MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE
LA RECHERCHE SCIENTIFIQUE
UNIVERSITE ABDELHAMID IBN BADIS MOSTAGANEM

Faculté des Sciences exactes et d'Informatique
Département de mathématiques et d'Informatique

Raisonnement à Partir de Cas En Utilisant le Systéme jCOLIBRI

Etudiants :
BENAMAR Fouad
BOUGUELMOUNA Fethi
Encadreur :
Mr. HENNI Fouad

Deuxième Année Master Ingénierie des Systèmes d'Information

Année Universitaire 2012/ 2013

Sommaire

Chapitre 2

Chapitre 3

Chapitre 4

Introduction Générale

Le plus grand objectif de l'intelligence artificielle consiste à permettre à un ordinateur de reproduire le raisonnement humain. Avec l'accroissement de la puissance de calcul et de la mémoire des machines modernes, il est devenu clair que ces ressources ne suffiraient pas, à elles seules, pour conférer l'intelligence à cet assemblage de puces électroniques, aussi sophistique soit-il.

Plusieurs techniques ont vu le jour pour tenter de rendre l'ordinateur plus intelligent, notamment les systèmes experts à base de règles, les moteurs d'inférence, ... Malheureusement, toutes ces approches nécessitent un expert connaisseur du domaine et voire même initié à l'intelligence artificielle pour programmer le système, pour lui fournir un ensemble exhaustif de règles ou de prédicats. Ce transfert de connaissance nécessite beaucoup de temps et d'efforts, ainsi qu'une connaissance approfondie du domaine traité. Un élément qui n'est pas couvert par le système expert ou le moteur d'inférence peut exiger la réécriture de plusieurs règles afin d'apporter une correction, afin de permettre l'apprentissage. Il nous faut une méthode de raisonnement plus proche de celle de l'être humain afin de reproduire l'intelligence humaine au sein d'une machine.

Le raisonnement à partir de cas (RàPC, ou CBR : Case Based Reasoning) est une nouvelle discipline qui a vu le jour depuis à peu près deux décennie. Son principe est de résoudre un problème nouveau en s'appuyant sur les résolutions de problèmes antérieurs similaires. On peut dire qu'un système qui applique le RàPC est un système qui apprend à chaque nouvelle expérience. Beaucoup de travaux ont été faits dans le domaine de RàPC. Certains de ces travaux ont pu concevoir des systèmes paramétrables qui permettent aux utilisateurs initiés de concevoir un système basé sur le RàPC appliqué à un domaine particulier. jCOLIBRI est un système de RàPC développé par des chercheurs de l'université de Madrid est mis à la disposition des développeurs de systèmes basés sur le RàPC.

Le but de ce projet est d'abord de comprendre les principes de développement d'un système RàPC, puis d'apprendre à utiliser une plateforme type (jCOLIBRI). Ensuite il sera question de modifier et d'ajouter quelques fonctionnalités à jCOLIBRI. A cet effet, nous allons nous concentrer sur le calcul de similarité entre cas, l'un des

problèmes les plus importants lors de la conception d'un système qui applique le principe du RàPC.

Organisation du mémoire

Au premier chapitre, nous introduisons des principes fondamentaux du raisonnement à partir de cas et des méthodes utilisées dans son cycle, et nous citons quelques exemples de systèmes de RàPC. Parmi les différentes applications possibles dans les systèmes de RàPC, nous allons choisir Jcolibri.

Au deuxième chapitre, nous présentons la plateforme Jcolibri comme une boite noire et sa configuration, la création de CBR application, Ontologies dans les applications CBR.

Au troisième chapitre, nous allons expliquer le processus de recherche en détaille, les classe de Jcolibri comme boite blanche, et citons quelques fonctions des similarités.

Au quatrième chapitre, nous présentons les outils utilisés pour notre travaille eclipse, Jcolibri,…….etc. et la présentation de l'application.

Liste des figures

Chapitre 1

Le raisonnement à partir de cas

1. Introduction

Au cours des dernières années, le raisonnement à partir de cas, ou RàPC, (Case-Based Reasonning, en anglais) est passé d'un domaine de recherche assez spécifique et isolé à un domaine d'intérêt généralisé .Les activités sont en croissance rapide-comme on le voitpar le nombreimportantde documents de recherche, la disponibilité des produits commerciaux,ainsi que les rapportssur les applicationsen cours d'utilisationrégulière.

2. Historique du RàPC : notions de base et notations

Ce chapitre présente brièvement les notions de base du RàPC qui sont utiles à la compréhension du reste de ce document, ainsi que les notations associées. Le RàPC a été introduit par Roger Schank et Janet Kolodner dans les années 80, dans la continuité des travaux de Schank sur la modélisation de la mémoirehumaine.

3. Le Raisonnement à Partir de Cas

Le RàPC est un processus qui vise la réutilisation des expériences passées. Cette méthodologie, provenant du domaine de l'Intelligence Artificielle, a été utilisée dans les systèmes experts et les sciences cognitives. Dans cette approche, l'utilisateur essaie de résoudre un nouveau problème en reconnaissant les similarités avec des problèmes préalablement résolus, appelés : cas. Un cas est communément un problème spécifique qui a été identifié, résolu, stocké et indexé dans une mémoire avec sa solution, et éventuellement le processus d'obtention de celle-ci. Les systèmes de RàPC sont appliqués dans de nombreux domaines comme : la médecine, le commerce, le diagnostic industriel, le contrôle et l'analyse financière[1].

4. Cycle du RàPC

Le RàPC dispose d'un cycle dont le nombre de phases varie selon les différentes sources bibliographiques. Les premiers auteurs à avoir décrit le cycle du RàPC sontAamodt et Plaza (1994) et le composent de quatre phases : la *recherche* de cas similaires (ou la *remémoration*), la *réutilisation* du cas trouvé(ou l'*adaptation*), la *révision* du cas sélectionné(ou la *validation*) etl'*apprentissage* (ou la *mémorisation*), voir **Figure 1** ci-dessous [1].

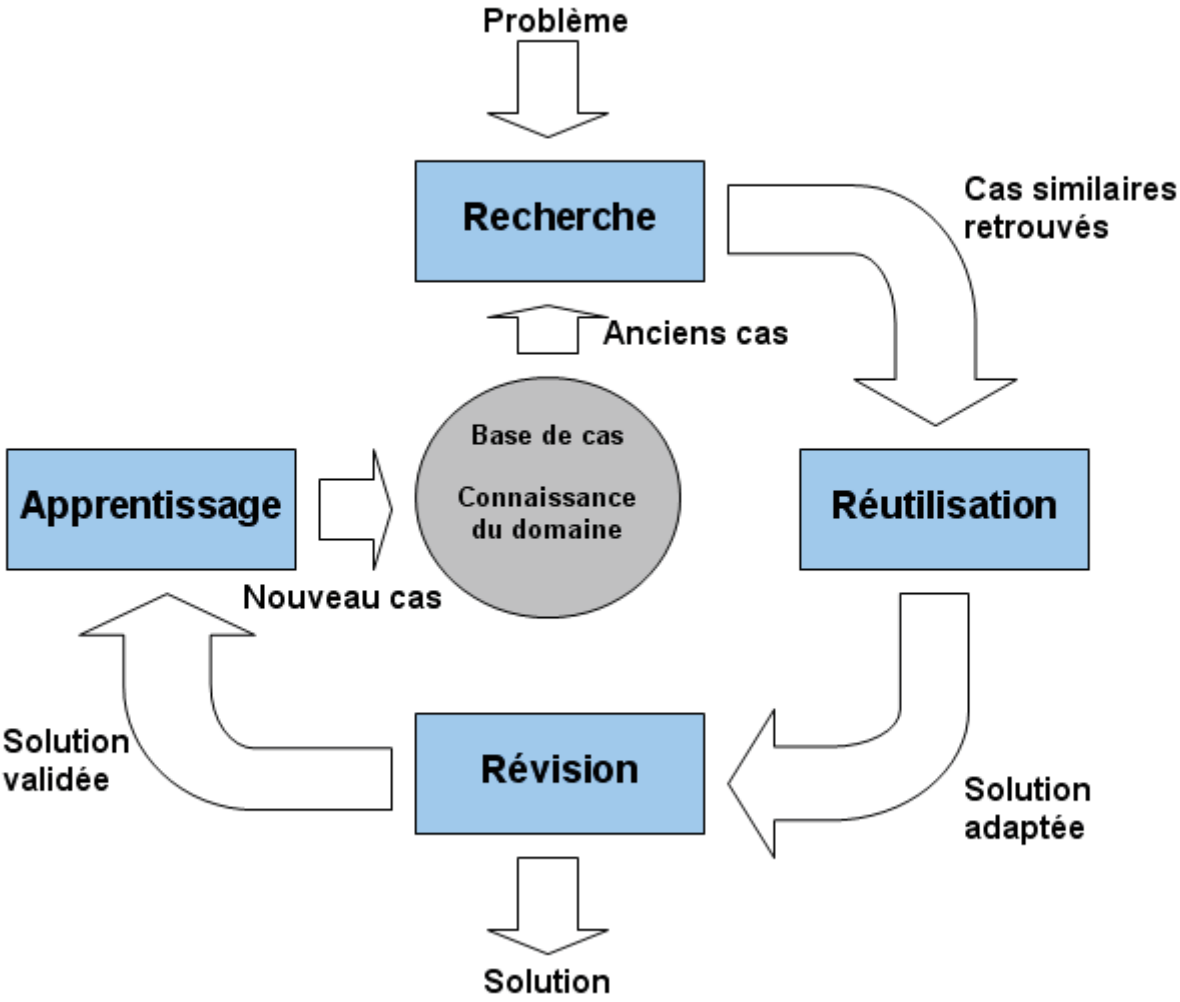


Figure 1: Le cycle du raisonnement à partir de cas [3]

4.1. Phase d'élaboration du cas

La phase d'élaboration concerne l'acquisition des informations sur le nouveau problème ce qui représente l'élaboration de sa description et se fait régulièrement au début du cycle d'utilisation du système RàPC. L'élaboration de la description du nouveau problème dépend essentiellement du modèle de raisonnement à partir de cas utilisés [1].

4.2.Phase de Recherche (la remémoration)

Cette phase consiste à rechercher dans la base de cas le ou les cas sources les plus proches à partir de la description de la partie problème du cas cible, qui vont être

utilisés pour le résoudre. Cette phase doit permettre d'obtenir la meilleure solution en effectuant les tâches suivantes : l'identification des caractéristiques pertinentes du problème, la remémoration et la sélection des meilleurs cas parmi les cas sources. L'exécution de ces tâches est dépendante de la représentation de cas, de leur indexation et de leur organisation dans la base de cas. Les méthodes de recherche se trouvent fortement inspirées de l'apprentissage machine[1].

4.2.1. Algorithmes de recherche des cas similaires

Plusieurs algorithmes ont été mis en place pour rechercher des cas appropriés. Citons quelques exemples :

- Les K plus proches voisins(KPPV)est la méthode la plus habituellement utilisée. Cette approche implique l'évaluation de la similarité entre un cas cible (nouveau problème) et un cas source de la base de cas. K est le nombre de cas sources voisins considérés comme étant proches (autour) du cas cible.

- L'induction basée sur la connaissanceutilisela connaissance dans le processus d'induction en identifiant manuellement les caractéristiques des cas.

- La recherche basée sur la structure(*templateretrieval*) est similaire aux requêtes SQL où l'algorithme recherche des cas correspondant a certains paramètres.

- Les approches inductivesdéterminent les caractéristiques pertinentes dans la discrimination des cas et génèrent une structure arborescente des décisions afin d'organiser les cas dans la mémoire[2].

4.3. Phase de Réutilisation(l'adaptation)

La phase d'adaptation dans le cycle du RàPC est le processus proposant une solution à un nouveau problème à partir des solutions appartenant aux cas sources remémorés. Cette phase peut se faire soit via une intervention humaine (manuelle) soit d'une manière automatique à l'aide d'algorithmes, de méthodes, de formules, de règles, etc.

