MASTER EN SCIENCES INFORMATIQUES

Cadre de référence pour l'amélioration de la qualité d'un logiciel à base de connaissances
application à COSMIcxpert et SMXpert

Sandron, Stéphane; Vanderose, Benoit

Award date:
2005

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique.

Cadre de référence pour l'amélioration de la qualité d'un logiciel à base de connaissances (application à COSMICXpert et SMXpert.)

par

Stéphane SANDRON & Benoît VANDEROSE

Mémoire présenté pour l'obtention du diplôme de

Maître en informatique

Année académique 2004-2005

Résumé

Les étapes nécessaires à la restructuration d'un système à base de connaissances doivent offrir la possibilité d'augmenter la qualité de celui-ci. Dans ce mémoire, nous apportons des indices théoriques et pratiques allant dans ce sens. Pour la démonstration, nous choisissons un système à base de connaissances développé successivement par différentes équipes de chercheurs de l'ÉTS (École de Technologie Supérieure) à Montréal et des FUNDP (Facultés Universitaires Notre-Dame de la Paix) à Namur. COSMICXpert a été développé pour la première fois en 2002 dans le contexte d'une thèse de doctorat (Ph. D.) pour initier les apprentis mesureurs à la méthode de mesure fonctionnelle appelée COSMIC-FFP (ISO 19761). COSMICXpert a bénéficé d'améliorations en 2003 de la part d'une nouvelle équipe. En 2004, différents chercheurs de l'ÉTS et des FUNDP ont utilisé la « coquille » de COSMICXpert pour construire un nouveau système à base de connaissances appelé SMXpert. Ce mémoire explique le processus qui a été utilisé et présente quelques mesures de la qualité des deux systèmes. Il pose également une question théorique : s'agit-il d'un processus de réingénierie ou de restructuration ?

Abstract

The necessary steps to reorganize a software containing knowledge should offer possibility of increasing the quality of this one. In this memoir, we are bringing theoretical indices ands experiments, allowing supporting this conviction. For the demonstration we choose a knowledge system developed successively by different teams of searcher from ÉTS (École de Technologie Supérieure) in Montreal and FUNDP (Facultés Universitaires Notre-Dame de la Paix) in Namur. COSMICXpert was first developed in 2002 in the context of a Ph. D. thesis to help novice measurer with a functional measure method call COSMIC-FFP (ISO 19761). There was also some improvement in 2003 on COSMICXpert by a new team. In 2004, different searchers from ÉTS and FUNDP, used the COSMICXpert shell to construct a new knowledge system for software maintenance renamed SMXpert. This memoir explains the process that was used and how, with some measurement, this improves the quality of both knowledge systems. It also addresses a theoretical question: can we call this process a reengineering or a restrcuturation process?

Avant-propos

Ce document est l'aboutissement d'un stage de fin d'études à l'École de Technologie Supérieure (ÉTS) de Montréal, Canada. La réflexion et le travail fournis sont le fruit de deux étudiants en dernière année de maîtrise en informatique aux Facultés Notre-Dame de la Paix (FUNDP) de Namur, Belgique. Les résultats obtenus n'auraient pas été possibles sans le transfert de connaissances et la collaboration entre ces deux universités.

La rédaction de ce document n'est pas seulement l'œuvre de deux étudiants, c'est pourquoi nous voudrions remercier toutes les personnes qui, de près ou de loin, ont permis la réalisation de ce travail.

Nous remercions tout d'abord le Professeur Naji Habra, promoteur de ce mémoire, pour ses encouragements tout au long de notre stage et de notre mémoire.

Nous remercions également le Professeur Jean-Marc Desharnais, notre maître de stage, parmi les fers de lance du projet COSMICXpert, pour son aide précieuse et son perpétuel suivi, autant sur place qu'outre-atlantique.

Nous remercions, bien entendu, tous nos proches, pour leur soutien et leur patience.

Finalement, nous souhaiterions dédier une partie de cet avant-propos aux personnes ayant également consacré du temps et de l'énergie aux projets COSMICXpert ou SMXpert dont le Professeur Alain April (ÉTS). Nous remercions aussi les personnes du laboratoire GELOG de l'ÉTS.

