknown(widthIndex) &&
               !indices.isUnknown(heightIndex)) {
                     float length = indices.getFloatIndice(lengthIndex);
                     float width = indices.getFloatIndice(widthIndex);
                     float height = indices.getFloatIndice(heightIndex);
                     return (length > 180 && height > 50 && width > 50)?CmpBoolValue.TRUE:CmpBoolValue.FALSE;
               }
            return CmpBoolValue.TRUE;
   }
};
p3.setName("BigCars");
protoIndex.addPrototype(p3);

// build the tree
p1.addSon(p2);
p1.addSon(p3);
protoIndex.setRoot(p1);

addIndex(protoIndex);
}
```

De manière identique à la première stratégie, un autre index par plus proches voisins est déclaré.

```
protected void declareKnnForPrototype() {
   LinearKnnIndex knnIndex = new LinearKnnIndex(new CmpFactorValueOrder(),new CarSimilarity(),5);
   knnIndex.setName("CarCaseKnnAfterProto");
   knnIndex.setComments("This index uses Knn method to retrieve car cases");
   knnIndex.setUnknownCmpValue(new CmpFactorValue((float)0.5));
   addIndex(knnIndex);
}
```

Enfin, l'index par hiérarchie de prototypes et l'index par plus proches voisins sont combinés grâce à un index composite instance de ToDynamicKnnIndex (cf. Figure C-6)

```
protected void buildProtoKnn() {
    ToDynamicKnnIndex index = new ToDynamicKnnIndex();
    index.setName("Proto+KNN");
    index.addIndex(getIndex("ProtoIndex"));
    index.addIndex(getIndex("CarCaseKnnAfterProto"));
    addTopLevelIndex(index);
}
```

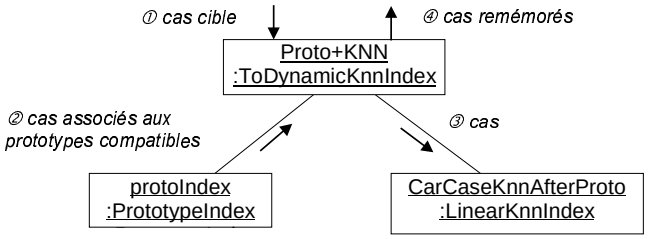


Figure C-6 : structure de la deuxième stratégie d'indexation

2.3 Gestion du raisonnement

La gestion du raisonnement nécessite la spécialisation de quatre points d'ouverture (cf. Figure C-7) : phase du raisonnement (recherche, réutilisation et apprentissage) et la fabrique chargée d'instancier les objets spécifiques de l'application.

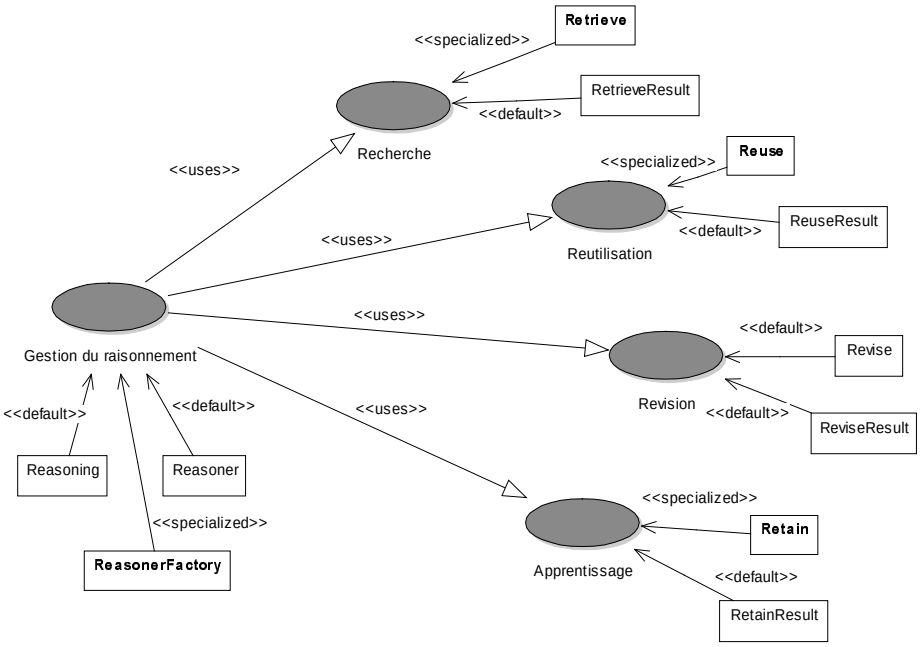


Figure C-7 : utilisation des points d'ouverture pour le contrôle du raisonnement

2.3.1 Recherche

Le protocole de recherche est spécialisé avec la classe CarRetrieve qui redéfinit la méthode updatedRetrieveResult pour transformer les résultats du parcours des index en résultats de la phase de recherche : récupération des identificateurs de cas et transformation de la similarité des cas retrouvés en degré d'utilité pour la phase de réutilisation.

```
public class CarRetrieve extends BasicRetrievePattern {

  protected  void updateRetrieveResult(IndexResult result, RetrieveResult stepResult){
      List res= Assoc.getFactory().newListAssoc();
      res.addAll(((CaseSetIndexResult)result).retrievedCases());
      stepResult.setCases(res);

      for (Iterator it = ((CaseSetIndexResult)result).retrievedCases().iterator();
          it.hasNext(); ) {
         Object id = it.next();
         float f = ((CmpFactorValue)((CaseSimIndexResult)result).getCaseMatchingResults(id)).getFactor();
         ((UsefulnessRetrieveResult)stepResult).setCaseUsefulness(id,f);
      }
  }
}
```

2.3.2 Réutilisation

La phase de réutilisation se base sur le schéma d'adaptation par formule prenant en compte l'ensemble des cas retrouvés. Le protocole de ce schéma (classe abstraite CaseSetFormulaReuse) est spécialisé dans la classe CarReuse. Tout d'abord, les méthodes de lecture et d'écriture de la solution d'un cas sont spécifiées, puis le constructeur initialise l'objet avec une formule d'adaptation par moyenne pondérée tout en prenant soin d'arrondir le résultat qui doit être un entier compris entre -3 et 3 (à noter, encore une fois, l'utilisation d'une classe anonyme de Java).