Les principaux types d'adaptation automatique sont [1]:

- L'adaptation générative part du fait que nous disposons de toutes les connaissances pour résoudre le problème à partir de zéro. Le cas retrouvé retrace le raisonnement ayant mené à la solution ;
- L'adaptation transformationnelle est contraire à la précédente, c'est-à-dire qu'on ne dispose pas de toutes les connaissances pour résoudre le problème à partir de zéro ;
- L'adaptation compositionnelle utilise deux ou plusieurs cas similaires remémorés pour effectuer l'adaptation en composant les différentes solutions proposées ;
- L'adaptation hiérarchique où les cas sont organisés en plusieurs niveaux dans la hiérarchie de généralisation.

4.4. Phase de révision(la validation)

Au cours de la phase de révision, la solution proposée à l'issue de la phase d'adaptation sera évaluée. Cette évaluation concerne plusieurs actions pouvant être employées [1]:

- Tester la solution proposée dans le monde réel ;
- Faire une introspection dans la base de cas en utilisant l'ensemble des descripteurs de problème et de solution afin de vérifier que les cas similaires ont donné entière satisfaction ;
- Utiliser une autre méthode d'évaluation de la solution (simulateur, système expert classique, etc.). La phase de révision consiste donc à continuer éventuellement l'élaboration de la solution cible si besoin.

4.5. Phase d'apprentissage (la mémorisation)

Cette phase consiste à incorporer ce qui est utile à retenir dans la base de cas et permet de synthétiser les nouvelles connaissances qui vont être réutilisées ultérieurement. Cet apprentissage peut s'effectuer non seulement à partir du succès mais aussi de l'échec dans la résolution du problème cible. Le stockage d'un nouveau cas permet donc d'enrichir la base de cas permettant l'augmentation de l'expérience du système [1].

5. Représentation des cas

5.1. Définition du cas

Un cas est une expérience représentée par une connaissance. Cette expérience constitue une leçon permettant au système de RàPC de résoudre des problèmes de différentes natures. Selon le domaine d'application et les objectifs à atteindre, les informations contenues dans le cas varient. On peut définir un cas comme étant la description informatique d'un épisode de résolution de problème.

La définition d'un cas (dans la base de cas) passe par trois étapes :

La première étape concerne « *la synthèse* » qui consiste à trouver une structure permettant de satisfaire des spécifications. La deuxième étape concerne « l'*analyse* » qui, à partir d'une structure particulière, consiste à trouver le comportement associé. La troisième étape concerne « l'*évaluation* » qui consiste à vérifier que le comportement est conforme à ce qui est attendu.

Nous allons détailler la structure d'un cas et son indexation dans la base des cas selon plusieurs points vue existants dans la littérature[2].

5.2. Structure du cas

Tout d'abord, un cas en RàPC est généralement composé de deux espaces disjoints : L'espace des problèmes et l'espace des soulions. L'espace problème concerne la partie dans laquelle on trouve les objectifs à atteindre Quant à l'espace solution, il regroupe la description de la solution apportée par le raisonnement, sa justification, son évaluation ainsi que les étapes qui ont mené à cette solution.
On peut distinguer deux types de cas : cas source et cas cible .Le cas source est celui dans lequel les parties « problème » et « solution » sont renseignées .Donc, c'est un cas dont on va s'inspirer pour résoudre un nouveau problème .Le cas source peut aussi contenir une autre partie appelée « information de qualité ». Cette partie contient des informations sur l'utilisation du cas dans le système. Quant au cas cible, c'est celui qui porte le problème et dont sa partie solution n'est pas renseignée. Suivant la nature du problème à traiter, il existe plusieurs représentations de cas. Les approches traditionnelles les classent en trois catégories :

- La représentation textuelle ;
- La représentation semi structurée (vecteur de composants),
- La représentation structurée.

Cependant, la représentation structurée est la plus utilisée dans la majorité des travaux.

Ainsi, le cas est souvent représenté sous la forme d'un ensemble de descripteurs.Un descripteur « d » est défini par une paire d = (a, v) où « a » est un attribut et « v » est la valeur qui lui est associée.

Un cas source est représenté par un couple (srce, Sol(srce)) et le cas cible par le couple (cible, Sol(cible)), où Sol(cible) est inconnue et pour laquelle on voudrait lui apporter un résultat [2].

5.3. Indexation du cas

Les cas sont organisés dans une mémoire appelée base de cas. Afin de faciliter cette organisation et ainsi la recherche du cas le plus approprié au problème posé, il faut désormais les indexer. Il est à noter que lors de la recherche des cas, c'est la partie problème qui va être sollicitée. Or, cette partie problème est décrite par un ensemble de caractéristiques pertinentes nommées « indices ». Ces indices vont déterminer dans quels contextes et dans quelles situations les cas vont être recherchés et retrouvés pour les proposer au problème rencontré. Alors il faut trouver le moyen de bien manipuler ces indices pour une configuration optimale. Pour cela, il y a plusieurs méthodes d'indexation : manuelles ou automatiques. Dans le cas des méthodes manuelles, il est supposé que l'objectif d'utilisation des cas, et surtout des circonstances dans lesquelles les cas seront utiles, soit déterminé précisément. Toutefois, les méthodes d'indexation sont de plus en plus automatisées.

Par ailleurs, le choix des indices dépend du domaine d'application. La communauté du RàPC recommande que ces indices vérifientles propriétés suivantes :

• Prédictifs afin de jouer un rôle déterminant dans le choix d'une solution pour un nouveau problème ;

• Suffisamment abstraits pour que le cas ait la possibilité d'être utilisé plusieurs fois pour la résolution de plusieurs problèmes ;

• Suffisamment concrets pour que le cas soit reconnu le plus rapidement possible pour la résolution d'un nouveau problème [2].

6.Modèles de RàPC

Les systèmes utilisant le RàPC comme approche de résolution de problèmes peuvent disposer de plusieurs modèles. Ces modèles peuvent être regroupés en trois grandes familles à savoir : les modèles conversationnels, textuels et structurels [5].

6.1.Le modèle conversationnel

Le modèle conversationnel est essentiellement utilisé dans des systèmes de RàPC dédiés aux applications commerciales [5].

Un cas dans le modèle conversationnel a étédécritpar le triplet (Problème P, Questions QA, Action A) comme cela est montré sur la **Figure 2**:

Cas : 241
Titre : cartouche d'encre endommagée causant des traces noires
Description : l'imprimante laisse de petits points noirs sur les deu x côtés de la page.
Parfois des larges tâches couvrent également la région à imprimer.
Questions :
Est-ce que les copies sont de mauvaise qualité ? Réponse : oui Score : (-)
Quels types de problèmes avez-vous ? Réponse : trace noires Score : (default)
Est-ce qu'un nettoyage de l'imprimante règle le problème ? Rép : non …
Actions : vérifier la cartouche d'encre et la remplacer si le niveau d'encre est faible

Figure 2. Exemple de représentation d'un cas dans un modèle conversationnel

6.2.Le modèle textuel

Les premiers travaux sur le modèle textuel datent du milieu des années 90 et à ce jour, aucune représentation standard n'est apparue pour ce type de modèle. La description exacte du cas est primordiale dans ces modèles.Les cas dans les modèles textuels sont semi-structurés ou non-structurés :

• Les cas non-structurés sont les cas qui ne disposent que d'un seul attribut dont la description est complètement en texte libre (free-text) ;

• Les cas semi-structurés sont les cas qui disposent de plusieurs attributs étiquetés contenant du texte [5].

6.3.Le modèle structurel

Le modèle structurel est le premier modèle qui a été utilisé lors de l'apparition des premiers systèmes de RàPC. Les cas sont complètement structurés dans ce modèle de données et représentés par des paires <attribut, valeur>. Un attribut représente une spécification importante du domaine étudié. Quant à la valeur, elle vient structurer les attributs et elle est souvent exprimée par une échelle de valeur d'entiers, réels, booléens ou symboliques. Un exemple d'un cas représenté par la paire <attribut, valeur> est montré sur la **Figure 3**[5].

***Cas** : 1979*
Atelier : HMK
Nombre machines : 07
Horizon de travail : 25
Encours : 153
Débit production : 26.11
Date : 20/10/2007

Figure 3. Exemple de représentation d'un cas dans un modèle structurel de RàPC

7.Base de cas (la mémoire dans les systèmes de RàPC)

La base de cas représente le cœur du système de RàPC. Le bon fonctionnement et les performances d'un système de RàPC sont fortement liés à l'organisation de sa mémoire. En effet, la mémoire qui contient tous les cas sources précédemment retenus est appelée base de cas. La base de cas est un élément majeur dans l'indexation et l'organisation des cas afin de pouvoir les retrouver facilement et efficacement [2]. Nous pouvons distinguer deux types d'organisation de la base de cas :

7.1.Base de cas plate

Dans laquelle les cas sont organisés de manière linéaire (vecteur, tableau, graphe, etc). Autrement dit, les cas sont stockés dans une liste séquentielle. C'est sans doute l'organisation la plus simple. De plus, cette organisation est prise en compte dans la majorité des travaux de RàPC [2].

7.2.Base de cas hiérarchique

Dans laquelle la structuration et l'organisation des cas est faite selon des niveaux hiérarchiques donnés. La mémoire plate est considérée comme le niveau bas de la mémoire et elle est divisée en plusieurs groupes. Chaque groupe est représenté par un prototype dans le réseau. Quant à la mémoire hiérarchique, elle sert comme un système d'indexation pour les zones de la mémoire plate formant le haut niveau de la mémoire [2].

Lors de la création d'une base de cas, trois points principaux doivent être considérés:

- La structure et la représentation des cas ;
- Le modèle de la mémoire utilisée pour organiser la base de cas ;
- La sélection des indices qui sont utilisés pour identifier chaque cas.

8. Quelques systèmes de RàPC

8.1.PERSUADER

PERSUADER est un outil de gestion de conflits basé sur le raisonnement par cas. Il fonctionne sur le principe de négociation/médiation. Il est capable de fournir des solutions documentées pour la résolution de problèmes de groupe. PERSUADER fait en sorte de pouvoir construire un règlement mutuellement convenu entre les différents acteurs de la dispute [4].

8.2.ReMind

ReMind est actuellement l'un des outils les plus utilisés pour le CBR. Il permet d'importer des donnés à partir de base de données existantes.

ReMind permet l'utilisation de plusieurs moyens pour arriver à son but, comme les algorithmes de plus proches voisins, la construction d'arbres de décision, guidée par une connaissance extérieure ou non, mais ne permet pas de cumuler ces approches, ce qui est regrettable.

Un autre de ses atouts est de permettre un ajustement à l'aide de formules mathématiques, pour un apprentissage plus fiable.

Il est à noter que l'un de ses défauts principaux est qu'il a tendance à fournir des résultats peu fiables si les données possèdent de nombreuses valeurs nulles [4].

8.3.CBR-Works

CBR-Works a l'avantage de s'intéresser à la chaîne de traitement complète.Ce logiciel permet la représentation de cas complexes, propose des moyens de calcul de la similarité prédéfinis et permet également de définir de manière intuitive des calculs de similarité personnalisés [4].

8.4. Kate Suite

Kate-CBR est issu de la suite Kate bien connue, notamment pour son éditeur de texte. Ce logiciel utilise une détection des plus proches voisins combinée à une induction dynamique, qui propose à l'utilisateur de discriminer les différents cas de la base de

cas, pour ne pas qu'un cas « extraordinaire » n'ait une trop grande importance sur le résultat final [4].

8.5.jCOLIBRI

jCOLOBRI est un framework développé par un groupe de chercheurs de l'université de Madrid. Le groupe GAIA qui développe le logiciel jCOLIBRI est un groupe qui réalise des applications dans le domaine de l'intelligence artificielle. Leur cible est de trouver de nouvelles solutions pour l'enseignement assisté par ordinateur. [4]

Chapitre 2

Le jCOLIBRI CBR framework

9. Le jCOLIBRI CBR framework :

9.1. Définition :

jCOLIBRI est un framework orienté objet écrit en Java pour les systèmes pour la construction de système basés sur le RàPC. Il représente une évolution des travaux antérieurs de l'équipe GAIA (université de Madrid) sur le développement de systèmes d'apprentissage a des fins d'enseignement assisté par ordinateur. La première version du logiciel (COLIBRI1) a été prototypée en utilisant le langage fonctionnel LISP.

