



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 outils²⁴ 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 cas²⁵, 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.

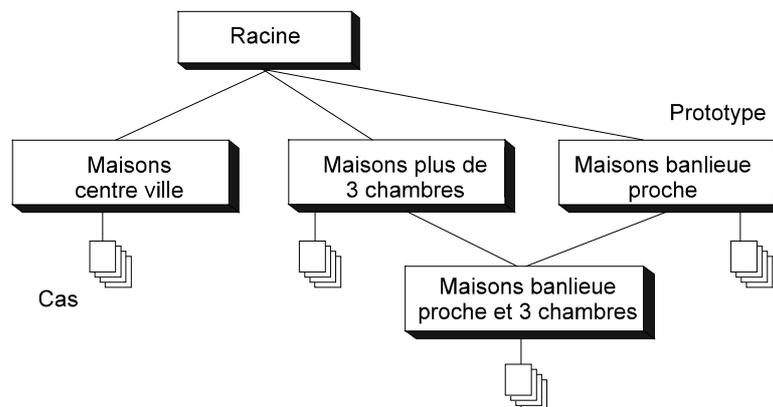


Figure III-1 : exemple d'une hiérarchie de prototypes

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.

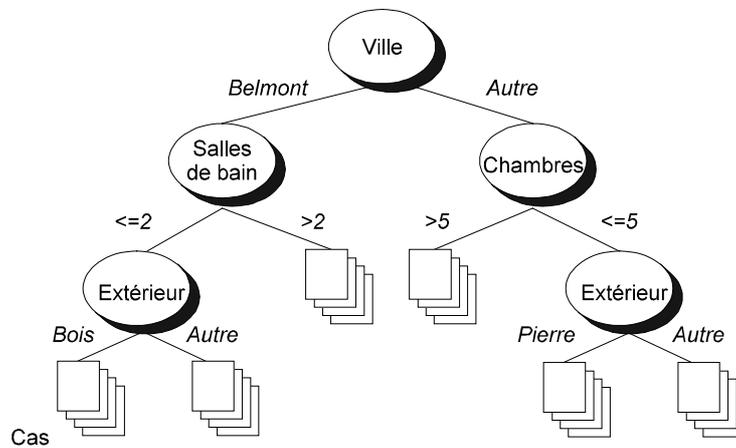


Figure III-2 : arbre d'induction

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 :

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

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 :

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

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.

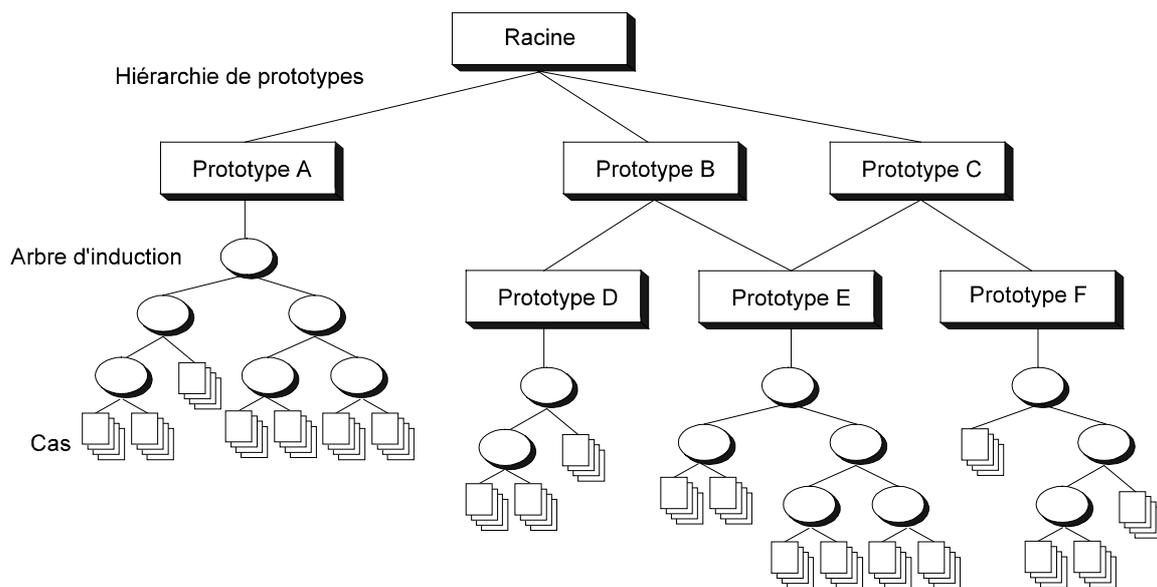


Figure III-3 : index hybride dans REMIND

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 ».