```
public  class CarReuse extends CaseSetFormulaReusePattern  {

protected  float getSourceCaseSolution(CbrCase c){
      return ((CarCase)c).risk;
  }

protected  void setTargetCaseSolution(CbrCase c, float sol){
      ((CarCase)c).risk=(int)sol;
  }

CarReuse () {
      setFormula( new MeanAdaptationFormula() {
         public float compute(float[] solutions, float[] usefullness) {
                float f = super.compute(solutions,usefullness);
                return Math.round(f);
         }
      });
  }
}
```

2.3.3 Apprentissage

La phase d'apprentissage est réalisée suivant le schéma par chaîne de responsabilité. La classe CarRetain est alors réduite à une simple déclaration de deux analyseurs : le premier examine si la base de cas comporte plus de 250 cas, et le deuxième vérifie s'il existe déjà un cas dont la similarité est très proche (au-dessus de 0,95) du cas cible courant. Ainsi, de manière très flexible et simple, le cas cible courant sera ajouté et indexé dans la mémoire si et seulement si ces deux conditions ne sont pas satisfaites.

```
public  class CarRetain extends ResponsibilityChainRetainPattern {
CarRetain() {
        RetainAnalyser a1 = new CaseBaseSizeRetainAnalyser(250);
        RetainAnalyser a2 = new SimilarityRetainAnalyser(new CmpFactorValue((float)0.95));
        a1.setNextAnalyser(a2);
        setAnalyser(a1);
   }
}
```

2.3.4 Fabrique des objets

Enfin, la fabrique d'objets est spécialisée pour référencer les classes d'objets spécifiquement créées pour l'application.

```
public class CarReasonerFactory extends AbstractReasonerFactory {
   public CbrCase newTagetCase() {
        return new CarCase();
   }
   public CaseBase newCaseBase() {
        return new CarFileSimpleCaseBase("CarCaseBase","cars.txt");
   }
   public Retrieve newRetrieveStep() {
        return new CarRetrieve();
   }
   public Reuse newReuseStep() {
        return new CarReuse();
   }
   public Retain newRetainStep() {
        return new CarRetain();
   }
   public IndexBase newIndexBase() {
        return new CarIndexBase();
   }
}
```

3 Exemples d'exécution

Nous présentons deux exemples d'exécution de raisonnements réalisés dans la classe CarsApp. Ces raisonnements montrent l'utilisation de deux configurations différentes.

3.1 Premier raisonnement

Dans le premier raisonnement, nous lançons l'exécution avec les paramètres par défaut. Tout d'abord, nous instancions un nouveau contrôleur de raisonnement (variable reasoner) depuis la fabrique.

```
ReasonerFactory factory = new CarReasonerFactory();
Reasoner reasoner = factory.getReasoner();
```

Nous pouvons alors ouvrir la base de cas, et l'indexation automatique de l'ensemble des cas issus de la base est effectuée suivant les deux stratégies d'indexation présentes.

```
reasoner.getMemory().getCaseBase().open();
reasoner.getMemory().buildFromCaseBase(new GlobalIndexParams());
```

Puis, un nouveau raisonnement est lancé après avoir instancié et initialisé un cas cible.

```
CbrCase target = factory.newTagetCase();
initialize(target);
```

```
Reasoning reasoning = reasoner.startReasoning(target);
```

L'utilisation de l'objet sauvant l'état du raisonnement est alors très pratique pour effectuer un rapport d'exécution (cf. Tableau C-2).

```
printReport(reasoning);
```

```
started at Mon Jul 13 17:34:05 CEST 1998
done in 460 ms

1/ retrieve done in 240 ms
found 5 cases
caseid=201 usefulness=0.86 risk=2.0
caseid=197 usefulness=0.86 risk=2.0
caseid=184 usefulness=0.86 risk=2.0
caseid=3 usefulness=0.87 risk=1.0
caseid=205 usefulness=0.96 risk=1.0

2/ reuse done in 20 ms
assigned risk by adaptation is 2.0

3/ revise not done

4/ retain done in 140 ms
 target case not added
```

Tableau C-2 : rapport d'exécution du premier raisonnement

3.2 Deuxième raisonnement

Dans le deuxième raisonnement, nous nous proposons d'utiliser la deuxième stratégie d'indexation (hiérarchie de prototypes et plus proches voisins). Nous passons également le nombre maximum de cas à retrouver de 5 initialement à 10. Le rapport (cf. Tableau C-3) montre alors un temps d'exécution divisé par deux par rapport au premier raisonnement pour l'obtention du même résultat.

```
((BasicRetrievePattern)reasoner.getRetrieveStep()).getDefaultParams().
        setSelectedIndex(reasoner.getMemory().getIndexBase().getIndex("Proto+KNN"));
((KnnIndex)reasoner.getMemory().getIndexBase().getIndex("CarCaseKnnAfterProto")).setK(10);
target = factory.newTagetCase();
initialize(target);
reasoning = reasoner.startReasoning(target);
```

```
started at Mon Jul 13 17:34:06 CEST 1998
done in 230 ms

1/ retrieve done in 70 ms
found 10 cases
caseid=201 usefulness=0.86 risk=2.0
caseid=199 usefulness=0.84 risk=2.0
caseid=197 usefulness=0.86 risk=2.0
caseid=184 usefulness=0.86 risk=2.0
caseid=181 usefulness=0.84 risk=3.0
caseid=179 usefulness=0.84 risk=3.0
caseid=3 usefulness=0.87 risk=1.0
caseid=205 usefulness=0.96 risk=1.0
caseid=2 usefulness=0.83 risk=1.0
caseid=203 usefulness=0.86 risk=2.0

2/ reuse done in 10 ms
assigned risk by adaptation is 2.0

3/ revise not done

4/ retain done in 130 ms
 target case not added
```

Tableau C-3 : rapport d'exécution du deuxième raisonnement

4 Code source complet de l'exemple

L'exemple représente la définition de dix classes dont nous donnons l'intégralité du code Java correspondant dans l'ordre alphabétique : CarCase, CarCaseSituation, CarFileSimpleCaseBase, CarIndexBase, CarReasonerFactory, CarRetain, CarRetrieve, CarReuse, CarsApp, CarSimilarity.

4.1 Classe CarCase

```
package aid.cbr.samples.cars;

import aid.cbr.tools.core.cbrcase.CompoundIndice;
import aid.cbr.tools.core.cbrcase.AbstractCbrCase;

public class CarCase extends AbstractCbrCase {

   public int risk = 0;
   public CarCaseSituation indices = new CarCaseSituation();
   public CompoundIndice getPotentialIndices() {
     return indices;
   }
}
```

4.2 Classe CarCaseSituation