Le framework jCOLIBRI dispose de deux versions majeures :

jCOLIBRI version 1: version jCOLIBRI 1 est la première version du framework . Il comprend une interface utilisateur graphique complète qui guide l'utilisateur dans la conception d'un système RàPC. Cette version est recommandée pour les utilisateurs non-développeurs qui souhaitent créer des systèmes de RàPC sans programmer n'importe quel code [7].

jCOLIBRI version 2: version jCOLIBRI 2 est une nouvelle implémentation qui suit une nouvelle architecture claire et divisée en deux couches: l'une orientée vers les développeurs et l'autre orientée vers les concepteurs . Cette nouvelle conception est un framework complet présenté sous forme de boîte blanche ouverte aux développeurs Java qui souhaitent inclure les caractéristiques de jCOLIBRI dans leurs applications utilisant le RàPC [7].

9.2. Architecture de jCOLIBRI2

jCOLIBRI2 est le résultat de l'expérience acquise au cours de l'élaboration de la première version. Il résout de nombreux inconvénients comme la représentation des cas, la gestion des problèmes de développement, de métadonnées, … , Mais l'architecture de cette nouvelle version est très différente (bien compatible) comme il est basé sur une révision complète des systèmes de RàPC et les framework [7].

Un framework est un ensemble de classes qui incarne une conception abstraite de solutions à une famille de problèmes connexes. En d'autres termes, un framework est une conception partielle et mise en œuvre pour une application dans un domaine de problème donné.

Pour classer un framework, on peut distinguer deux types. Un framework boîte blanche est réutilisé pour la plupart par sous-classement et un framework boîte noire est réutilisé par paramétrage. Le développement habituel d'un framework commence avec une conception comme une architecture boîte blanche qui évolue vers une boîte noire. La boîte noire résultante comporte un générateur associé qui va générer le code de l'application. Le constructeur visuel permet au concepteur de logiciel pour connecter les objets frameworks et les activer. La première version de jCOLIBRI est plus proche d'un framework boîte noire avec le constructeur visuel et d'un manque évident au niveau de la structure de boîte blanche.

La nouvelle conception de jCOLIBRI2 a pour objectif de remodeler l'architecture dans un système boîte blanche clairement orientée pour les programmeurs, et une boîte noire avec une couche de constructeur qui est orientée vers les concepteurs. L'idée maîtresse de la nouvelle conception consiste à séparer les classes de base et l'interface utilisateur. Cette séparation nous donnera l'architecture à deux couches de la **figure 4**:

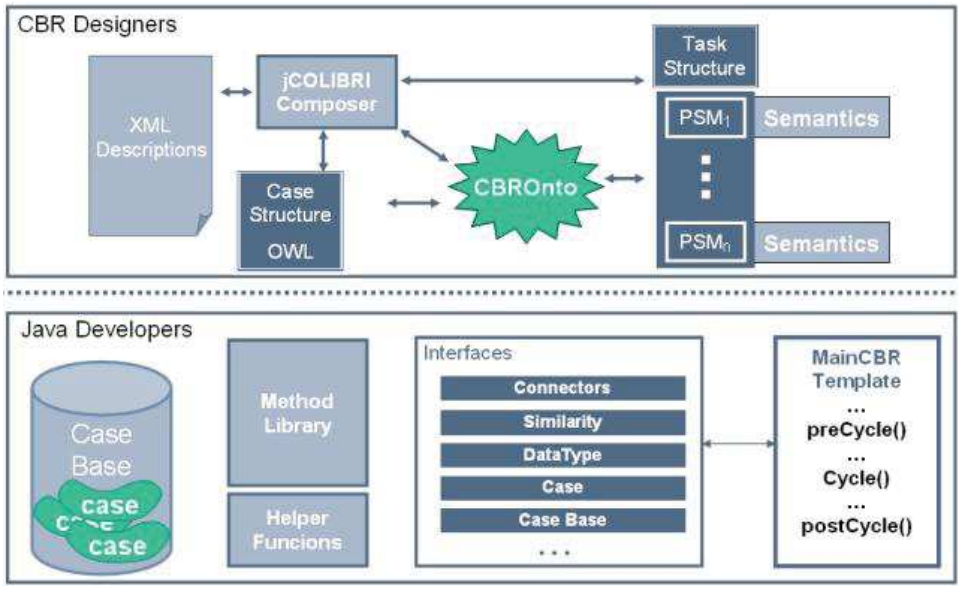


Figure 4: L'architecture en deux couches de jCOLIBRI2 [7].

La couche inférieure contient les composants de base du framework avec des interfaces bien définies et claires. Cette couche ne contient aucun type d'outil graphique pour le développement d'applications de RàPC, c'est simplement une boîte blanche orientée objet qui doit être utilisée par les programmeurs. La couche supérieure contient des descriptions sémantiques des composants et plusieurs outils

d'aides dans le développement d'applications basées sur le RàPC (boîte noire avec constructeur visuel de framework).

La couche inférieure a de nouvelles fonctionnalités qui permettent de résoudre la plupart des problèmes identifiés dans la première version. Elle tire parti des nouvelles possibilités offertes par les nouvelles versions du langage Java. Le changement le plus important est la représentation des cas comme des « Java Beans ».

9.3. L'architecture à deux couches de persistance de jCOLIBRI2 :

Les Composants de cas définis dans la section précédente sont utilisés pour remplir les attributs des objets CBRCase lors du chargement des cas. Avant d'expliquer ce mécanisme, nous allons décrire l'architecture à deux couches de persistance de jCOLIBRI2.

Les systèmes de RàPC doivent accéder aux cas enregistrés d'une manière efficace, un problème qui devient plus pertinent lorsque la taille de la base de cas augmente. jCOLIBRI (les deux versions 1 et 2) divise le problème de la gestion de la base de cas en deux parties [6]. Cette architecture est illustrée dans la **Figure 5**.

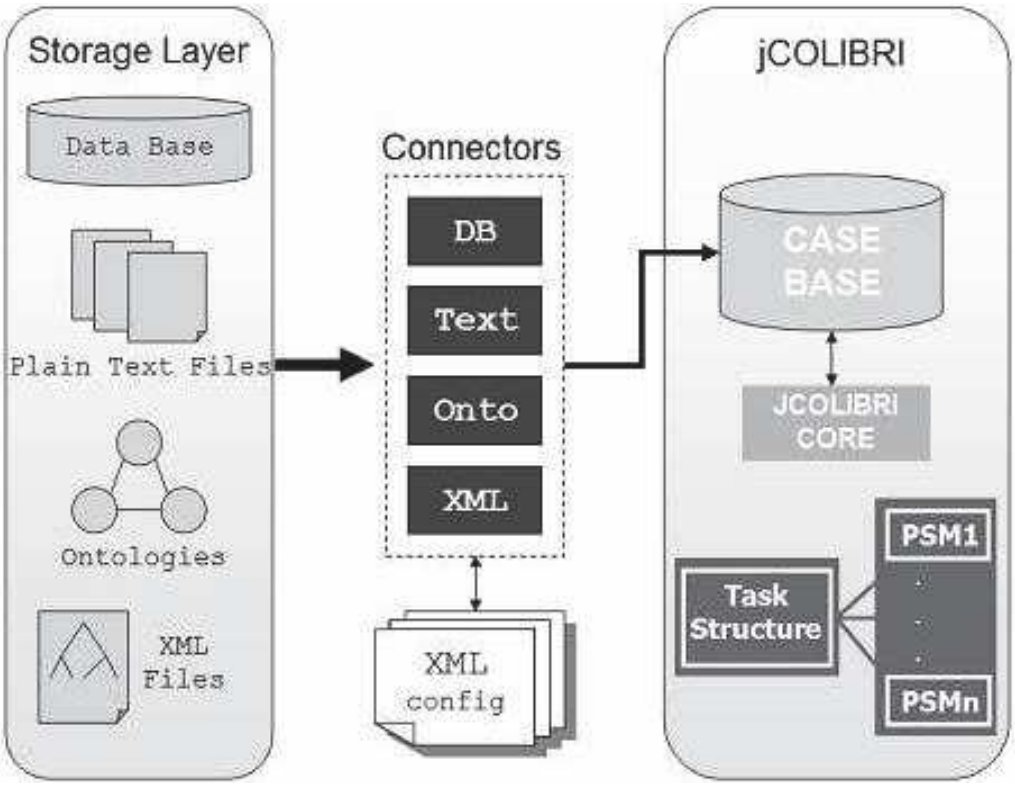


Figure 5 : La gestion de la base de cas de jCOLIBRI2 [6]

9.4. La création d'une application CBR par jCOLIBRI :

La création d'une application de RàPC par jCOLIBRI se déroule en plusieurs étapes :

- La définition du « Case Structure » durant laquelle l'utilisateur explique au système quelles sont les données qu'il utilise pour représenter les cas. Ces données ont un type (par ex. Boolean, String, Integer…) et la structure est sauvegardée sous la forme d'un fichier XML pour les besoins des modules de l'application.

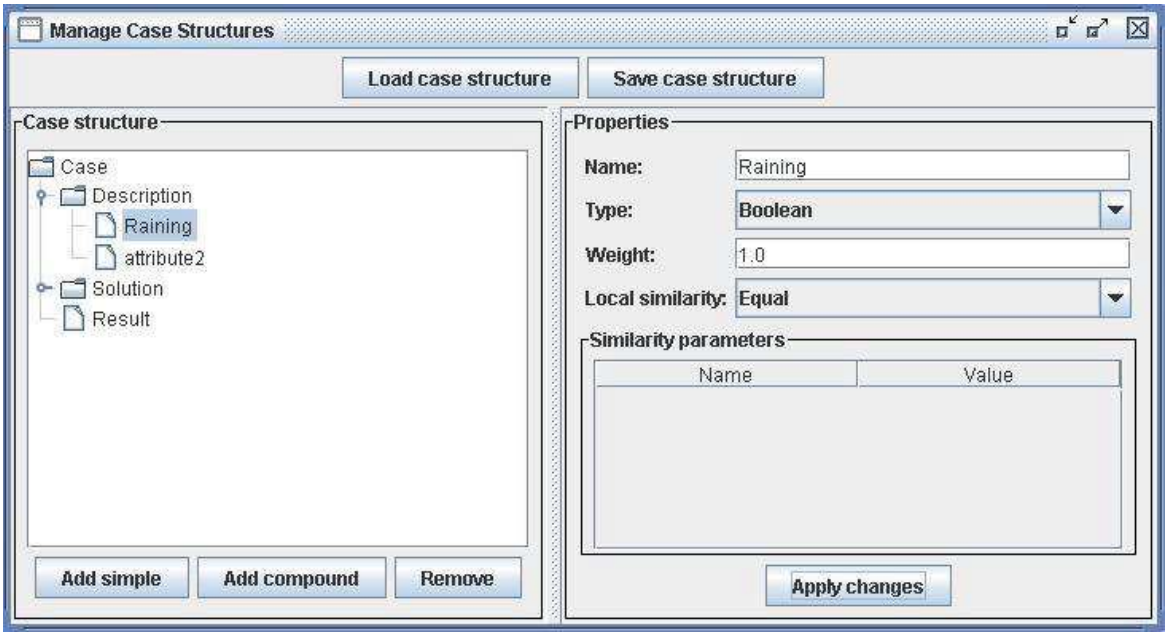


Figure 6 : Création d'une structure de cas

- La définition du « Connector » durant laquelle l'utilisateur explique comment sont stockées les données précédemment décrites. Il s'agit ici de décrire le format de fichier utilisé (fichier de base SQL, simple fichier texte…) afin de permettre aux modules de jCOLIBRI l'interfaçage avec le fichier de données.