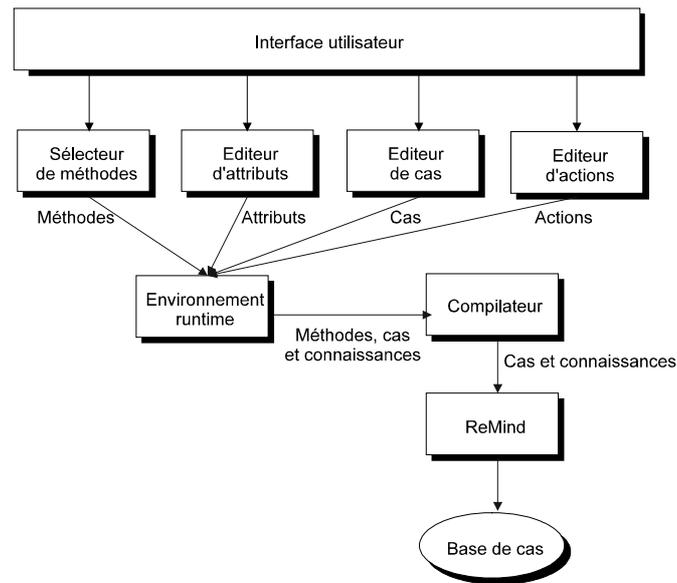


Figure III-4 : architecture de REPRO (Mark *et al.*, 1996)

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 (sélectionneur 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

KATE²⁶ 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 M5PSEEDIT. 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).

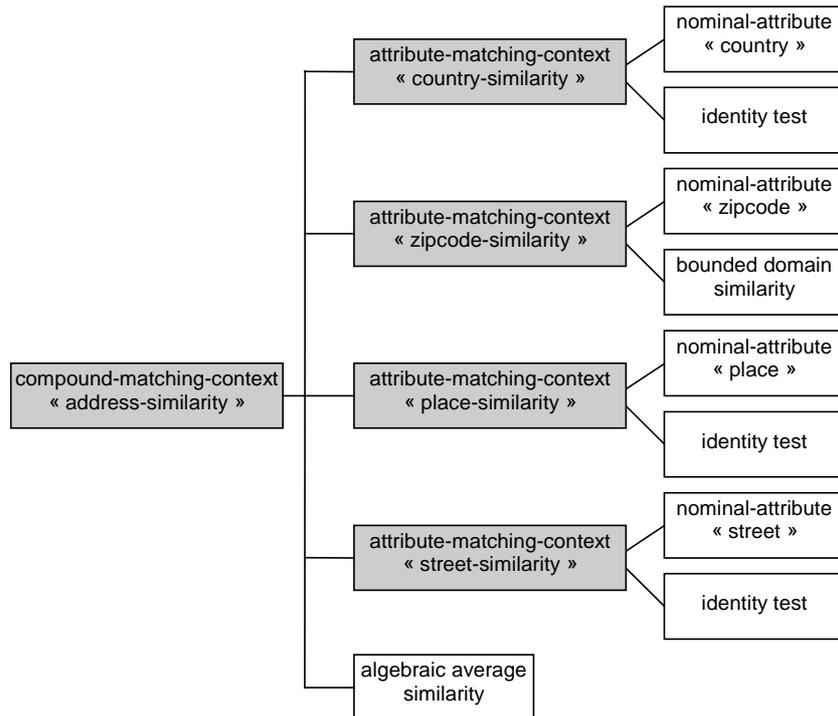


Figure III-6 : contexte du calcul d'une similarité dans RABIT (Gebhardt *et al.*, 1997, page 54)

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 M5PSEEDIT, 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 M5PSEEDIT. 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.