```
package aid.cbr.samples.cars;

import aid.cbr.tools.core.cbrcase.*;

public class CarCaseSituation extends JavaClassIndice {

   // these fields are used to store indices
   public int normalizedLosses;
```

```
    public String make;
    public int fuelType;
    public int aspiration;
    public int numOfDoors;
    public String bodyStyle;
    public int driveWheels;
    public int engineLocation;
    public float wheelBase;
    public float length;
    public float height;
    public float width;
    public float curbWeight;
    public String engineType;
    public int numOfCylinders;
    public float engineSize;
    public String fuelSystem;
    public float bore;
    public float stroke;
    public float compressionRatio;
    public float horsePower;
    public float peakRpm;
    public float cityMpg;
    public float highwayMpg;
    public float price;

    static private  int _fuel []  = {1,2};
    static private  int _doors []  = {2,4};
    static private  int _asp [] = {1,2};
    static private  int _cylinders []  = {2,3,4,5,6,8,12};
    static private  int _wheels [] = {1,2,3};
    static private  int _loc [] = {1,2};

    static {
        ArrayIndiceDescriptor desc =
         new ArrayIndiceDescriptor("aid.cbr.samples.cars.CarCaseSituation");
         desc.addIndiceDescriptor(new IndiceDescriptor("normalizedLosses", new RangeIntType(65,256)));
       desc.addIndiceDescriptor(new IndiceDescriptor("make", StringType.STRING));
       desc.addIndiceDescriptor(new IndiceDescriptor("fuelType", new ListIntType(_fuel)));
       desc.addIndiceDescriptor(new IndiceDescriptor("aspiration", new ListIntType(_asp)));
       desc.addIndiceDescriptor(new IndiceDescriptor("numOfDoors", new ListIntType(_doors)));
       desc.addIndiceDescriptor(new IndiceDescriptor("bodyStyle", StringType.STRING ));
       desc.addIndiceDescriptor(new IndiceDescriptor("driveWheels", new ListIntType(_wheels)));
       desc.addIndiceDescriptor(new IndiceDescriptor("engineLocation", new ListIntType(_loc) ));
       desc.addIndiceDescriptor(new IndiceDescriptor("wheelBase", new RangeFloatType(86,121)));
       desc.addIndiceDescriptor(new IndiceDescriptor("length", new RangeFloatType(140,210)));
       desc.addIndiceDescriptor(new IndiceDescriptor("height", new RangeFloatType(60,75)));
       desc.addIndiceDescriptor(new IndiceDescriptor("width", new RangeFloatType(47,60)));
       desc.addIndiceDescriptor(new IndiceDescriptor("curbWeight", new RangeFloatType(1488,4066)));
       desc.addIndiceDescriptor(new IndiceDescriptor("engineType", StringType.STRING));
       desc.addIndiceDescriptor(new IndiceDescriptor("numOfCylinders",new ListIntType(_cylinders)) );
       desc.addIndiceDescriptor( new IndiceDescriptor("engineSize", new RangeFloatType(60,326)));
       desc.addIndiceDescriptor( new IndiceDescriptor("fuelSystem",StringType.STRING));
       desc.addIndiceDescriptor( new IndiceDescriptor("bore", new RangeFloatType(2,4)));
       desc.addIndiceDescriptor( new IndiceDescriptor("stroke", new RangeFloatType(2,5)));
       desc.addIndiceDescriptor( new IndiceDescriptor("compressionRatio", new RangeFloatType(7,23)));
       desc.addIndiceDescriptor( new IndiceDescriptor("horsePower", new RangeFloatType(48,288)));
       desc.addIndiceDescriptor( new IndiceDescriptor("peakRpm", new RangeFloatType(4150,6600)));
       desc.addIndiceDescriptor( new IndiceDescriptor("cityMpg", new RangeFloatType(13,49)));
       desc.addIndiceDescriptor( new IndiceDescriptor("highwayMpg", new RangeFloatType(16,54)));
       desc.addIndiceDescriptor( new IndiceDescriptor("price", new RangeFloatType(5000,50000)));
    }
}
```

4.3 Classe CarFileSimpleCaseBase

```
package aid.cbr.samples.cars;
import aid.cbr.tools.core.memory.casebase.*;
import aid.cbr.tools.core.cbrcase.*;
import java.util.StringTokenizer;
import java.util.NoSuchElementException;

public class CarFileSimpleCaseBase extends FileSimpleCaseBase {
  public CarFileSimpleCaseBase (String name,String filename ) {
      super (name,filename);
  }

  protected void setFloat(String name, JavaClassIndice s, String token){
      int i = s.indexOf(name);
      if (token.equals("?")) {
       s.setUnknown(i);
      } else {
         s.setFloatIndice(Float.valueOf(token).floatValue(),i);
      }
  }
  protected void setInt(String name, JavaClassIndice s, String token){
      int i = s.indexOf(name);
      if (token.equals("?")) {
         s.setUnknown(i);
      } else {
         s.setIntIndice(Integer.parseInt(token),i);
      }
  }
  protected void setSymbol(String name, JavaClassIndice s, String token){
      int i = s.indexOf(name);
      if (token.equals("?")) {
         s.setUnknown(i);
      } else {
         s.setIndice(token,i);
      }
  }

  protected CbrCase stringToCase(String  l) throws CaseBaseAccessException{
      CarCase c= new CarCase();
      CarCaseSituation s = (CarCaseSituation)c.getPotentialIndices();

      StringTokenizer t = new StringTokenizer(l,",");
      try {
         //caseid
         c.setCaseId(new Integer(Integer.parseInt(t.nextToken())));
         // risk
         String token = t.nextToken().trim();
         if (token.equals("?")) throw new CaseBaseAccessException ("Token cannot be unknown");
         c.risk= Integer.parseInt(token);

         // potential indices
         setInt("normalizedLosses",s,t.nextToken().trim());
         setSymbol("make",s,t.nextToken().trim());
         setInt("fuelType",s,t.nextToken().trim());
         setInt("aspiration",s,t.nextToken().trim());
         setInt("numOfDoors",s,t.nextToken().trim());
         setSymbol("bodyStyle",s,t.nextToken().trim());
         setInt("driveWheels",s,t.nextToken().trim());
         setInt("engineLocation",s,t.nextToken().trim());
         setFloat("wheelBase",s,t.nextToken().trim());
         setFloat("length",s,t.nextToken().trim());
         setFloat("height",s,t.nextToken().trim());
         setFloat("width",s,t.nextToken().trim());
         setFloat("curbWeight",s,t.nextToken().trim());
```