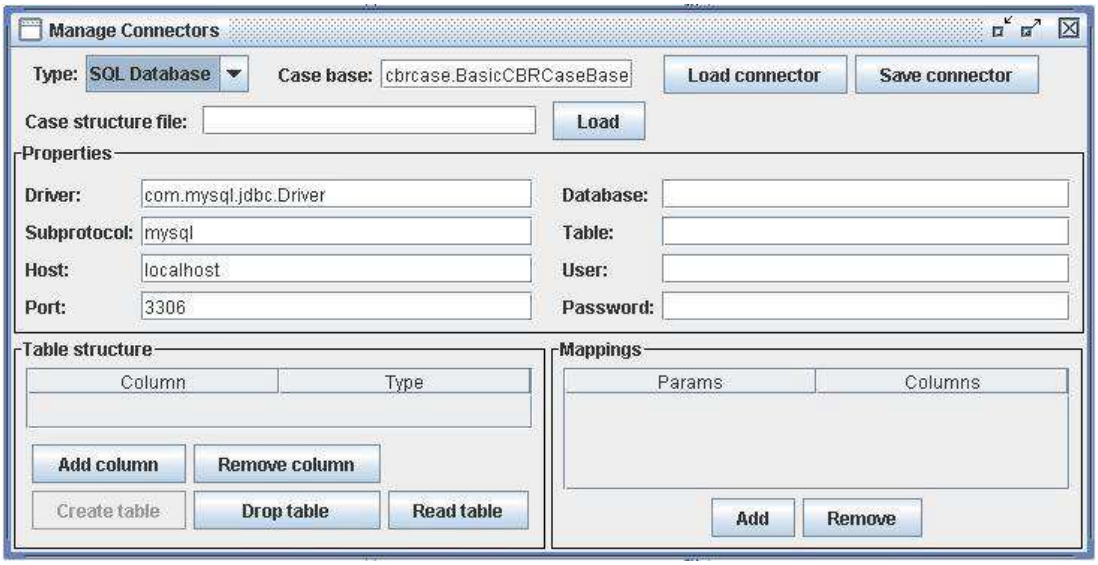


Figure 7 : Création d'un connecteur

- L'instanciation effective des 3 phases de traitement des données : le prétraitement des données (comprenant le chargement des données et leur mise en forme), le traitement des données (c'est le cœur de l'application, appliquant les principes de RàPC) et le post-traitement des données (la sauvegarde sur un médium persistant) .

10. Ontologies dans les applications CBR :

Les ontologies sont utiles pour concevoir des applications gourmandes en connaissances RàPC car elles permettent à l'ingénieur de connaissance d'utiliser les connaissances déjà acquises, conceptualisées et mises en œuvre dans un langage formel, comme la logiques de description (DL : Description Logic), ce qui réduit considérablement le goulot d'étranglement d'acquisition de connaissances. De plus, la réutilisation d'ontologies à partir d'une bibliothèque bénéficie également de leur fiabilité et leur cohérence. Les ontologies sont utilisées pour construire des modèles de connaissance du domaine général. Bien que dans un système de RàPC la principale source de connaissance est l'ensemble des expériences précédentes, l'approche du RàPC est vers les applications intégrées qui combinent les connaissances cas particulier des modèles de connaissance du domaine général. Le plus de connaissances est intégré dans le système, le plus efficace il devrait être. Les processus sémantiques du RàPC peuvent tirer parti de cette connaissance du domaine et obtenir des résultats plus précis [6].

La formalisation des ontologies est utile pour la communauté en ce qui concerne le RàPC à des fins différentes, à savoir:

1. La persistance de cas et / ou index à l'aide des individus ou des concepts qui sont incorporés dans l'ontologie elle-même.
2. Comme vocabulaire pour définir la structure de cas, soit si les cas sont intégrés en tant qu'individus dans l'ontologie elle-même, ou si les cas sont stockés dans un support de persistance différente comme une base de données.
3. Comme terminologie de définir le vocabulaire de la requête. L'utilisateur peut mieux exprimer ses besoins s'il peut utiliser un vocabulaire plus riche pour définir la requête. Lors du calcul de similarité l'ontologie permet de combler le fossé entre la requête et la terminologie utilisée dans la base de cas.
4. La récupération et la similitude, d'adaptation et d'apprentissage.
5. La réutilisation des connaissances entre les différents systèmes de RàPC.

jCOLIBRI2 permet de mettre en œuvre les systèmes de RàPC avec toutes ces fonctionnalités ou, du moins, offre une architecture de gestion d'ontologies qui est la base de ce type d'applications RàPC.

10.1. Cas et le requêtes de vocabulaire :

En ce qui concerne le vocabulaire cas, il s'agit d'une approche directe consistant sur l'utilisation d'une ontologie de domaine d'une manière orientée objet : les concepts sont des types ou des classes, les individus sont autorisés valeurs, ou des objets, et les relations sont les attributs décrivant les objets.

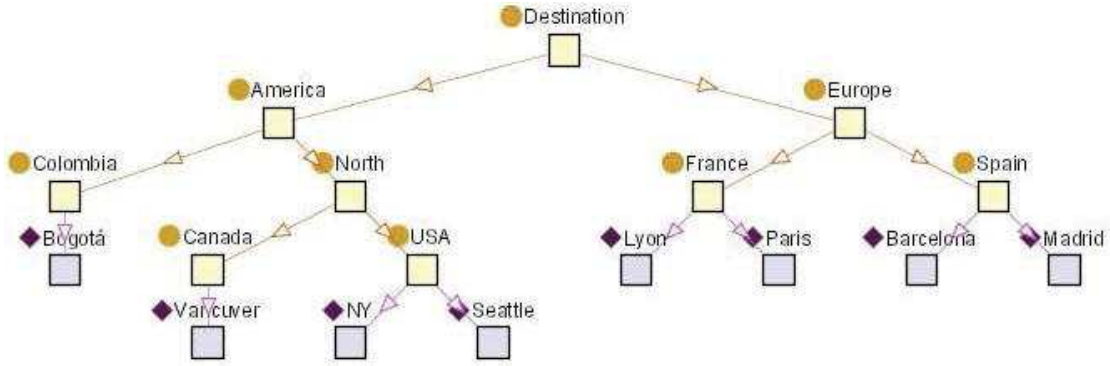
Il ya aussi des types simples comme les cordes ou les numéros qui sont considérés de façon traditionnelle. La structure est définie par cas types de l'ontologie même si les cas ne sont pas stockés en tant qu'individus dans l'ontologie.

10.2. Exemple: Dans le domaine de voyage, laisse supposer que nous avons une base de la jurisprudence existante où il est défini un type énuméré pour l'attribut Région où les valeurs autorisées sont les pays : l'Espagne, la France, l'Italie et les autres. Supposons que $case_i$ est un cas dont la destination est l'Espagne. Nous ne voulons pas de restreindre le vocabulaire requête du même type mais plus larges permettent requêtes, par exemple :

- Requête 1: "Je veux aller à Madrid"
- Requête 2: "Ma destination préférée est l'Europe"

- Requête 3: «Je voudrais voyager en Espagne"

Dans les trois requêtes et l'utilisation de l'ontologie de la **figure 8** nous avons pu trouver $case_i$ comme un candidat approprié.



	fdeep_basic	fdeep	cosine	detail
Sim(Mad,Bcn)	3/4	1	1	5/6
Sim(Mad,Paris)	1/2	1/2	2/3	3/4
Sim(NY,Seattle	1	1	1	7/8
Sim(Seattle, Vanc)	3/4	3/4	3/4	5/6
Sim(Bcn,Bogot)	0	0	$1/\sqrt{12}$	1/2
Sim(Mad,Bogot)	0	0	$1/\sqrt{12}$	1/2
Sim(Vanc,Bogot)	1/2	1/2	1/2	3/4

Figure 8: Exemple d'application des fonctions de similarité [6]

11. Conclusion

Cette étude nous a permis de découvrir un domaine très vaste d'apprentissage qui utilise le RàPC. Nous avons pu comprendre le principe du RàPC ainsi que les raisons pour lesquelles il est adopté comme outil dans de plus en plus de domaines d'applications.

C'était en plus une opportunité pour mettre en œuvre les concepts et fondements du RàPC à travers l'outil boîte blanche « jCOLIBRI ». La disponibilité d'un outil permet de mieux comprendre les principes appliqués dans le RàPC et les différents défis auxquels cette approche fait face.

En effet, beaucoup de problèmes relatifs à la représentation des cas, au choix d'une fonction appropriée de calcul de similarité, à la sélection et à l'adaptation des cas, font

encore l'objet de recherches intensives afin de développer des systèmes RàPC de plus en plus efficaces et performants.

Ceci constituera la suite logique de notre travail. Nous envisageons d'utiliser le framework jCOLIBRI comme boîte blanche sous l'environnement « eclipse ». Ceci nous permettra de comprendre la structure hiérarchique de ces classes ainsi que le principe de chaque traitement. Il deviendra ensuite possible d'apporter une contribution à ce framework.

Chapitre 3

Le processus de recherche de cas

12-Le processus de recherche de cas :

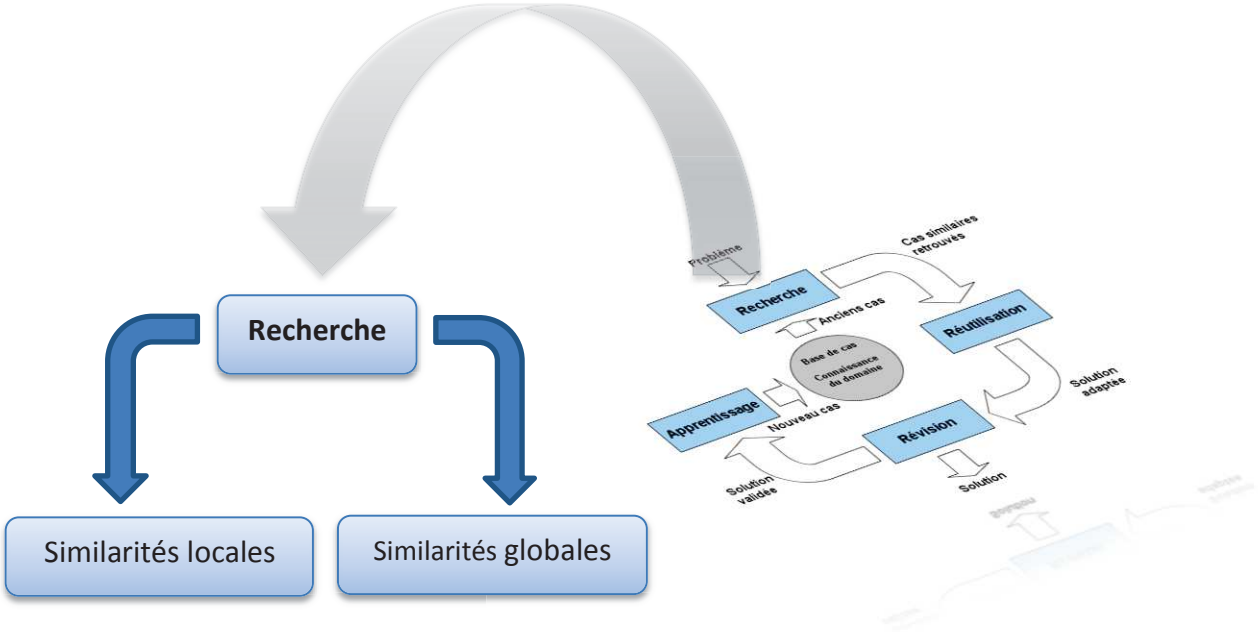


Figure 09: Les deux étapes de la phase de recherche (case retrieving)

12.1- Introduction

Elle consiste à rechercher dans la base de cas le ou les cas sources les plus proches à partir de la description de la partie problème du cas cible, qui vont être utilisés pour le résoudre. Cette phase doit permettre d'obtenir la meilleure solution en effectuant les tâches suivantes : l'identification des caractéristiques pertinentes du problème, la remémoration et la sélection des meilleurs cas parmi les cas sources. L'exécution de ces tâches est dépendante de la représentation de cas, de leur indexation et de leur organisation dans la base de cas. Deux approches sont considérées dans cette phase, celles :

- reposant sur le calcul de la similarité entre le cas source et le cas cible.
- utilisant, en plus de la notion de similarité, la notion de diversité.