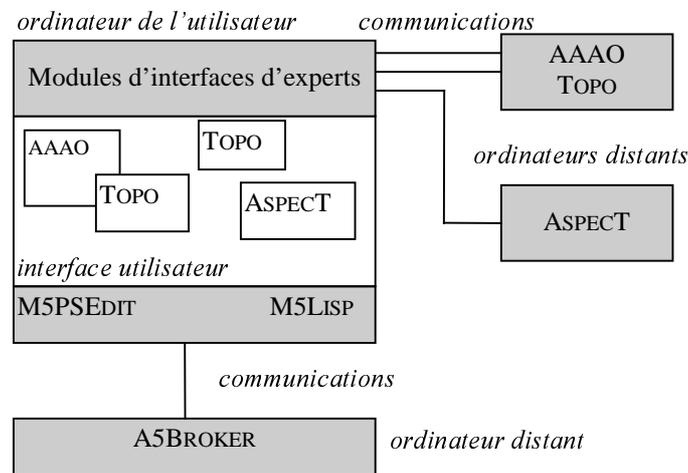


Figure III-7 : exemple de configuration distribuée du prototype FABEL (d'après Gebhardt *et al.*, 1997, page 205)

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 3²⁷ 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.

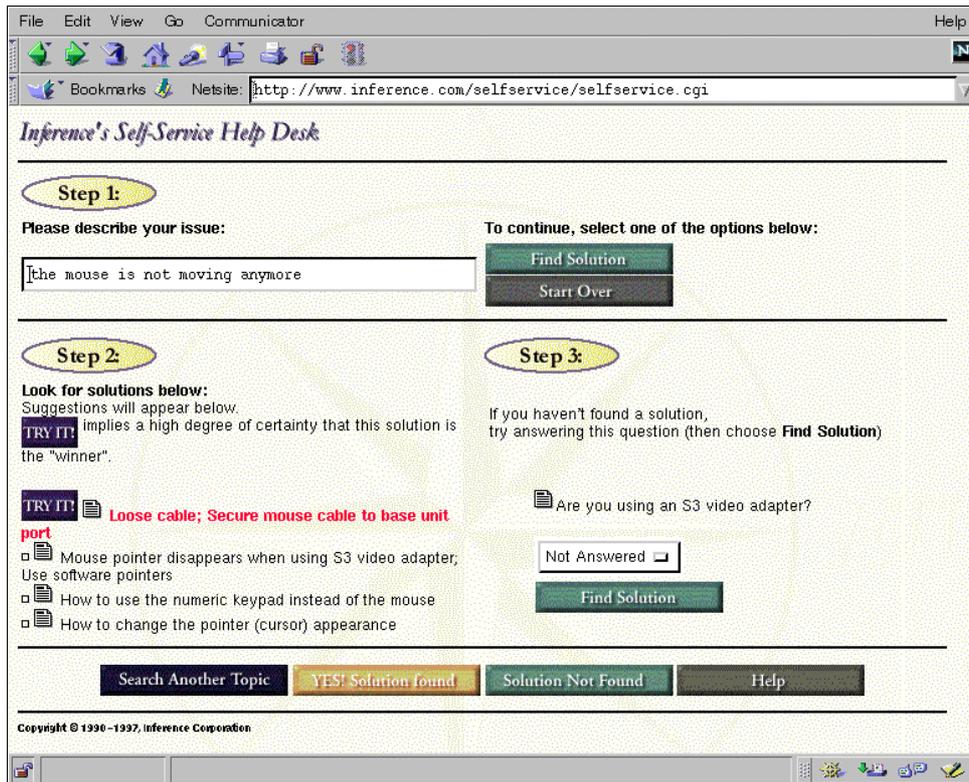


Figure III-8 : résolution d'un problème technique avec CasePoint WebServer

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 différents 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 Enterprise*, <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 CPR²⁸ (*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 cas²⁹ 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!CPR²⁸, 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.