```
            setSymbol("engineType",s,t.nextToken().trim());
            setInt("numOfCylinders",s,t.nextToken().trim());
            setFloat("engineSize",s,t.nextToken().trim());
            setSymbol("fuelSystem",s,t.nextToken().trim());
            setFloat("bore",s,t.nextToken().trim());
            setFloat("stroke",s,t.nextToken().trim());
            setFloat("compressionRatio",s,t.nextToken().trim());
            setFloat("horsePower",s,t.nextToken().trim());
            setFloat("peakRpm",s,t.nextToken().trim());
            setFloat("cityMpg",s,t.nextToken().trim());
            setFloat("highwayMpg",s,t.nextToken().trim());
            setFloat("price",s,t.nextToken().trim());

            return c;
        } catch (NoSuchElementException e) {
            throw new CaseBaseAccessException("Token missing");
        }
    }
    protected String caseToString(CbrCase c){
        CarCaseSituation s = (CarCaseSituation)c.getPotentialIndices();
        StringBuffer buff = new StringBuffer(255);
        buff.append(c.getCaseId()); buff.append(",");
        buff.append(((CarCase)c).risk);             buff.append(",");
        if (s.isUnknown(s.indexOf("normalizedLosses"))) buff.append("?"); else  buff.append(s.normalizedLosses);  buff.append(",");
        if (s.isUnknown(s.indexOf("make"))) buff.append("?"); else buff.append(s.make);          buff.append(",");
        if (s.isUnknown(s.indexOf("fuelType"))) buff.append("?"); else buff.append(s.fuelType);       buff.append(",");
        if (s.isUnknown(s.indexOf("aspiration"))) buff.append("?"); else buff.append(s.aspiration);     buff.append(",");
        if (s.isUnknown(s.indexOf("numOfDoors"))) buff.append("?"); else buff.append(s.numOfDoors);     buff.append(",");
        if (s.isUnknown(s.indexOf("bodyStyle"))) buff.append("?"); else buff.append(s.bodyStyle);      buff.append(",");
        if (s.isUnknown(s.indexOf("driveWheels"))) buff.append("?"); else buff.append(s.driveWheels);    buff.append(",");
        if (s.isUnknown(s.indexOf("engineLocation"))) buff.append("?"); else buff.append(s.engineLocation); buff.append(",");
        if (s.isUnknown(s.indexOf("wheelBase"))) buff.append("?"); else buff.append(s.wheelBase);       buff.append(",");
        if (s.isUnknown(s.indexOf("length"))) buff.append("?"); else buff.append(s.length);          buff.append(",");
        if (s.isUnknown(s.indexOf("height"))) buff.append("?"); else buff.append(s.height);          buff.append(",");
        if (s.isUnknown(s.indexOf("width"))) buff.append("?"); else buff.append(s.width);           buff.append(",");
        if (s.isUnknown(s.indexOf("curbWeight"))) buff.append("?"); else buff.append(s.curbWeight);      buff.append(",");
        if (s.isUnknown(s.indexOf("engineType"))) buff.append("?"); else buff.append(s.engineType);     buff.append(",");
        if (s.isUnknown(s.indexOf("numOfCylinders"))) buff.append("?"); else buff.append(s.numOfCylinders); buff.append(",");
        if (s.isUnknown(s.indexOf("engineSize"))) buff.append("?"); else buff.append(s.engineSize);      buff.append(",");
        if (s.isUnknown(s.indexOf("fuelSystem"))) buff.append("?"); else buff.append(s.fuelSystem);     buff.append(",");
        if (s.isUnknown(s.indexOf("bore"))) buff.append("?"); else buff.append(s.bore);            buff.append(",");
        if (s.isUnknown(s.indexOf("stroke"))) buff.append("?"); else buff.append(s.stroke);          buff.append(",");
        if (s.isUnknown(s.indexOf("compressionRatio"))) buff.append("?"); else buff.append(s.compressionRatio);buff.append(",");
        if (s.isUnknown(s.indexOf("horsePower"))) buff.append("?"); else buff.append(s.horsePower);      buff.append(",");
        if (s.isUnknown(s.indexOf("peakRpm"))) buff.append("?"); else buff.append(s.peakRpm);         buff.append(",");
        if (s.isUnknown(s.indexOf("cityMpg"))) buff.append("?"); else buff.append(s.cityMpg);         buff.append(",");
        if (s.isUnknown(s.indexOf("highwayMpg"))) buff.append("?"); else buff.append(s.highwayMpg);      buff.append(",");
        if (s.isUnknown(s.indexOf("price"))) buff.append("?"); else buff.append(s.price);
        return new String(buff);
    }
}
```

4.4 Classe CarIndexBase

```
package aid.cbr.samples.cars;

import aid.cbr.tools.core.cbrcase.*;
import aid.cbr.tools.core.memory.similarity.*;
import aid.cbr.tools.core.memory.index.*;

public class CarIndexBase extends BasicIndexBase {

    CarIndexBase() {
        // First strategy: knn
```

```
        buildAndDeclareKnn();

        // Second strategy: prototype + knn
        declarePrototype();
        declareKnnForPrototype();
        buildProtoKnn();
    }
    protected void buildAndDeclareKnn() {
        LinearKnnIndex knnIndex = new LinearKnnIndex(new CmpFactorValueOrder(),new CarSimilarity(),5);
        knnIndex.setName("CarCaseKnn");
        knnIndex.setComments("This index uses Knn method to retrieve car cases");
        knnIndex.setUnknownCmpValue(new CmpFactorValue((float)0.5));
        addTopLevelIndex(knnIndex);
    }

    protected void declarePrototype() {
        PrototypeIndex protoIndex = new PrototypeIndex();
        protoIndex.setName("ProtoIndex");
        protoIndex.setAddCaseToAcceptedPrototypes(true);

        Prototype p1 = new BooleanPrototype () { // root prototype
          public CmpValue computeMatching(CompoundIndice indices) {
                return CmpBoolValue.TRUE;
          }
        };
        p1.setName("Root");
        protoIndex.addPrototype(p1);

        Prototype p2 = new BooleanPrototype () { // expensive cars prototype
          public CmpValue computeMatching(CompoundIndice indices) {
                int priceIndex = ((JavaClassIndice)indices).indexOf("price");
                if (!indices.isUnknown(priceIndex)) {
                    float price = indices.getFloatIndice(priceIndex);
                    return (price > 10000)?CmpBoolValue.TRUE:CmpBoolValue.FALSE;
                }
                return CmpBoolValue.TRUE;
          }
        };
        p2.setName("ExpensiveCars");
        protoIndex.addPrototype(p2);