Table des matières

Table des illustrations

Chapitre 1 : Introduction

1.1 Historique

La création de la méthode de mesure fonctionnelle COSMIC-FFP[1] [1] a été le déclencheur de plusieurs recherches dans ce domaine spécifique de la qualité logicielle. Cette méthode, visant à améliorer la mesure des logiciels de domaines autres que ceux de gestion, a fait l'objet en décembre 2002 d'une norme ISO/IEC portant le numéro 19761 [1]. De plus, utiliser les méthodes fonctionnelles de mesure du logiciel demande un apprentissage de durée variable puisque difficilement automatisable. Dans ce contexte, plusieurs recherches ont été réalisées pour faciliter l'apprentissage de la méthode COSMIC-FFP. Un peu plus tôt, des recherches sur la qualité des mesures fonctionnelles avaient été menées, recherches ayant conduit au développement d'un système à base de connaissances pour la méthode de mesure fonctionnelle COSMIC-FFP.

En janvier 2002, Solange Ndagijimana Munezero a été une des premières personnes à travailler, en collaboration avec le professeur Desharnais, lors de son stage de 4 mois à Montréal, sur la qualité des mesures fonctionnelles. Elle a analysé les résultats de mesure d'une trentaine de projets réalisés par trois mesureurs experts dans le cadre d'un contrat de mesure avec une grande entreprise. Les experts avaient qualifié la qualité de la documentation pour les fins de mesure et donc indirectement, la fiabilité des résultats de mesure. Ce travail l'a mené à la rédaction d'un mémoire, sous la supervision du professeur Desharnais, ayant pour titre: « Etude de la mesure des points de fonction: application de la méthode de mesure fonctionnelle COSMIC-FFP et analyse de la documentation fonctionnelle » [16].

Cette recherche était une prémisse pour la création d'un système à base de connaissances pour l'apprentissage de la méthode de mesure fonctionnelle COSMIC-FFP. Cette tâche fut réalisée dans le cadre d'une thèse de doctorat et plusieurs mémoires d'étudiants. Cette thèse suggère une approche cognitive pour permettre l'application de la mesure fonctionnelle ISO 19761.

[1] COSMIC est en fait l'acronyme de Common Software Measurement International Consortium.

Elle est à la base du projet COSMICXpert, qui tente de vérifier l'hypothèse selon laquelle un système à base de connaissances peut aider le mesureur à acquérir et à maintenir les connaissances nécessaires à la mesure fonctionnelle des logiciels.

La première mouture du logiciel COSMICXpert a été développée par un étudiant allemand, Tim Küssing lors de son stage à Montréal avec le professeur Desharnais [17]. L'implémentation du logiciel s'est faite en Visual Basic. L'utilisation de Visual Basic a permis la construction d'une base de connaissance composée d'un nombre important, voire excessif, de fichiers textes. L'outil permettait à l'expert d'entrer des cas problèmes à des fins d'apprentissage par un mesureur. L'outil a démontré la faisabilité du concept mais a aussi rapidement montré ses limites. Les limites principales étaient de deux ordres:

- le logiciel était en mode local, ce qui veut dire qu'il était difficilement accessible aux mesureurs
- les données ne pouvaient pas être validées facilement car un grand nombre de fichiers au format texte devaient être créés pour un seul cas (entre 5 et 10 fichiers texte par cas problème). Il y avait au départ 35 cas problèmes dans le logiciel à base de connaissances.

À l'automne 2002, François Gruselin et Julien Vilz, stagiaires à Montréal avec le professeur Desharnais, ont proposé une solution WEB qui avait pour avantage de rendre disponible le logiciel COSMICXpert à un grand nombre d'utilisateurs et aussi, par l'utilisation de fichiers XML, de réduire considérablement le nombre de fichiers. Le premier travail des stagiaires avait consisté à créer plusieurs nouveaux cas. Le nombre de nouveaux cas est passé de 35 à 105, alors que le nombre de fichiers texte était rendu à plus de 800. La solution proposée a permis de réduire ce nombre de fichiers à moins de 150 mais aussi de vérifier et valider plus facilement le contenu des fichiers [2].

Donc, lors de ce travail, ils formalisèrent les connaissances liées au domaine et explicitèrent les relations entre les différents concepts dégagés. Cette démarche permit de migrer la base de données vers un ensemble de fichiers XML. Il en a résulté un nombre de fichiers moins élevé et donc une facilité accrue de validation et de vérification de la base de connaissances. Dans cette deuxième version du logiciel COSMICXpert, l'interface « expert » est inexistante car seuls les besoins des mesureurs ont été pris en compte. De ce fait, la modification et la

maintenance de la base de connaissance sont toujours aussi compliquées. Par exemple, l'ajout d'un cas ne peut se faire que par la réécriture de fichiers XML précis. Ces étudiants ont produit un mémoire en 2003 intitulé: « Vérification et validation d'un système expert pour la mesure fonctionnelle (CosmicXpert) » [2].