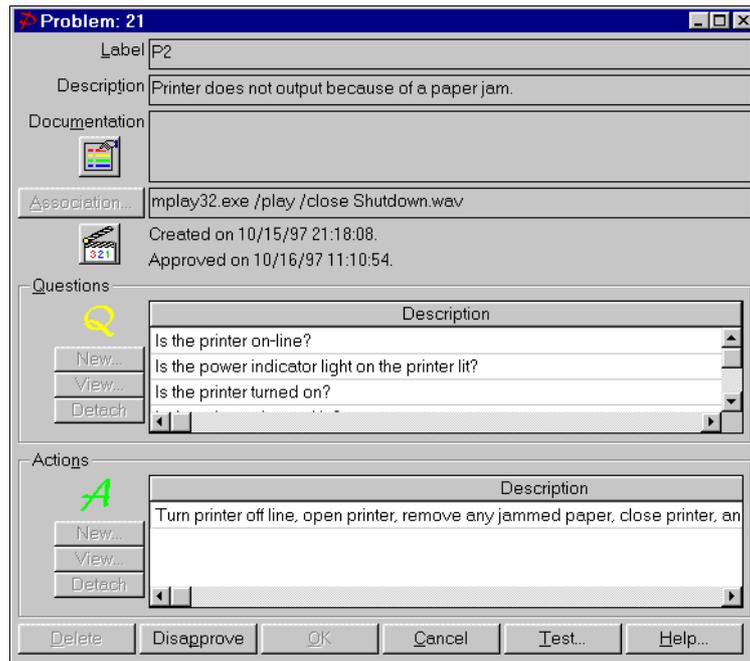


Figure III-9 : représentation d'un cas dans Help!CPR

Pour finir, nous citons brièvement d'autres outils qui sont plus amplement décrits dans (Watson, 1997) : ART*Entreprise, Esteem, Recall, CBRWorks et Caspian. ART*Entreprise³⁰ 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. Esteem³¹ est également un outil reposant sur un moteur d'inférence permettant d'utiliser des règles pour l'adaptation et propose différentes méthodes de recherche. Recall³² 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. CBRWorks³³ est un outil qui permet une représentation structurée des cas et dont les mesures de similarité peuvent être étendues. Enfin, Caspian³⁴ 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 des 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 Fédérale 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).

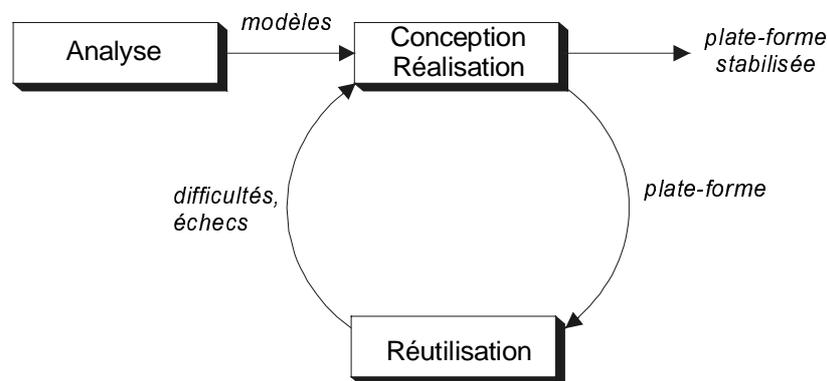


Figure III-11 : cycle de développement d'une plate-forme

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.

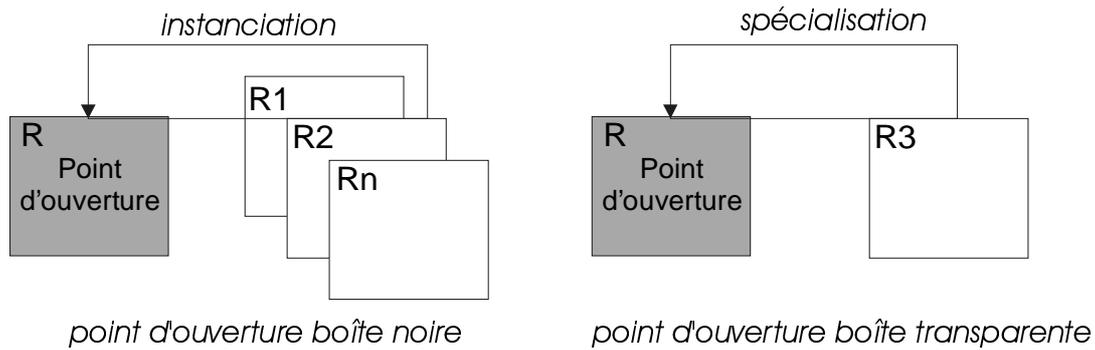


Figure III-12 : types de points d'ouverture (d'après Schmid, 1997)

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.