        Prototype p3 = new BooleanPrototype () { // big cars prototype
          public CmpValue computeMatching(CompoundIndice indices) {
                int lengthIndex = ((JavaClassIndice)indices).indexOf("length");
                int widthIndex = ((JavaClassIndice)indices).indexOf("width");
                int heightIndex = ((JavaClassIndice)indices).indexOf("height");
                if ( !indices.isUnknown(lengthIndex) &&
                    !indices.isUnknown(widthIndex) &&
                    !indices.isUnknown(heightIndex)) {
                            float length = indices.getFloatIndice(lengthIndex);
                            float width = indices.getFloatIndice(widthIndex);
                            float height = indices.getFloatIndice(heightIndex);
                            return (length > 180 && height > 50 && width > 50)?CmpBoolValue.TRUE:CmpBoolValue.FALSE;
                    }
                return CmpBoolValue.TRUE;
          }
        };
        p3.setName("BigCars");
        protoIndex.addPrototype(p3);
```

```
        // build the tree
        p1.addSon(p2);
        p1.addSon(p3);
        protoIndex.setRoot(p1);

        addIndex(protoIndex);
    }

    protected void declareKnnForPrototype() {
        LinearKnnIndex knnIndex = new LinearKnnIndex(new CmpFactorValueOrder(),new CarSimilarity(),5);
        knnIndex.setName("CarCaseKnnAfterProto");
        knnIndex.setComments("This index uses Knn method to retrieve car cases");
        knnIndex.setUnknownCmpValue(new CmpFactorValue((float)0.5));
        addIndex(knnIndex);
    }

    protected void buildProtoKnn() {
        // connect the indexes
        ToDynamicKnnIndex index = new ToDynamicKnnIndex();
        index.setName("Proto+KNN");
        index.addIndex(getIndex("ProtoIndex"));
        index.addIndex(getIndex("CarCaseKnnAfterProto"));
        addTopLevelIndex(index);
    }
}
```

4.5 Classe CarReasonerFactory

```
package aid.cbr.samples.cars;

import aid.cbr.tools.core.reasoner.*;
import aid.cbr.tools.core.cbrcase.*;
import aid.cbr.tools.core.memory.index.*;
import aid.cbr.tools.core.memory.casebase.*;

public class CarReasonerFactory extends AbstractReasonerFactory {
    public CbrCase newTagetCase() {
        return new CarCase();
    }
    public CaseBase newCaseBase() {
        return new CarFileSimpleCaseBase("CarCaseBase","cars.txt");
    }
    public Retrieve newRetrieveStep() {
        return new CarRetrieve();
    }
    public Reuse newReuseStep() {
        return new CarReuse();
    }
    public Retain newRetainStep() {
        return new CarRetain();
    }
    public IndexBase newIndexBase() {
        return new CarIndexBase();
    }
}
```

4.6 Classe CarRetain

```
package aid.cbr.samples.cars;

import aid.cbr.tools.core.cbrcase.CbrCase;
import aid.cbr.tools.core.reasoner.*;
import aid.cbr.tools.core.memory.similarity.CmpFactorValue;

public  class CarRetain extends ResponsibilityChainRetainPattern {

   CarRetain() {
        RetainAnalyser a1 = new CaseBaseSizeRetainAnalyser(250);
        RetainAnalyser a2 = new SimilarityRetainAnalyser(new CmpFactorValue((float)0.95));
        a1.setNextAnalyser(a2);
        setAnalyser(a1);
   }
}
```

4.7 Classe CarRetrieve

```
package aid.cbr.samples.cars;

import aid.cbr.tools.core.reasoner.*;
import aid.cbr.tools.core.memory.index.*;
import aid.cbr.tools.core.memory.similarity.*;
import com.sun.java.util.collections.*;
import aid.util.assoc.*;

public class CarRetrieve extends BasicRetrievePattern {
   protected  void updateRetrieveResult(IndexResult result, RetrieveResult stepResult){
        List res= Assoc.getFactory().newListAssoc();
        res.addAll(((CaseSetIndexResult)result).retrievedCases());
        stepResult.setCases(res);
        for (Iterator it = ((CaseSetIndexResult)result).retrievedCases().iterator();
            it.hasNext(); ) {
           Object id = it.next();
           float f = ((CmpFactorValue)((CaseSimIndexResult)result).getCaseMatchingResults(id)).getFactor();
           ((UsefulnessRetrieveResult)stepResult).setCaseUsefulness(id,f);
        }
   }
}
```

4.8 Classe CarReuse

```
package aid.cbr.samples.cars;

import aid.cbr.tools.core.cbrcase.CbrCase;
import aid.cbr.tools.core.reasoner.*;

public  class CarReuse extends CaseSetFormulaReusePattern  {

   protected  float getSourceCaseSolution(CbrCase c){
        return ((CarCase)c).risk;
   }

   protected  void setTargetCaseSolution(CbrCase c, float sol){
        ((CarCase)c).risk=(int)sol;
   }
```

```
    CarReuse () {
        setFormula( new MeanAdaptationFormula() {
            public float compute(float[] solutions, float[] usefullness) {
                    float f = super.compute(solutions,usefullness);
                    return Math.round(f);
            }
        });
    }
}
```

4.9 Classe CarsApp

```
package aid.cbr.samples.cars;

import aid.cbr.tools.core.cbrcase.*;
import aid.cbr.tools.core.reasoner.*;
import aid.cbr.tools.core.memory.similarity.*;
import aid.cbr.tools.core.memory.index.*;
import aid.cbr.tools.core.memory.casebase.*;
import aid.cbr.tools.core.memory.*;
import com.sun.java.util.collections.*;
import java.util.Date;

public class CarsApp {
    public static void initialize(CbrCase c)throws  CaseBaseAccessException {
        CaseBase cb = new CarFileSimpleCaseBase("TargetCases","target.txt");
        cb.open(true);
        ((CarCase)c)._indices=(CarCaseSituation)cb.getCase(new Integer(1)).getPotentialIndices();
        cb.close();
    }

    public static void printReport(Reasoning r) throws  CaseBaseAccessException{
        StringBuffer buf = new StringBuffer(250);
        buf.append("\n\n*****Reasoning Report*****\n");
        buf.append("\nstarted at "+new Date(r.getBeginDate()));
        buf.append("\ndone in "+r.duration()+ " ms");

        appendRetrieveReport(r,buf);
        appendReuseReport(r,buf);
        appendReviseReport(r,buf);
        appendRetainReport(r,buf);