Par conséquent, le choix de la distance utilisée pour la mesure de proximité entre deux points de l'espace d'entrée est très important. Le choix d'une métrique dépend donc

de l'application visée et, plus précisément, par la taille de la base de cas disponible, par le codage utilisé, par le degré de recouvrement entre les différentes classes si elles existent dans la base, par la normalisation utilisée ainsi que par le bruit présent dans la base (cas parasites qui détériorent la qualité de prédiction). Ces mesures peuvent être locales ou globales [5].

12.2- Similarités locales :

Elles sont basées sur les caractéristiques du cas. Elles dépendent du type des caractéristiques et des rangs (plage) des valeurs des caractéristiques. Généralement, le calcul des similarités locales dépend du type de descripteur et est basé sur la distance.

- pour les valeurs de descripteurs numériques :

$$sim(a\,,b) = 1 - \frac{|a-b|}{range}$$

- pour les valeurs de descripteurs symboliques (mono-valeurs) :

$$sim(a\,,b) = \begin{cases} 1\ pour\ a = b \\ 0\ pour\ a \neq b \end{cases}$$

- pour les valeurs de descripteurs symboliques (multi-valeurs) :

$$sim(a\,,b) = \frac{card(a) \cap card(b)}{card(a \cup b)} \qquad [5]$$

- pour les valeurs de descripteurs taxonomiques :

$$sim(a\,,b) = \frac{h(commonnode(a,b))}{\mathrm{Min}(h(a),h(b))} \qquad [5]$$

Où :

a et *b*: sont des valeurs des descripteurs.

card : est la cardinalité de l'ensemble

*rang*e : est la valeur absolue de la différence entre la borne supérieure et la borne inférieure de l'ensemble des valeurs

h : est le poids (le nombre de niveaux) de l'arbre taxonomique

12.3- Similarités globales :

Elles sont calculées au niveau des cas ou des objets en agrégeant les similarités locales. Plusieurs similarités globales sont utilisées dans les systèmes de ràpc et aucune d'elles n'est universelles et restent dépendantes du domaine concerné. [5] On peut citer :

- Weighted Block-City

$$sim(A,B) = \sum_{i=1}^{n} w_i sim_i(a_i, b_i)$$

- la distance de Minkowski . Si r = 2 alors on retrouve la distance Euclidienne et si r = 1 alors il s'agira de la distance de Manhattan .

$$sim(A,B) = \left[\frac{1}{2}\sum_{i=1}^{n} sim_i(a_i, b_i)^r\right]^{\frac{1}{r}}$$

- Maximum based

$$sim(A,B) = max_i w_i sim_i(a_i, b_i)$$

Où :

n est le nombre d'attributs,

w_i est le poids (évalué en fonction de l'importance) de l'attribut i,

sim_i est la similarité locale calculée pour l'attribut i.

Il existe d'autres types de mesures de similarité qui tiennent compte des historiques, des séquences dans les cas, du temps, de l'espace, des structures complexes, des plans, des séries, etc.

Une fois la mesure de similarité établie pour un système donné, la remémoration des cas dans la base de cas se fait suivant un algorithme de recherche. Plusieurs types d'algorithmes peuvent être alors appliqués. L'algorithme des K Plus Proches Voisins « KPPV » (plus connus en Anglais sous le nom K-Nearest Neighbors (K-NN) est la méthode la plus habituellement utilisée. La méthode utilise donc deux paramètres : le

nombre K et la fonction de similarité pour comparer le nouveau cas aux cas déjà classés. On peut également citer l'algorithme basé sur les approches inductives (KD-arbres, ID3, C4.5, etc.), les algorithmes appliqués à l'historique des séquences, l'algorithme de chemin de similarité, l'induction basée sur la connaissance, la recherche basée sur la structure (« template retrieval ») qui est similaire aux requêtes SQL, la recherche Case Retrieval Nets et l'algorithme basé sur l'exploitation des différentes vues sur les cas – Fish & Shrink – qui combine dynamiquement différentes mesures de similarité pendant la recherche .[5]

12.3.1- Similarité dimensionnelle :

La fonction de similarité dimensionnelle $\mu_a(v1, v2)$ d´dépend du type de l'attribut a et du type de degré de similarité voulu. Traditionnellement, ces degrés se présentent sous la forme de distances, c'est-`a-dire un nombre positif entre 0 et ∞. Plus la distance est petite, plus v1et v2 sont similaires. Cela pose des problèmes de comparaison si les distances ne sont pas normalisées et si |A | varie d'un cas à l'autre pour une requête R. Il est possible de normaliser une distance en la transformant en mesure de similarité

$$\mu_a(v1, v2) = \frac{1}{1 + d\,(v1, v2)}$$

Si $0 \leq d\ (v1, v2) \leq M$, il est possible de normaliser la distance et la transformer en mesure de similarité :

$$\mu_a(v1, v2) = 1 - \frac{d\,(v1, v2)}{M}$$

Un degré de similarité de 0 indique que v1 et v2 sont totalement différents tandis qu'un degré de 1 montre une similarité parfaite. Outre le degré de similarité, il peut parfois être utile de d´définir un ordre sur les valeurs des attributs. Nous traiterons ici les distances dimensionnelles quantitatives et qualitatives les plus courantes, mais d'autres mesures peuvent exister. [2]

12.3.2- Plus proches voisins :

La méthode des plus proches voisins compare la requête *R* avec un sous-ensemble de cas de La base de cas afin d'associer un score à chacun d'eux. Ce score est utilisé pour classer les cas par ordre de similarité afin de retourner le ou les cas les plus similaires. L'algorithme général de recherche se passe en plusieurs étapes. [2]

1. Un préfiltrage de la base de cas fournit un ensemble *S* de cas potentiellement similaires à *R*. L'ensemble résultat doit être relativement petit tout en contenant le cas le plus similaire.

2. Idéalement en parallèle, pour chaque cas $C \in S$,

(a) Les attributs communs à *R* et *C* sont d´exterminés. Soit *A* l'ensemble de ces attributs.

(b) Pour chaque attribut $a \in A$, une fonction de similarité dimensionnelle m*a*(*Ra*,*Ca*) est calculée, où *Ra* représente la valeur de l'attribut *a* dans *R* et *Ca*, la valeur dans *C*.

(c) Un degré de similarité global *s*(*R*,*C*) est d´déterminé en pondérant les degrés de similarités dimensionnels. Souvent,

$$s(R,C) = \frac{\sum_{a \in A} w_a(R,C)\mu_a(R_a,C_a)}{\sum_{a \in A} w_a(R,C)} \qquad [2]$$

3. Classer les degrés de similarité et retourner le ou les cas les plus similaires à *R*.

12.3.3- Sélection finale :

Parfois, on est obligé de sélectionner un seul cas parmi les cas les plus proches déjà sélectionnés, cela dépend du domaine, par exemple dans le cas d'un diagnostic d'un patient, le médecin doit préciser le cas choisit et le valider par la suite. Par contre dans l'application que nous utiliserons c'est l'utilisateur qui a la possibilité de choisir la solution qui convient selon le critère souhaité (prix du voyage, l'emplacement de l'hôtel…).

13- Calcul de similarité dans jCOLIBRI

13.1- Les Fonctions de similarité :

Il existe de nombreuses fonctions de similarité dans le package « *jcolibri.method.retrieve.nnretrieval.similarity* ». Chaque mesure de similarité locale ne peut être appliquée qu'à certains types de données. De cette façon, il ya des similitudes locales applicables aux entiers, des chaînes, des instances, ...

Les développeurs peuvent définir leurs propres mesures de similarité mettant en œuvre les interfaces:

Jcolibri.method.retrieve.nnretrieval.similarity.globalsimilarityfunction et jcolibri.method.retrieve.nnretrieval.similarity.localsimilarityfunction. La signature des deux interfaces sont [7]:

13.2- l'interface *localsimilarityfunction :*

Dans l'interface " *localsimilarityfunction* " il y a la fonction "*compute* " qui calcule la similarité local entre deux objets « attributs ».

Les paramètres de cette fonction est :

Caseobject : objet de cas.

Queryobject: objet de requête.

Enfin il return une valeur entre [0..1].

```
package jcolibri.method.retrieve.NNretrieval.similarity;


public interface LocalSimilarityFunction {
      public double compute(Object caseObject, Object queryObject)
throws jcolibri.exception.NoApplicableSimilarityFunctionException;

}
```

Figure 10 : l'interface *localsimilarityfunction*

13.3- l'interface *globalsimilarityfunction :*

Dans l'interface " *globalsimilarityfunction* " il y a la fonction "*compute* " qui calcule la similarité global entre deux cas.

Les paramètres de cette fonction est :

Componentofcase : attribut composé du cas.

Componentofquery : attribut composé de la requête.

_case : cas étant comparée.

_query : requête étant comparée.

Numsimconfig : Configuration des fonctions de similarité.

Enfin il return une valeur entre [0..1].

```
package jcolibri.method.retrieve.NNretrieval.similarity;

public interface GlobalSimilarityFunction {

      public double compute(CaseComponent componentOfCase,
CaseComponent componentOfQuery, CBRCase _case, CBRQuery _query,
NNConfig numSimConfig);
}
```

Figure 11 : l'interface *globalsimilarityfunction*

Les fonctions de similarité locales ne peuvent pas accéder aux autres attributs de l'affaire pour calculer leurs mesures. Cela peut être un problème dans certaines applications où la similitude des deux attributs dépend de la valeur des autres attributs dans le même composant de cas. Pour résoudre cet inconvénient, jcolibri2 comprend la classe abstraite

jcolibri.method.retrieve.nnretrieval.similarity.incontextlocalsimilarityfunction qui intègre des informations sur le contexte (casecomponent) de l'attribut.

13.4- Sélection des cas :

La plupart des cas similaires doivent être sélectionnés une fois qu'ils ont été marqués en fonction de leur similarité avec la requête. Habituellement, seuls les meilleurs k la plupart des cas similaires sont sélectionnés. Ce processus de récupération qui combine

la notation de voisin le plus proche et la haute sélection de k est communément appelé k-NN récupération.

La classe *jcolibri.method.retrieve.selection.selectcases* inclut des méthodes de base pour sélectionner les cas. [7]

13.5- Les ontologies dans jcolibri2 :

Jcolibri2 permet d'utiliser des ontologies comme support de persistance des cas. L'architecture des connecteurs du cadre facilite l'intégration de cette fonctionnalité comme la seule exigence est la mise en œuvre d'un nouveau connecteur. Ce connecteur est jcolibri.connector.ontologyconnector. [7]

13.6- Exemple de fonction de similarité locale(Equal) :

```
package jcolibri.method.retrieve.NNretrieval.similarity.local;

import
jcolibri.method.retrieve.NNretrieval.similarity.LocalSimilarityFunction;


/**
 * This function returns 1 if both individuals are equal, otherwise
returns 0
 */
public class Equal implements LocalSimilarityFunction {

      /**
       * Applies the similarity function.
       * @param o1 Object.
       * @param o2 Object.
       * @return the result of apply the similarity function.
       */
    public double compute(Object o1, Object o2) throws
jcolibri.exception.NoApplicableSimilarityFunctionException{
        if ((o1 == null) || (o2 == null))
            return 0;
        return o1.equals(o2) ? 1 : 0;
    }

    /** Applicable to any class */
      public boolean isApplicable(Object o1, Object o2)
      {
            return true;
      }
}
```

Figure 12[7]: Exemple de fonction de similarité locale

Chapitre 4

Conception et implémentation

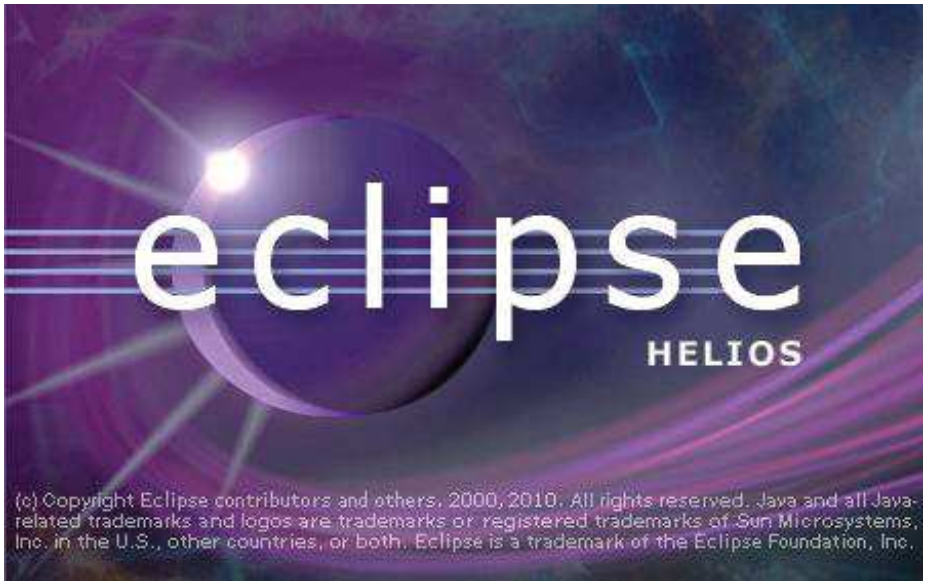
14- Conception et implémentation :

Dans le cadre de notre projet de fin d'études nous avons essayé de comprendre les classe de la plateforme Jcolibri, et intégrer notre exemple de voyage **The Travel Recommender** dans cette plateforme ensuite comprendre les classe de cette exemple et la relation avec les classe abstract de la plateforme Jcolibri puis ajouté notre fonction de similarité.