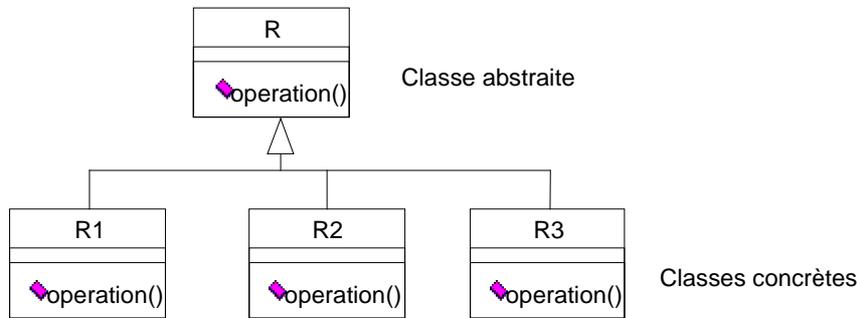


Figure III-13 : représentation d'un point d'ouverture

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).

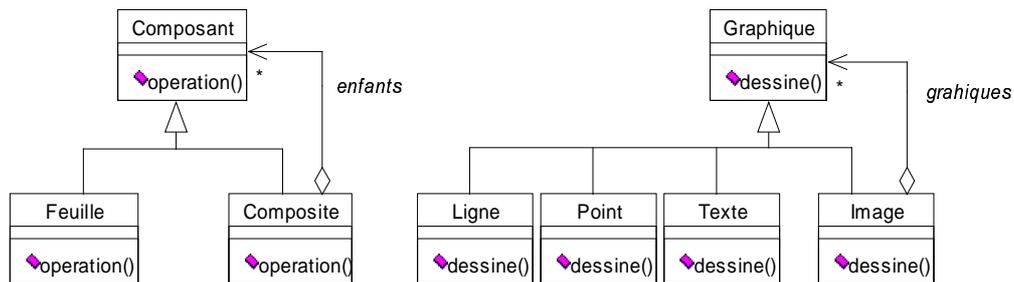


Figure III-14 : patron de conception *Composite* et son application pour des objets graphiques

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 instantiation 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*, Froedlich *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'ouverture³⁷

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.

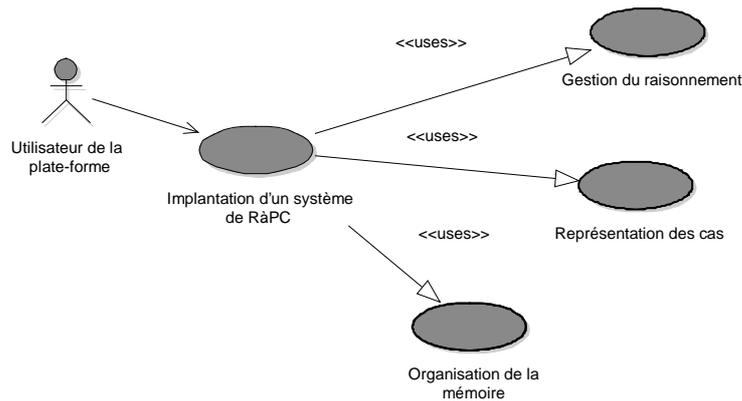


Figure III-15 : décomposition des points d'ouverture en trois axes de variabilité

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).