        //print the report
        System.out.print(buf);
    }
    public static void appendRetrieveReport(Reasoning r, StringBuffer buf) throws  CaseBaseAccessException{
        RetrieveResult retrieve = r.getRetrieveResult();
        if (retrieve!=null && retrieve.done()) {
            buf.append("\n\n1/ retrieve done in "+retrieve.duration()+ " ms");
            if (retrieve.isCaseRetrieved()) {
                    int nbCases = retrieve.cases().size();
                    buf.append("\nfound "+nbCases+" cases");
                    for (Iterator it = retrieve.cases().iterator();
                        it.hasNext();){
                        Object id = it.next();
                        float f =((UsefulnessRetrieveResult)retrieve).getCaseUsefulness(id);
                        float sol = ((CarCase)r.getMemory().getCaseBase().getCase(id)).risk;
                        buf.append("\ncaseid="+id+" usefulness="+f+ " risk="+sol);
                    }
            } else {
                    buf.append("\nno case found");
            }
        }else {
            buf.append("\n\n1/ retrieve not done");
```

```
        }
    }
    public static void appendReuseReport(Reasoning r, StringBuffer buf){
        ReuseResult reuse = r.getReuseResult();
        if (reuse!=null && reuse.done()) {
            buf.append("\n\n2/ reuse done in "+reuse.duration()+" ms");
            if (reuse.isAdaptationDone()) {
                    buf.append("\nassigned risk by adaptation is "+reuse.getAdaptedSolution());
            } else {
                    buf.append("\nadaptation not performed");
            }
        }else {
            buf.append("\n\n2/ reuse not done");
        }
    }

    public static void appendReviseReport(Reasoning r, StringBuffer buf) {
        ReviseResult revise = r.getReviseResult();
        if (revise!=null && revise.done()) {
            buf.append("\n\n3/ revise done in "+revise.duration()+" ms");
        }else {
            buf.append("\n\n3/ revise not done");
        }
    }
    public static void appendRetainReport(Reasoning r, StringBuffer buf) {
        RetainResult retain = r.getRetainResult();
        if (retain!=null && retain.done()) {
            buf.append("\n\n4/ retain done in "+retain.duration()+" ms");
            if (retain.isTargetCaseRetained())
                    buf.append("\n target case added");
            else
                    buf.append("\n target case not added");
        }else {
            buf.append("\n\n4/ retain not done");
        }
    }

    public static void main(String[] args) {
        try {

            // create the reasoner
            ReasonerFactory factory = new CarReasonerFactory();
            Reasoner reasoner = factory.getReasoner();

            // initialise the case base and the indexes
            reasoner.getMemory().getCaseBase().open();
            reasoner.getMemory().buildFromCaseBase(new GlobalIndexParams());

            // start a reasoning
            CbrCase target = factory.newTagetCase();
            initialize(target);
            Reasoning reasoning = reasoner.startReasoning(target);

            // print the report
            printReport(reasoning);

            // start a second reasoning
            ((BasicRetrievePattern)reasoner.getRetrieveStep()).getDefaultParams().
                    setSelectedIndex(reasoner.getMemory().getIndexBase().getIndex("Proto+KNN"));
            ((KnnIndex)reasoner.getMemory().getIndexBase().getIndex("CarCaseKnnAfterProto")).setK(10);
            target = factory.newTagetCase();
            initialize(target);
            reasoning = reasoner.startReasoning(target);

            // print the report
```

```
            printReport(reasoning);

            reasoner.getMemory().getCaseBase().close();


        } catch (CaseBaseAccessException expt) {
            expt.printStackTrace();
        } catch (MemoryAccessException expt) {
            expt.printStackTrace();
        } catch (ReasoningAbortedException expt) {
            System.out.println("Reasoner exited abormally");
            expt.printStackTrace();
        }
    }

}
```

4.10 Classe CarSimilarity

```
package aid.cbr.samples.cars;

import aid.cbr.tools.core.memory.similarity.*;
import aid.cbr.tools.core.cbrcase.*;

public class CarSimilarity extends CompoundIndiceArraySimilarity {
    public CarSimilarity () {
        super(new MeanAggregationFct(),25);
          Class c = JavaClassIndice.getClass("aid.cbr.samples.cars.CarCaseSituation");

        // indices with float similarity on float Domain
        SimpleIndiceSimilarity floatAtt = new SimpleIndiceSimilarity(new FloatDomainSimilarityFct());
        setSubSimilarity(CarCaseSituation.indexOf(c,"wheelBase"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"length"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"height"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"width"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"curbWeight"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"engineSize"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"bore"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"stroke"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"compressionRatio"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"horsePower"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"peakRpm"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"cityMpg"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"highwayMpg"),floatAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"price"),floatAtt);

          SimpleIndiceSimilarity intAtt = new SimpleIndiceSimilarity(new IntDomainSimilarityFct());
        setSubSimilarity(CarCaseSituation.indexOf(c,"normalizedLosses"),intAtt);
        setSubSimilarity(CarCaseSituation.indexOf(c,"numOfCylinders"),intAtt);

        // indices with equality Similarity
        SimpleIndiceSimilarity eqsim = new SimpleIndiceSimilarity(new IdentitySimilarityFct(    new CmpFactorValue(1),
                                                                                                 new CmpFactorValue(0)));
        setSubSimilarity(CarCaseSituation.indexOf(c,"make"),eqsim);
        setSubSimilarity(CarCaseSituation.indexOf(c,"fuelType"),eqsim);
        setSubSimilarity(CarCaseSituation.indexOf(c,"aspiration"),eqsim);
        setSubSimilarity(CarCaseSituation.indexOf(c,"numOfDoors"),eqsim);
        setSubSimilarity(CarCaseSituation.indexOf(c,"driveWheels"),eqsim);
        setSubSimilarity(CarCaseSituation.indexOf(c,"engineLocation"),eqsim);
        setSubSimilarity(CarCaseSituation.indexOf(c,"engineType"),eqsim);
        setSubSimilarity(CarCaseSituation.indexOf(c,"fuelSystem"),eqsim);

        // Body style
        SymetricExtensionSimilarityFct bodyStyleSim = new SymetricExtensionSimilarityFct(    new CmpFactorValue(0),
```

```
                                                                                    new CmpFactorValue(1));
        bodyStyleSim.define("hardtop","convertible", new CmpFactorValue((float)0.9));
        bodyStyleSim.define("hardtop","hatchback", new CmpFactorValue((float)0.5));
        bodyStyleSim.define("hardtop","sedan", new CmpFactorValue((float)0.5));
        setSubSimilarity(CarCaseSituation.indexOf(c,"bodyStyle"),new SimpleIndiceSimilarity(bodyStyleSim));