Ce chapitre présente d'abord les outils utilisés pour notre travaille.

14.1- Eclipse :

Dans la programmation informatique, Eclipse est un environnement de développement intégré multi-langue (IDE) comprenant un espace de travail de base et une extensible plug-in système de personnalisation de l'environnement. Elle est écrite en Java. Il peut être utilisé pour développer des applications en Java et, d'autres langages de programmation. Il peut également être utilisé pour développer des packages pour le logiciel mathématique.



14.2- Importation de jCOLIBRI2 dans Eclipse :

Pour développer une nouvelle application CBR avec jCOLIBRI2 nous recommandons l'IDE Eclipse (www.eclipse.org). jCOLIBRI2 inclut les fichiers nécessaires pour importer le projet de cadre dans Eclipse:. classpath et. project. Mais si vous importez l'ensemble du projet, vous aurez les fichiers source du cadre dans votre projet Eclipse. La solution intelligente consiste à charger uniquement le fichier jcolibri2.jar (et bibliothèques associées) dans votre façon de projet. De cette façon, vous n'aurez que vos fichiers source dans le projet Eclipse.

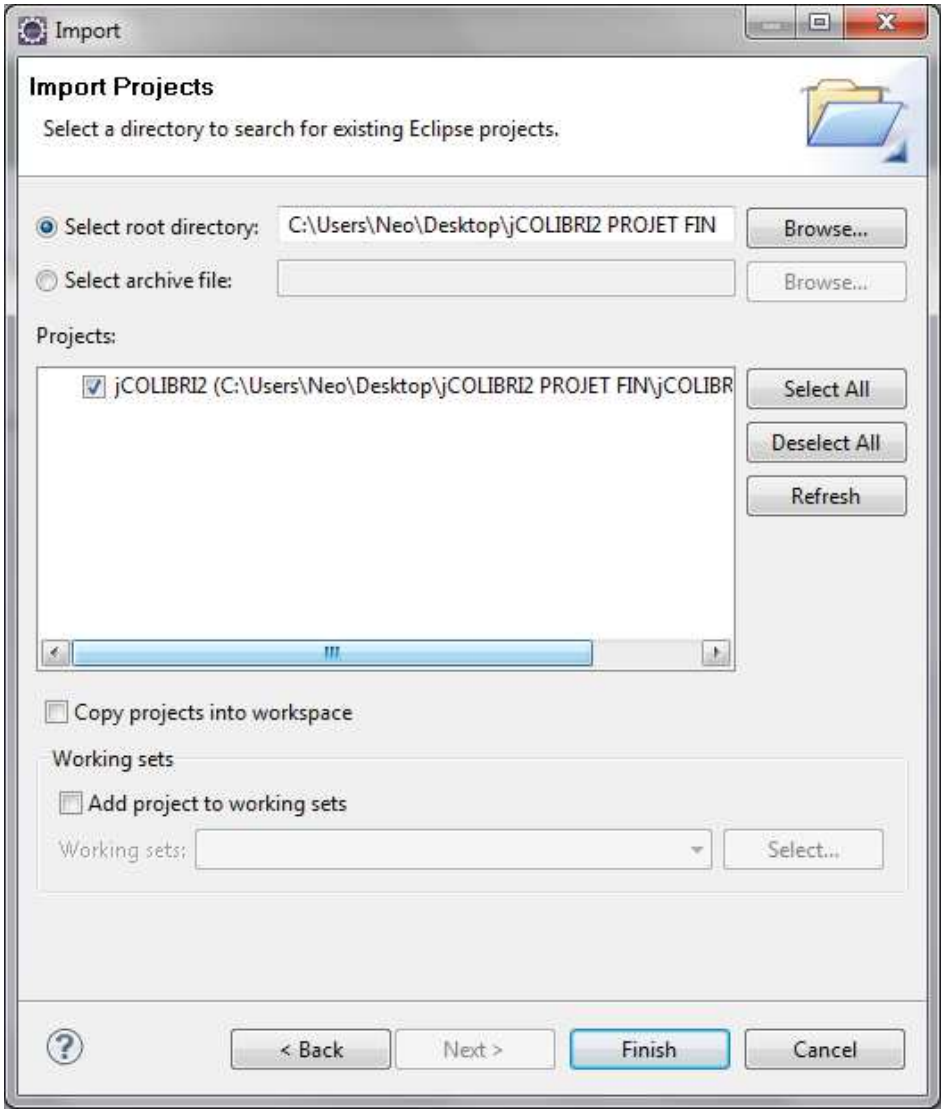


Figure 13 : Importation de jCOLIBRI2 dans Eclipse

Nous allons charger l'ensemble du projet jCOLIBRI dans Eclipse pour permettre une navigation plus facile à travers le code source du cadre. Pour éviter des modifications des fichiers sources de votre installation, nous allons faire une copie du projet dans l'espace de travail Eclipse lors de l'importation du projet Pour importer jCOLIBRI2 dans Eclipse utiliser le menu File - Import. Ensuite, choisissez l'onglet "General - Existing projects into workspace" option et enfin sélectionner le dossier d'installation du cadre dans le "Select root directory". Assurez-vous que le "Copy projects into workspace" est cochée et cliquez sur "Finish".

Une fois que le projet est importé, vous pouvez naviguer dans son contenu et les fichiers sources en utilisant le « Package Explorer »

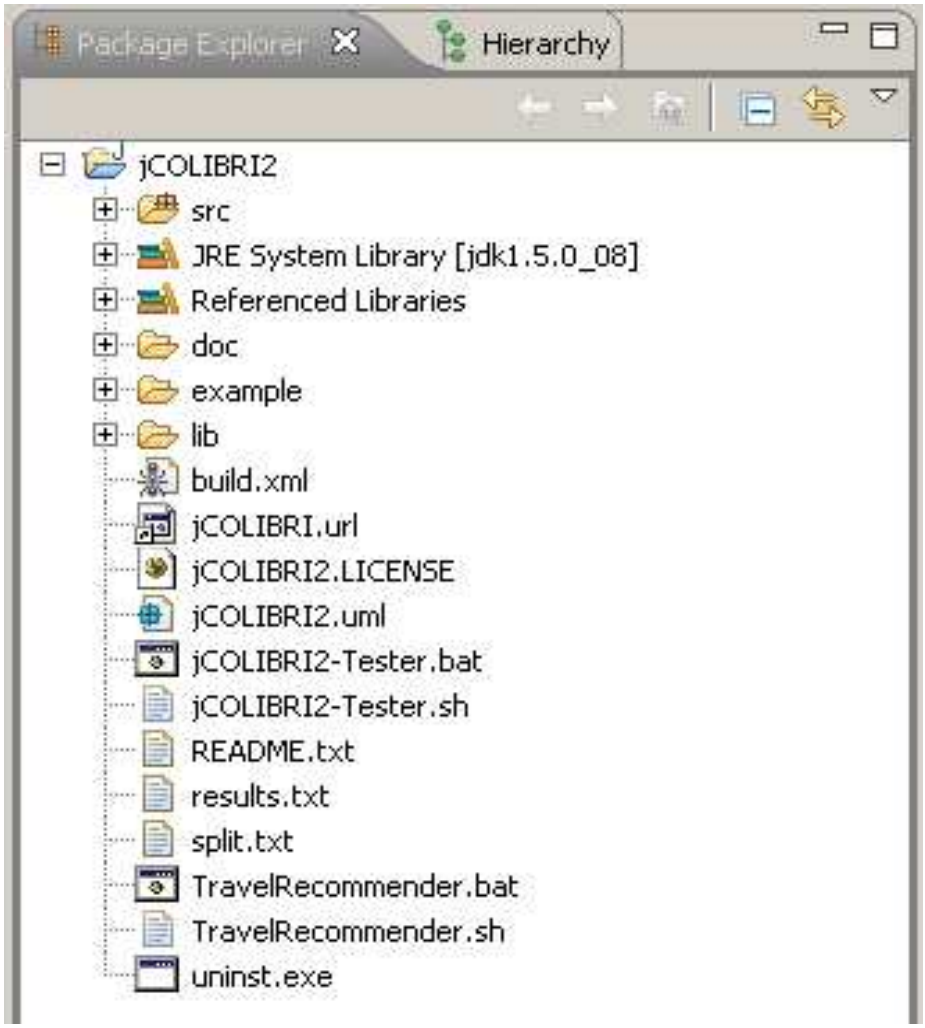


Figure 14 : Explorateur de paquet

14.3- Préparation des fichiers de Travel Recommender

Le code source complet de l'application de recommandation de voyage « Travel Recommender » se trouve dans le "exemple" sous-dossier du cadre et est compressé comme travelrecommender-Source.zip. vous devez décompresser le contenu du fichier zip dans le répertoire "src" de votre projet jCOLIBRI2 Eclipse (pas le dossier d'installation d'origine). Ce projet sera situé dans le répertoire de l'espace de travail Eclipse. Assurez-vous que vous conservez les noms de dossiers lorsque unziping le fichier. Une fois cela fait, revenez à Eclipse, sélectionnez le projet jCOLIBRI2 dans le "Package Explorer", et rafraîchir à l'aide de l'option de menu contextuel ou la touche F5. Ensuite, une nouvelle sous-package nommé jcolibri.examples.TravelRecommender apparaîtra dans le dossier source.

14.4- Travel Recommender :

Travel Recommender est un exemple d'application bien connue dans la communauté CBR. Par conséquent, je nous ai choisi cette application pour montrer comment mettre en œuvre des applications CBR à l'aide jCOLIBRI2.

Dans cette application, la base de cas est composé de plusieurs voyages. Le système reçoit un trajet désiré sous forme d'une interrogation, compare la requête avec les trajets dans la base de cas à l'aide d'une fonction de similarité, et renvoie les plus semblables. Après la récupération, ces cas les plus similaires peuvent être adaptés en fonction des restrictions de la requête.

Chaque cas est représenté par plusieurs attributs: id, type de vacances, nombre de personnes, la région, le transport, la durée, la saison, le type de logement, le prix et l'hôtel. Dans notre application, le prix et l'hôtel seront considérés comme la solution de l'affaire en attendant les attributs restants seront la description du cas. De cette façon, nos cas sont divisés en: une description utilisée pour récupérer des affaires similaires étant donné une requête, et une solution adaptée en fonction des valeurs de la requête. Regardons les différentes étapes de notre application CBR.

14.4.1- Définir la requête :

Dans cette étape, l'utilisateur définit la requête au système. Elle doit définir quelles sont les valeurs des différents attributs d'un voyage: la durée, la saison, le transport, etc.