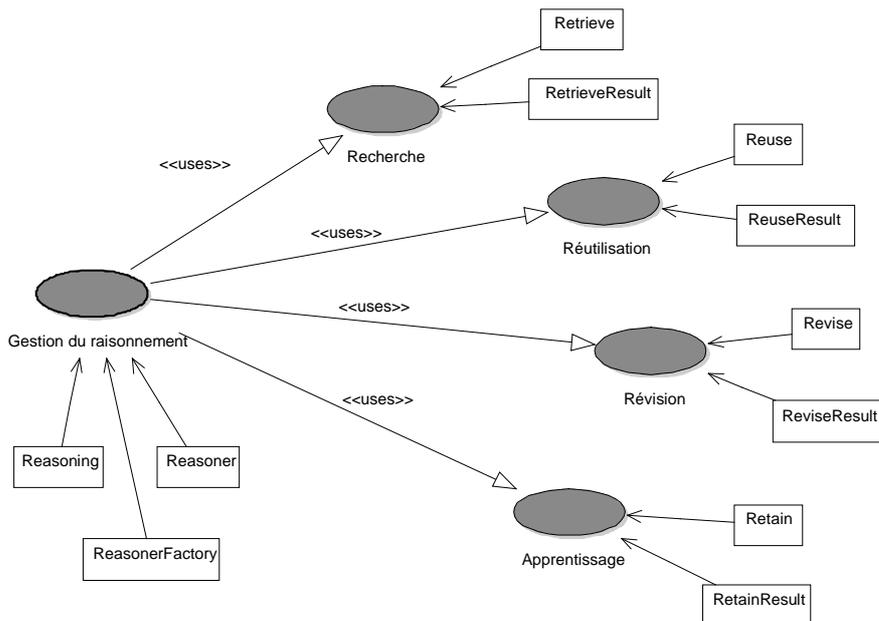


Figure III-16 : points d'ouverture pour la gestion du raisonnement

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 CompoundIndice³⁸). 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.

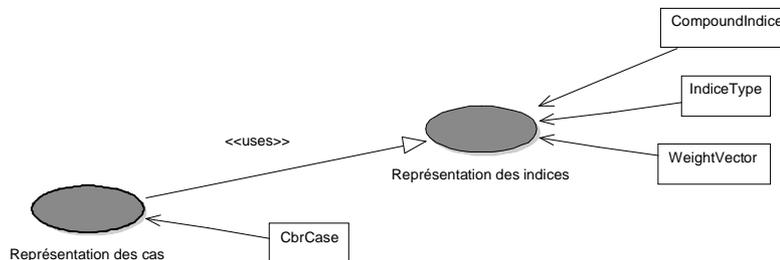


Figure III-17 : points d'ouverture pour la représentation des cas

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).

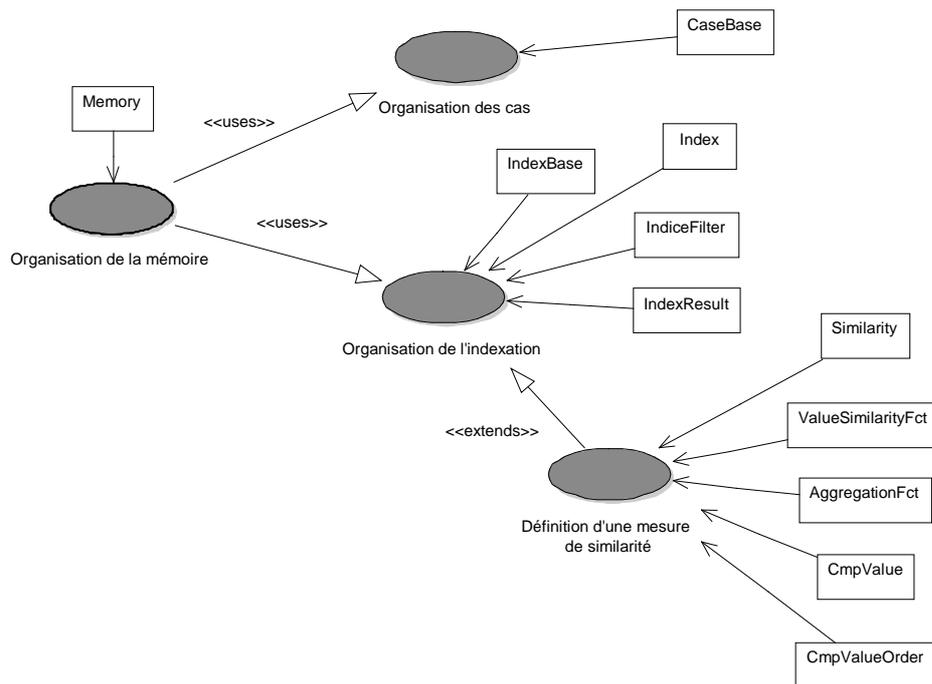


Figure III-18 : points d'ouverture pour l'organisation de la mémoire

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 utilisés (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).