    }
}
```

Annexe D : paramètres et algorithmes de Broadway

Cette annexe décrit les différents paramètres de la modélisation de Broadway ainsi que deux algorithmes utilisés : l'instanciation du patron de cas potentiels et le filtrage sur les comportements élémentaires.

1 Paramètres de la modélisation

Pour chaque paramètre nous donnons son nom, une brève description, la valeur actuellement utilisée dans la modélisation.

Nom Paramètre	*Description*	*Valeur choisie*
V	Vecteur des variables d'observation des utilisateurs	(Adresse, Contenu, Evaluation Affichage)
$\nabla()$	Fonction d'égalité de deux URI	Cf. chapitre V
motSignificatif()	Fonction indiquant si un mot est significatif	Dictionnaire de 571 mots anglais et 347 mots français
racine()	Fonction donnant la racine d'un mot	Algorithme de Porter (Porter, 1980)
C	Vecteur de classes de mots d'une page HTML	(MotsClefs = métadonnées, MisEnValeur = titre + sous titres (1,2 et3) + description, Autre)
CP	Vecteur de poids des classes	(0.5, 0.3, 0.2)
$\Delta()$	Fonction donnant la discrimination des mots	$\forall\omega$ un mot, $\Delta(\omega)=1$
m	Nombre de mots clefs maximum pour une page	20
δ_{min}	Seuil de sélection d'un mot	0,7
E	Domaine d'évaluation explicite	{-1,0,1,2}
$a_{\max}$	Ration d'affichage maximum pour être significatif (secondes/mots)	2.4 (10 min/ 250 mots)

Tableau D-1 : paramètres d'observation

Nom Paramètre	*Description*	*Valeur choisie*
$l_{\min}$	Longueur minimale d'une navigation pour appliquer la patron de cas potentiels	3
p	Taille de la restriction en nombre de pages	3
e_{p}	Ensemble des valeurs d'évaluation pour la sélection des pages passées	{-1,2}
a_{p}	Seuil d'affichage pour la sélection des pages passées	0.48 (2min/250 mots)
τ_{loc}	Seuil de localisation des pages passées	0,9
φ	Fonction d'agrégation des évaluations explicite (x) et implicite (y)	0 si x=-1 (y/ $a_{\max}$) 0.66 si x=0 et y <= $a_{\max}$ 0,5 si x=0 et y > $a_{\max}$ 0,66 si x =1 1 si x=2
φ_f	Seuil de sélection des pages recommandées dans le patron de cas potentiels	0,7

Tableau D-2 : paramètres de représentation

Nom Paramètre	*Description*	*Valeur choisie*
k	Nombre de cas maximum à rechercher	20
τ	Seuil de similarité minimum	0,8
S	Vecteur de fonctions de similarité	$(s_{Adresse}, s_{Contenu}, s_{Evaluaution}, s_{Affichage})$
k_r	Nombre de cas maximum à rechercher sur la restriction	100
τ_r	Seuil de filtrage sur la restriction	0,5
MP	Matrice de poids pour la restriction	$\forall$i et $\forall$j, MP(i,j)=1
k_c	Nombre de cas maximum à rechercher sur les pages passées	50
τ_c	Seuil de filtrage sur les pages passées	0,5
VP	Vecteur de poids entre les variables pour une page passée	(1,1,1,1)
CP	Vecteur de poids entre la similarité des comportements et le ratio du nombre de comportements	(1, 0.2)
GP	Vecteur de poids pour l'agrégation finale	(0.5, 1, 0.5)

Tableau D-3 : paramètres de recherche

Nom Paramètre	*Description*	*Valeur choisie*
F	Vecteur de fonctions calculant les caractéristiques d'une page pour la réutilisation	cf. Chapitre V
FP	Vecteur de poids entre caractéristiques	(10,5,5,5,2,2)
r	Nombre de recommandations maximum	10
τ_r	Seuil de qualité minimum	0,2

Tableau D-4 : paramètres de réutilisation

Nom Paramètre	*Description*	*Valeur choisie*
e_n	Ensemble de valeurs d'évaluation que doit présenter au moins une page dans une navigation pour être sauvegardée	{-1, 2}
a_n	Seuil d'affichage minimum que doit présenter au moins une page dans une navigation pour être sauvegardée	0.48 (2min/250 mots)
f_{obs}	Taux minimum d'échec de chargement d'une page dans les 15 derniers jours rendant cette page obsolète	90%
τ_{obs}	Seuil de similarité minimum indiquant un changement de contenu, ne rendant pas la page obsolète	0,8
η	Taux minimum de pages obsolètes pour effacer une navigation	25%

Tableau D-5 : paramètres d'apprentissage et de maintenance

2 Algorithmes

Nous présentons les deux algorithmes importants utilisés dans Broadway : l'instanciation du patron de cas potentiels et le filtrage sur les comportements élémentaires. Notre objectif est d'améliorer la compréhension des mécanismes de Broadway et nous ne cherchons pas à décrire une version optimisée de ces deux algorithmes. L'expression de ces algorithmes utilise les fonctions sur les chroniques (cf. chapitre II, 2.1.3.3).

2.1 Instanciation du patron de cas potentiels

La fonction d'instanciation du patron de cas potentiels de Broadway prend comme argument la navigation courante et l'instant courant. De par la nature de l'unité de temps choisie, l'instant courant est un entier qui donne la position de la dernière page affichée dans la navigation. Les comportements élémentaires de la restriction ne sont pas instanciés par le patron. En effet, ces comportements sont directement gérés par les index lors de la recherche.

```
NouveauP (n : Navigation, i :  Entier) {
    CasPotentiel c = CasPotentiel(n,i) ;  // création du cas potentiel à retourner et initialise la composante instantanée
    ComposanteComportementale cc =
        c.Problème.SituationComportementale. ComposanteComportementale; // composante comportementale à construire
    Ensemble pages = ∅; // pages déjà examinées dans les itérations
    ComportementElémentaire ce = nil; // dernier comportement créé
    Entier n = 1; // numero du dernier comportement élémentaire créé

    Pour chaque k de i à 1 {
        URl p = Index(Chr(cc,Adresse),k);        // page
        MotsClefs m = Index(Chr(cc,Contenu),k); // mots clefs
        Entier e = Index(Chr(cc,Evaluation),k); // évaluation
        Réel a = Index(Chr(cc,Affichage),k); // ratio d'affichage
```