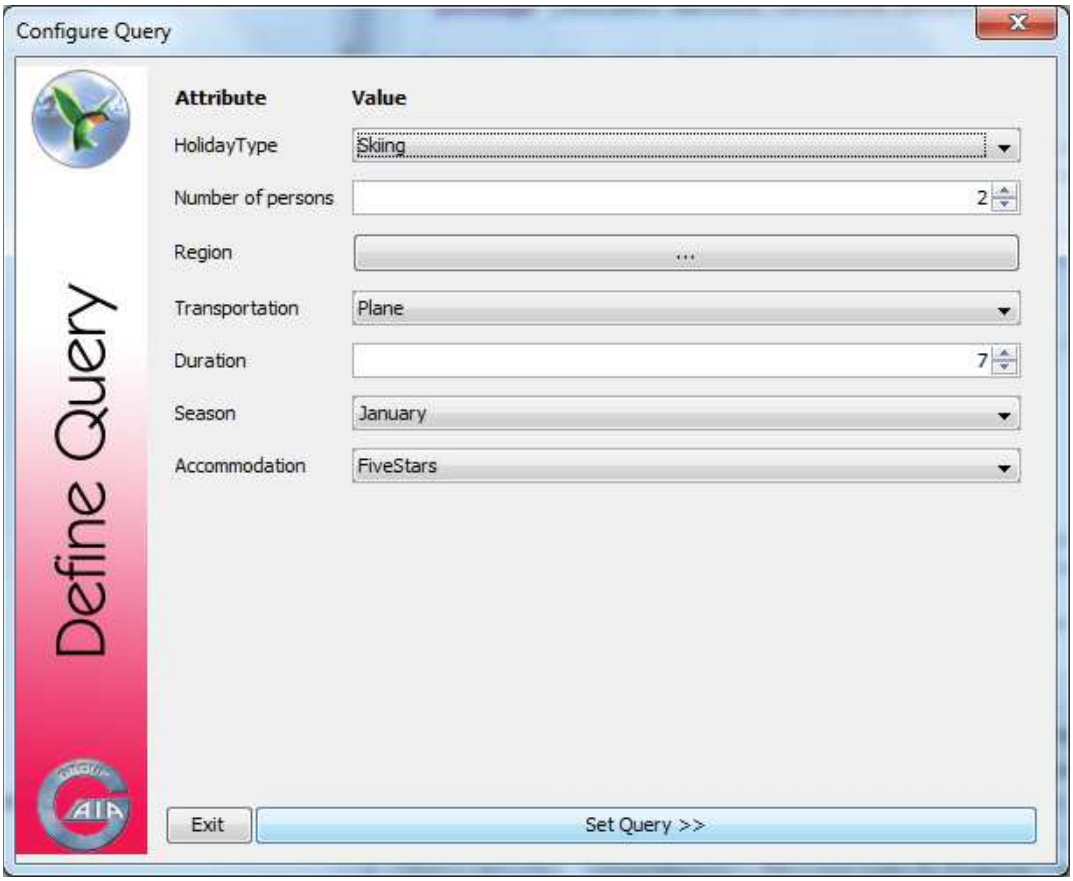


Figure 15 : Requête de configuration

14.4.2- Configurer similarité :

Ici, l'utilisateur configure la mesure de similarité utilisée pour récupérer les cas les plus similaires à la requête. jCOLIBRI2 met en oeuvre plusieurs fonctions de similitude qui peuvent être utilisées selon le type de l'attribut (entiers, des chaînes, etc.) En outre, nous pouvons définir nos propres mesures de similarité.

Vous pouvez également attribuer un poids à chaque attribut de la requête qui sera prise en compte lors du calcul de la moyenne de tous les attributs. En outre, certaines fonctions de similarité peuvent avoir des paramètres utilisés pour configurer la mesure de similarité.

Enfin, la valeur k indique le nombre de cas doivent être récupérées. Nous utilisons un algorithme K le plus proche voisin (k-NN) qui calcule la similitude de la requête avec tous les cas et ordonne alors le résultat en fonction de cette valeur de similarité. Puis les premiers k la plupart des cas similaires sont retournés.

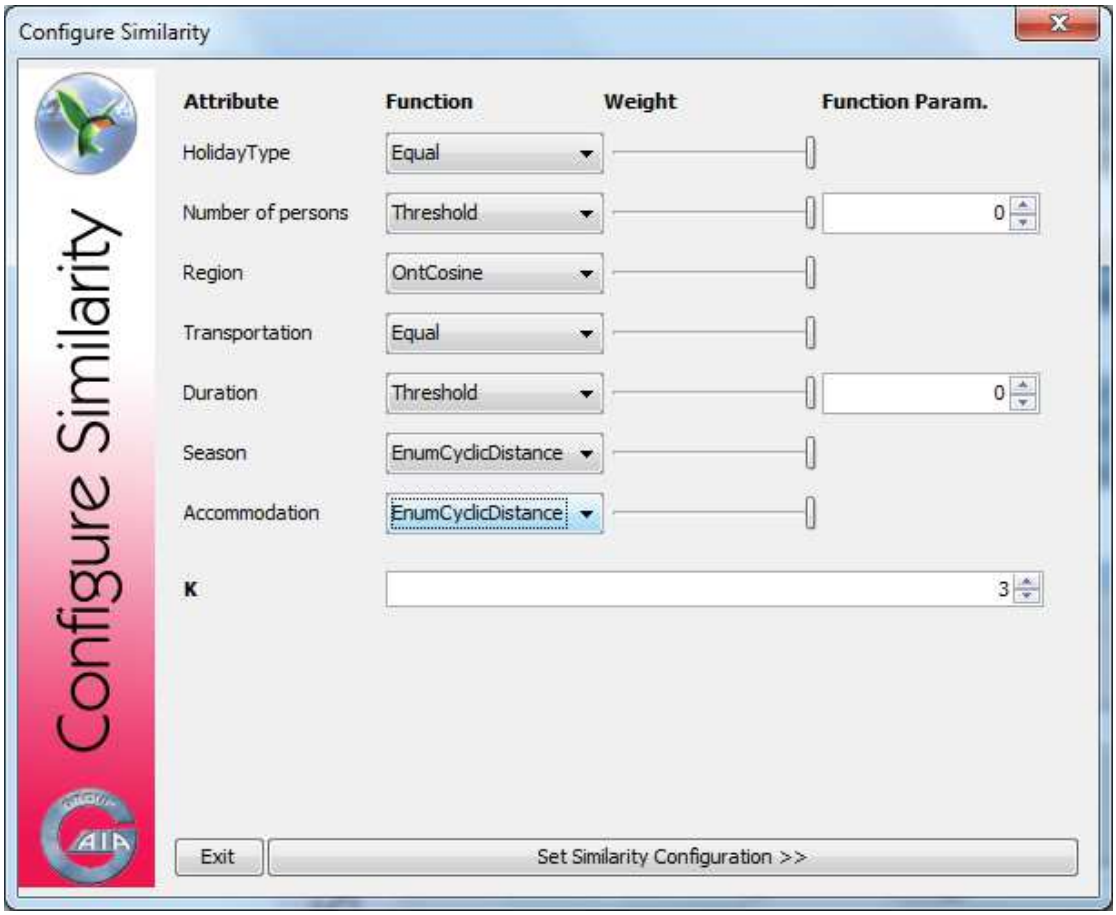


Figure 16: **Configuration du similarité**

14.4.3- Les cas sélections :

Cette étape montre les documents trouvés par la méthode k-NN. Il montre chaque cas et sa similitude avec la requête.

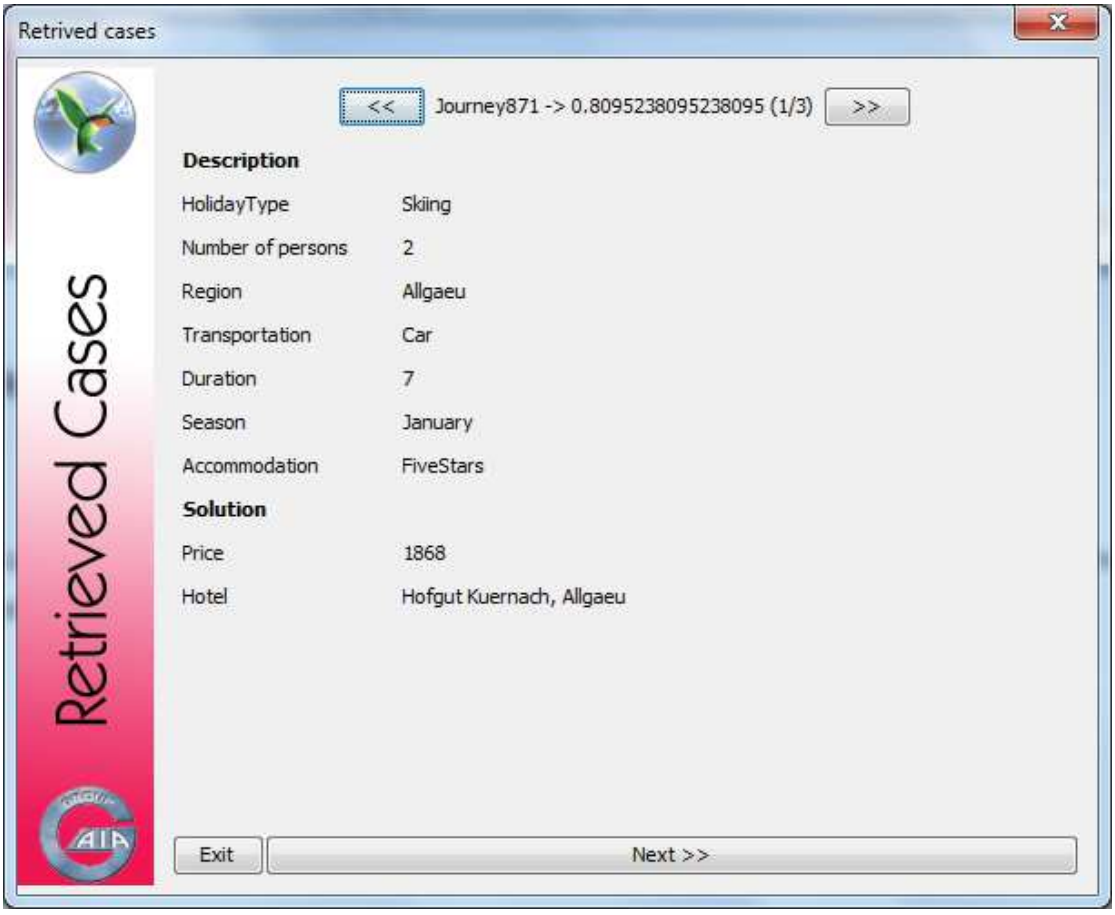


Figure 17 : Les cas sélections

14.4.4- Adaptation :

Dans cette étape, le système adapte les cas extraits aux exigences de l'utilisateur en fonction des valeurs définies dans la requête. Cette utilisation de l'étape d'être très dépendant domaine et sera différent dans d'autres systèmes CBR. Dans notre demande de référent de voyage, nous allons adapter le prix des trajets en fonction du nombre de personnes et la durée définie dans la requête.
Notre système utilise des proportions simples et directes pour effectuer l'adaptation. Par exemple, imaginez que l'affaire a récupéré une durée de 7 jours et coûte 1000. Si l'utilisateur est à la recherche d'un voyage de 14 jours nous devons adapter le prix en

utilisant une proportion directe: si un 7 jours frais de voyage 1000, puis un voyage de 14 jours coûte 2000. Ce processus est également répétée pour le nombre de personnes d'une manière analogue.

Après l'adaptation de la solution des cas, leur description peut être remplacée par la description de la requête. À ce stade, le système va gérer une liste de cas de travail qui sont différentes des cas dans la base de cas. Ces cas de travail représentent des solutions possibles au problème décrit dans la requête.