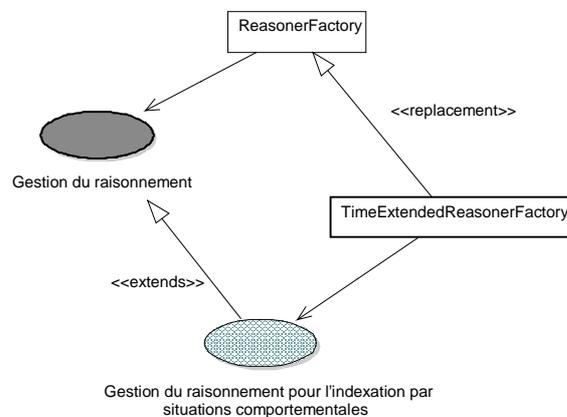


Figure III-19 : remplacement du point d'ouverture ReasonerFactory

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).

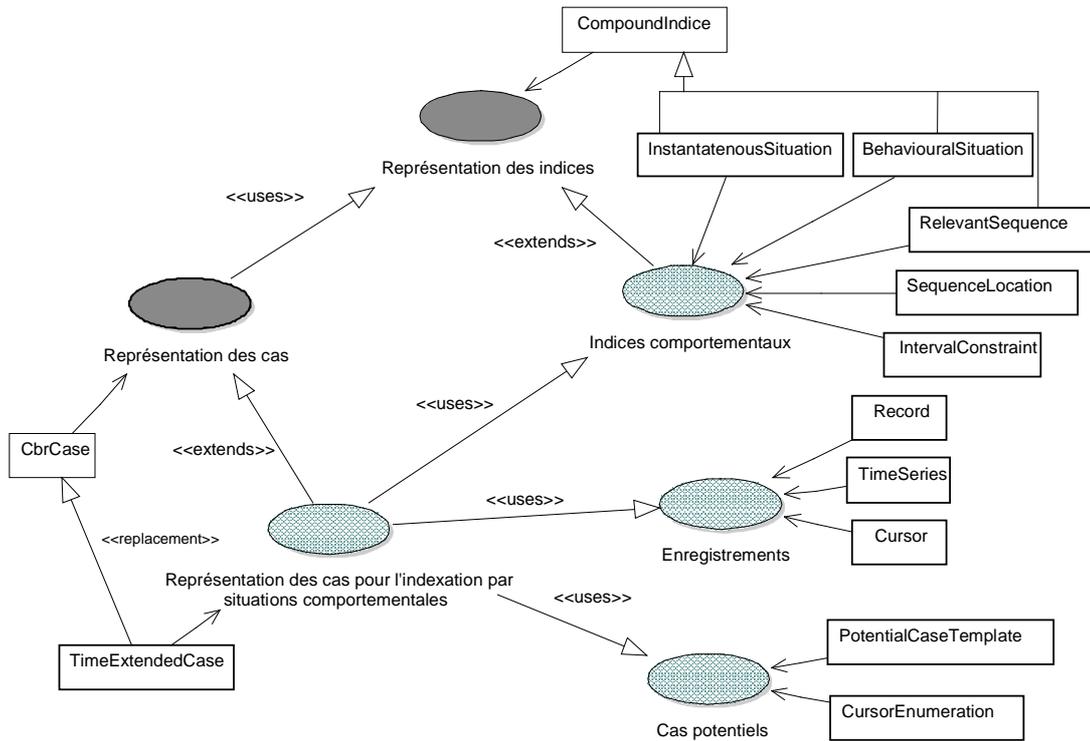


Figure III-20 : points d'ouverture pour la représentation des cas avec des indices comportementaux

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 être 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).

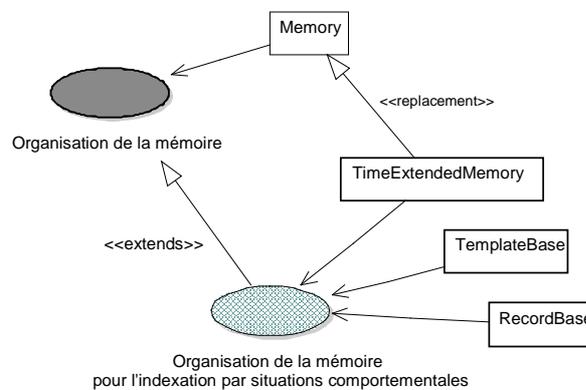


Figure III-21 : points d'ouverture pour l'organisation de la mémoire dans l'indexation par situations comportementales

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.