```
        si   p ∉ pages  et (e ∈ e_p ou a_max >= a > a_p)
        alors {
            pages = pages + p;
            ComportementElémentaire nce =   ComportementElémentaire(Chr(cc,Adresse), p,Ψ_B(.,Adresse,p,m));
            cc = cc + nce;
            cc= cc+  ComportementElémentaire(Chr(cc,Contenu), m, Ψ_R (.,f(.,n)));
            cc= cc+  ComportementElémentaire(Chr(cc,Evaluation), e, Ψ_R (.,f(.,n)));
            cc= cc+  ComportementElémentaire(Chr(cc,Affichage), a, Ψ_R (.,f(.,n)));
            n = n +  4;
            si ce ≠ nil alors  cc = cc + ContrainteAvant(nce, ce);
            ce=nce;
        }
    }

    pages = ∅ ;
    Pour chaque k de Len(Chr(cc,Adresse)) à i+1 {
        URI p = Index(Chr(cc,Adresse),k);
        si   p ∉ pages et φ( Index(Chr(cc,Evaluation), k), Index(Chr(cc,Affichage), k)) > φ_f
        alors {
            pages = pages + p;
            c.Solution = c.Solution + Recommandation(p, φ( Index(Chr(cc,Evaluation), k), Index(Chr(cc,Affichage), k))) ;
        }
    }
    retourne c;
}
```

2.2 Filtrage sur les comportements élémentaires

Le filtrage sur les comportements élémentaires est une étape complexe que nous précisons : identification des comportements, calcul de la similarité sur les pages puis sélection par plus proches voisins.

```
filtrage(casSources : Ensemble ; casCible : Cas) {
    Queue q = Queue(k_c, τ_c); // queue qui contient les meilleurs cas (plus proches voisins)
    Entier max = maxPages(casSources); // indique le nombre maximum de pages selectionnées dans les cas sources
    Pour chaque c dans casSources {
        // création d'une composante comportementale pour contenir les comportements homologues
        ComposantComportementale cc =
                ComposanteComportementale(casCible.Problème.Enregistrement, casCible.Problème.Instant);
        // ----- identification des comportements homologues
        Pour chaque comportement dans c.Probleme.SituationComportementale.ComportementElémentaires {
            cc=cc+ comportement.Ic(cc); // application de la contrainte de localisation
        }
        // ----- calcul de la similarité
        Réel sim = 0; // similarité
        Reel nombrePage = 0; // nombre de pages de la composante comportementale
        Entier k= 1; // indice d'itération sur les comportements élémentaires
        Tant que k <=  Card(c.Probleme.SituationComportementale.ComportementElémentaires) {
            si cc(k) ≠ [] alors {
                Reel s1= s_Adresse(cc(k), c.Probleme.SituationComportementale.ComportementElémentaires(k));
                Reel s2= s_Contenu(cc(k+1), c.Probleme.SituationComportementale.ComportementElémentaires(k+1));
                Reel s3= s_Evaluation(cc(k+2), c.Probleme.SituationComportementale.ComportementElémentaires(k+2));
                Reel s4= s_Affichage(cc(k+3), c.Probleme.SituationComportementale.ComportementElémentaires(k+3));
                sim = sim + g([s1,s2,s3,s4], VP);
            }
            nombrePage++;
            k= k+ 4;
        }
        sim = sim / nombrePage;
        // ----- agrégation avec le critère du nombre de comportements
```

```
            sim = g ([sim, nombrePage/max], CP);
            q.ajoute(c,sim);
        }
        retourne q;
    }
```

RÉSUMÉ. Dans de nombreux processus de résolution de problèmes (commande, prévision, assistance), la prise en considération des évolutions de variables d'observation est nécessaire. Dans cette thèse, nous proposons de faciliter l'analyse et l'utilisation du *raisonnement à partir de cas* (RàPC) à cette classe de problèmes. Le RàPC est un paradigme de résolution de problèmes basé sur la réutilisation par analogie d'expériences passées. Nous proposons, tout d'abord, le *modèle d'indexation par situations comportementales* permettant de réutiliser des cas suivant les comportements pertinents qui les caractérisent. Ce modèle d'indexation générique comprend un formalisme de représentation ainsi qu'un guide d'utilisation. Il permet la conception d'un meilleur processus de découverte des connaissances : extraction de cas potentiels, sauvegarde et mise à jour des cas potentiels sous la forme de cas concrets, et interprétation facilitée des cas concrets. Ensuite, pour gérer notre modèle et faciliter son utilisation au sein d'un cycle complet de raisonnement, nous proposons une *plate-forme à objets*, appelée CBR*Tools, intégrant des points d'ouverture spécialisables ainsi que des composants de base réutilisables. Notre modélisation repose sur la structuration des points d'ouverture suivant des cas d'utilisation, et sur l'utilisation systématique de patrons de conception permettant d'expliciter et de justifier les décisions de conception. Notre approche met ainsi en évidence des modèles à objets ouverts pour le RàPC et permet la capitalisation d'objets-métier RàPC. Enfin, nous avons appliqué nos travaux pour assister un groupe d'utilisateurs navigant sur le Web. Dans notre système, nommé Broadway, des expériences extraites des navigations passées sont réutilisées pour fournir des recommandations. Broadway est un exemple de notre approche originale d'assistance, dans laquelle nous proposons la réutilisation de comportements utilisateurs observés suivant un ensemble de variables.

MOTS CLEFS. Raisonnement à partir de cas (RàPC), indexation par historiques, plate-forme à objets, points d'ouverture, patrons de conception, recherche d'information sur le Web, système de recommandation.

Scheme and Object-Oriented Framework for Case Indexing by Behavioural Situations: Application to Assisted Web Browsing.

ABSTRACT. In many problem solving activities (control, forecast, assistance), it is necessary to take evolutions of observation variables into account. In this thesis, we propose to facilitate the analysis and the use of *case-based reasoning* (CBR) for this class of problems. CBR is a problem solving paradigm based on the reuse by analogy of past experiences. Firstly, we propose the *indexing scheme by behavioural situations* in order to reuse cases based on their relevant behaviours. This generic indexing scheme provides a representation formalism and guidelines. It allows the design of a better knowledge discovery process: extraction of potential cases, updating of potential cases saved as concrete cases, and facilitated interpretation of concrete cases. Secondly, in order to facilitate the management and the use of our scheme in a complete reasoning cycle, we propose an *object-oriented framework* called CBR*Tools that integrates specialisable hot spots and reusable components. The proposed model is based on use cases to structure hot spots, and on the systematic use of design patterns to comment and justify design decisions. Our approach proposes object-oriented models for CBR and allows the capitalisation of corporate objects for CBR. Finally, we have applied our work to assist a group of users browsing the Web. In our system, called Broadway, extracted experiences from past navigations are reused to provide recommendations. Broadway is an example of our original approach to aiding systems, where we propose to reuse behaviours of users observed through a set of variables.

KEYWORDS. Case-based reasoning (CBR), time series indexing, object-oriented framework, hot spots, design patterns, information retrieval on the Web, recommender system.