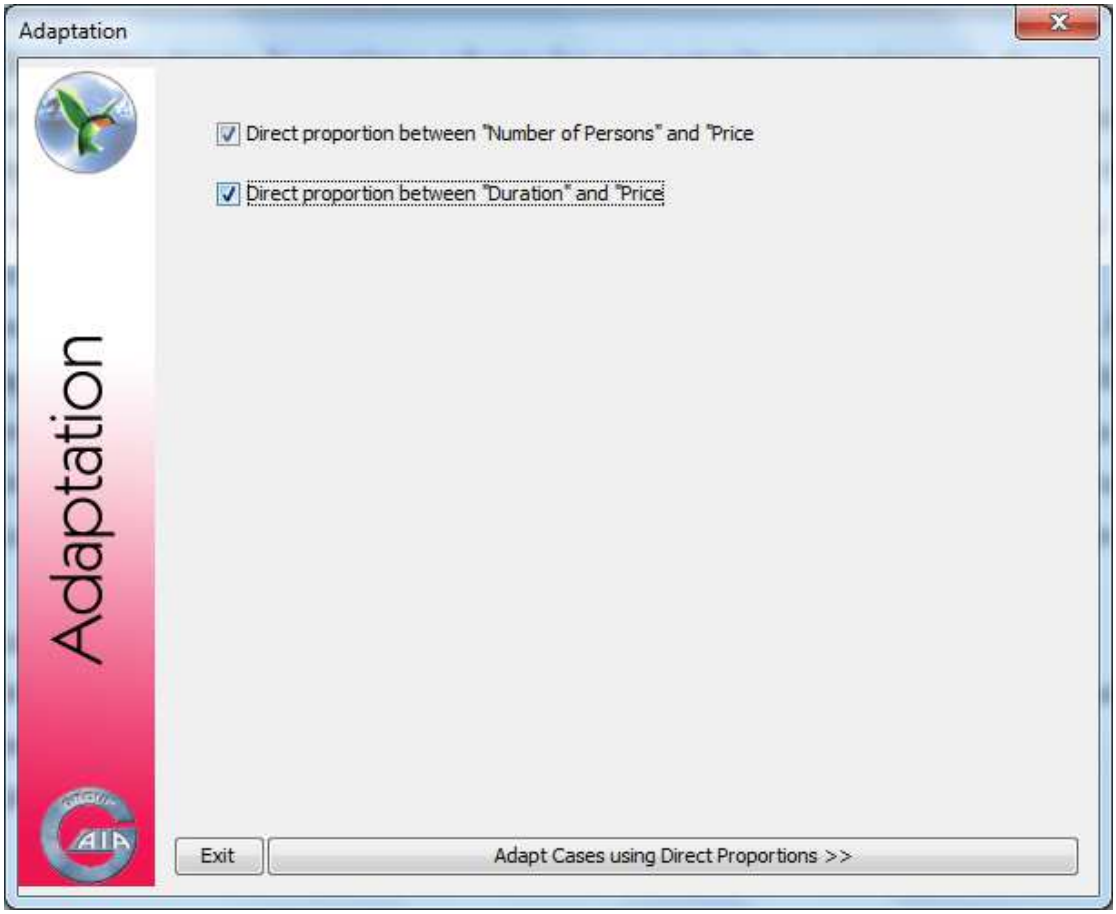


Figure 18 : **Adaptation**

14.4.5- Cas de révision :

Une fois les cas ont été adaptés, l'utilisateur (ou un expert du domaine, dans ce cas, l'agent de voyage) serait d'ajuster les valeurs des cas de travail de manière manuelle. Par exemple, imaginez que l'hôtel n'a pas chambres disponibles et les clients doivent se rendre dans un autre semblable. Une autre situation est que les cas récupérées ne sont pas assez similaires aux exigences de la requête et de l'agent de voyage doit définir manuellement la solution de l'affaire.

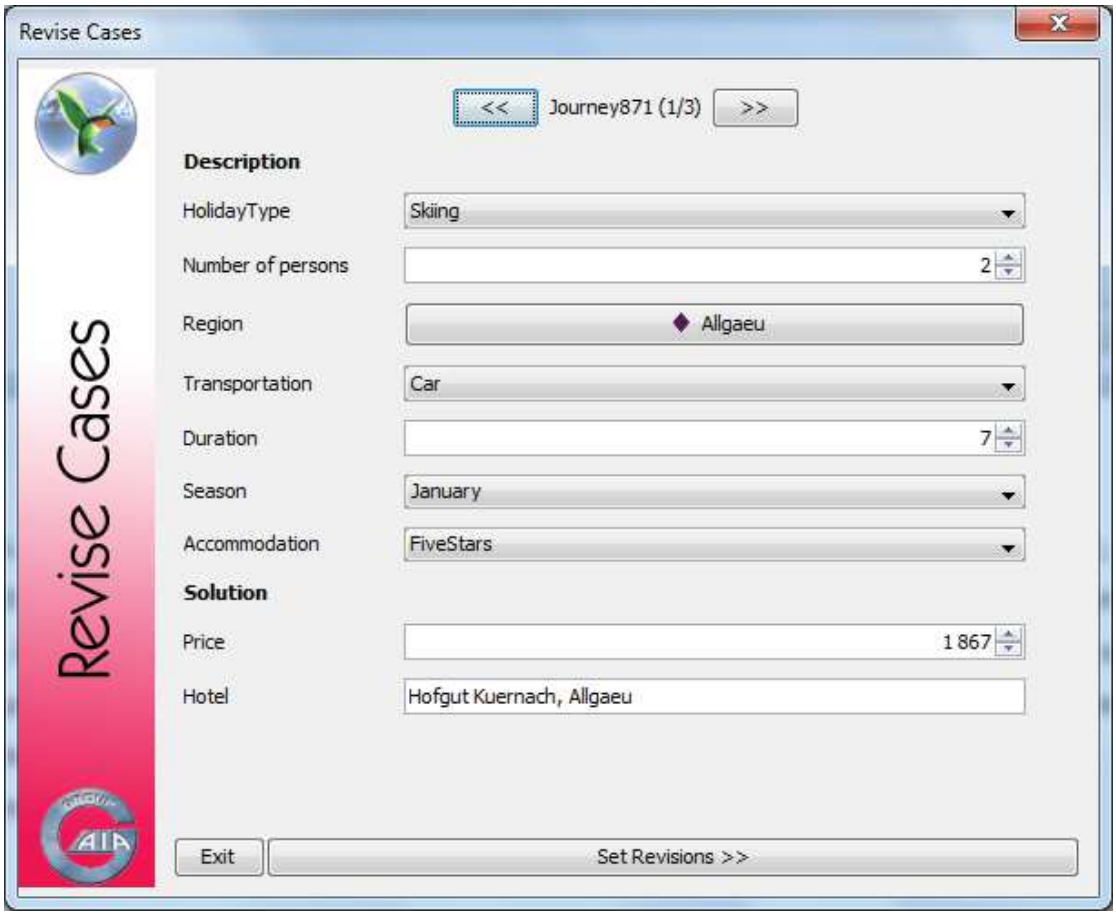


Figure 19 : Cas de révision (1)

Enfin, l'agent de voyage permettrait de sauver le nouveau cas de voyage dans la base de cas pour être utilisée dans des requêtes ultérieures. Si cela est fait, un nouvel ID doit être affecté à ce voyage.

Figure 20 : Cas de révision (2)

14.4.6- La base de cas de Travel Recommender

L'application Conseils sur les voyages utilise la base de données définie dans le fichier travel.sql situé dans le package jcolibri.example.TravelRecommender. Vous pouvez utiliser ce fichier pour générer la base de données dans votre DBM préféré. Ce fichier génère une base de données nommée voyage qui contient un tableau aussi appelé voyage. Le tableau contient 1 024 trajets et est définie par le schéma suivant:

```
c r e a t e t a b l e t r a v e l (
        c a s e I d VARCHAR( 1 5 ) ,
        Hol idayType VARCHAR( 2 0 ) ,
        P r i c e INTEGER,
        NumberOfPersons INTEGER,
        Region VARCHAR( 3 0 ) ,
        T r a n s p o r t a t i o n VARCHAR( 3 0 ) ,
        Du r a t i o n INTEGER,
        Season VARCHAR( 3 0 ) ,
        Accommodation VARCHAR( 3 0 ) ,
        Ho t e l VARCHAR( 5 0 ) ) ;
```

Pour simplifier l'utilisation des exemples, jCOLIBRI2 comprend la base HSQLDB Data Manager (www.hsqldb.org). Cette DBM est complètement implémenté en Java et peut être facilement inclus dans aucun autre projet Java. De cette façon, HSQLDB est utilisé par les exemples du cadre et par l'application de Conseils sur les voyages. Pour lancer la DBM vous pouvez utiliser la classe jcolibri.test.database.HSQLDBserver. Sa méthode init () initialise la DBM chargement des bases de données utilisées par les exemples. Cette initialisation comprend également les tables utilisées par notre référent de voyage définie dans travel.sql

Conclusion générale

Le but de ce projet était d'introduire quelques nouvelles fonctionnalités à la plateforme de RàPC jCOLIBRI. Cet objectif passait évidemment d'abord par la connaissance du domaine du RàPC ainsi que toutes les techniques qui y sont utilisées depuis la phase de recherche de cas, en passant par la sélection, et jusqu'à la phase d'adaptation d'une solution à un nouveau problème.

La tâche a été très dure puisqu'il s'agissait d'un domaine que nous avons découvert à l'occasion de ce projet et dans lequel beaucoup d'abstractions et d'algorithmes sont utilisés. C'est aussi un domaine particulier de l'intelligence artificielle dans lequel on ne peut jamais prétendre développer une solution générale. A chaque fois il s'agit d'une solution s'appliquant à un domaine particulier.

Au bout de ce projet, nous avons d'abord pu comprendre le principe de fonctionnement de la plateforme jCOLIBRI, et surtout son architecture car sans cela il est impossible d'introduire de nouvelles fonctionnalités. Nous nous sommes ensuite penchés sur les différentes fonctions de calcul de similarités qui sont très nombreuses. Cette diversité est due l'hétérogénéité qu'on peut rencontrer lors de la description des cas dans un domaine particulier. Nous avons pu intégrer quelques nouvelles fonctions de calcul de similarité en plus de celles existantes déjà dans la plateforme.

Ce projet était une occasion immense d'apprentissage et de découverte aussi bien sur le plan des lectures que sur le plan pratique. Nous avons pu découvrir et maitriser un domaine tout à fait nouveau pour nous. L'utilisation d'une plateforme existante était une opportunité d'appliquer notre savoir et d'approfondir nos connaissances.

Bibliographie

[1] Ivana Rasovska. Ecole Doctorale Sciences Physiques pour l'Ingénieur et Microtechniques Thèse : Contribution à une méthodologie de capitalisation des connaissances basée sur le raisonnement à partir de cas : Application au diagnostic dans une plateforme d'e-maintenance. Accessible à partit de l'url : http://hal.archives-ouvertes.fr/docs/00/25/78/93/PDF/These_Ivana_finale.pdf.

[2] Eric Buist. : Les éléments fondamentaux du raisonnement à base de cas. Accessible à partir de l'url : http://www.ericbuist.com/me/travaux/cbr.pdf.

[3] Raisonnement par cas : http://fr.wikipedia.org/wiki/Raisonnement_par_cas.

[4] *Maïté DAMME, Romain TAVENARD, Stéphane VENZIN document :* jCOLIBRI : Un atelier générique pour le raisonnement à partir de cas ? Accessible à partir de : http://liris.cnrs.fr/alain.mille/enseignements/Ecole_Centrale/projets_2006/jCOLIBRI.doc.

[5] Mohamed Karim Haouchine. Ecole Doctorale Sciences Physiques pour l'Ingénieur et MicrotechniquesThèse : Remémoration guidée par l'adaptation et maintenance des systèmes de diagnostic industriel par l'approche du raisonnement à partir de cas. Accessible à partir de l'url : http://hal.archivesouvertes.fr/docs/00/46/65/60/PDF/These_Karim_v_FINALE.pdf

[6] Juan A. Recio-Garcia, Beléen Diaz-Agudo, Pedro Gonzàlez-Calero, Antonio Sànchez-Ruiz-Granados livre : Ontology based CBR With jCOLIBRI. Proceedings of the Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Applications and Innovations in Intelligent Systems, AI 2006, 14, 149–162.

[7] Juan A. Recio-Garcia, Beléen Diaz-Agudo, Pedro Gonzàlez-Calero, Antonio Sànchez-Ruiz-Granados : jCOLIBRI2 Tutorial. Accessible à partir de l'url: http://gaia.fdi.ucm.es/files/people/juanan/jcolibri/downloads/tutorial.pdf.