Les stagiaires Christophe Duterme et Nicolas Fabry ont amélioré COSMICXpert en le dotant d'une interface « expert ». Les nouvelles fonctionnalités mises en place lors de ce travail permettent, en théorie, les modifications de la base de connaissance par l'expert (ajout, suppression, …). En plus de ce travail, ils ont ajouté un mécanisme permettant de gérer la concurrence lors d'interactions avec les connaissances. L'intégrité de la base de connaissances pouvait ainsi être préservée et les incohérences avaient moins de chances de se produire. Il ont produit un mémoire en 2004 intitulé: « Automatisation de la vérification d'une base de connaissances: apllication à l'interface Expert de CosmicXpert » [3].

À ce stade, le logiciel COSMICXpert sur le WEB était une réalité. Il est actuellement utilisé dans de nombreux pays à travers le monde. Cependant, le projet s'est étalé sur plusieurs années, ce qui a engendré une hétérogénéisation de la documentation ainsi que du code source. Au final, les architectures concrètes et physiques ne suivaient plus un modèle bien précis et il restait des erreurs handicapantes dans une utilisation journalière du programme.

En 2004, les stagiaires Stéphane Sandron et Benoît Vanderose ont entrepris une démarche ayant pour but d'obtenir un système à base de connaissances qui pourrait être adapté à un moindre coût à des domaines autres que celui de la mesure fonctionnelle (et plus particulièrement, la maintenance). La première étape fut d'homogénéiser les différentes documentations d'auteurs précédents et de corriger COSMICXpert en vue de le rendre totalement fonctionnel. Ensuite, ils ont analysé l'architecture physique en l'état, ainsi que les possibilités de la faire passer à une architecture du type « Modèle – Vue – Contrôleur ». Les résultats de leurs travaux sont présentés dans ce document.

Malgré toutes ces initiatives, il reste toujours beaucoup à faire. En effet, l'utilisation des interfaces est peu intuitive car elle n'a été le fruit d'aucune réflexion particulière. Il est à noter que les interfaces n'ont pas réellement évolué mais elles ont simplement subi des modifications en rapport à l'ajout de fonctionnalités au gré des versions.

1.2 Contexte

Au fur et à mesure de l'évolution de l'Humanité, nous accumulons de plus en plus de connaissances dans des domaines extrêmement variés. Quand cela concerne des spécialités plus techniques, il peut être souhaitable de décharger en partie l'expert pour qu'il puisse se concentrer sur la résolution de ce qui l'occupe.

En effet, la prise de décision peut faire intervenir plusieurs paramètres pour lesquels une petite différence entraînerait des conclusions radicalement différentes quant à l'attitude à adopter. Pour uniformiser ce genre de procédures et augmenter la productivité, il n'est pas utopique d'imaginer l'aide de systèmes à base de connaissances. Jeffrey J.P. Tsai, Alan Liu, Eric Juan et Avinash Sahay l'ont également remarqué dans un article ayant pour centre ce type de systèmes [4]:

> "As computers and networking technologies are getting more powerful and cheaper, more and more application softwares have been developed. Many industries, such as telecommunications, avionics, banking hospital, automotive, semiconductors, oil, pharmaceuticals, are all highly dependent on computers for their basic functioning. However, the techniques and tools to design and maintain complex software systems lag behind the growth of size and complexity of those systems."

Nous voyons de nos jours que l'évolution de l'informatique est de plus en plus perceptible au sein de la société. Il n'est donc pas impossible que la partie de la population ayant accès à ces technologies puisse en faire usage pour son travail. Dans ce contexte, les systèmes à base de connaissances dont l'interface est accessible via Internet ont tout intérêt à exister.

COSMICXpert fait partie de ceux-ci. Il est basé sur la méthode de mesure fonctionnelle qu'est COSMIC-FFP. Elle est le fruit de la recherche appliquée et a été développée, principalement, par le milieu universitaire. Malgré tout, on observe que ce type de mesure prend une place de plus en plus importante au sein de l'industrie. Cela s'explique par le fait qu'un grand nombre de projets informatiques accusent un certain retard et des erreurs découlant des contraintes liées au temps. Tous ces facteurs sont sources de dépenses

supplémentaires, surtout liées au fait que les organisations dépendent de plus en plus de l'informatique.

Pour tenter d'enrayer cette tendance, les entreprises se sont intéressées aux différentes méthodes de mesure de la qualité d'un logiciel et de l'évaluation de la charge nécessitant le développement de celui-ci. Il y a une nécessité de mesureurs ayant suivi des formations poussées. COSMICXpert vient les aider dans la conservation des connaissances acquises au cours de ces dernières. Cela assure à l'employeur de devoir réinvestir moins vite en reformation. En effet, le système à base de connaissances évolue au fur et à mesure des développements de la norme à laquelle il est rattaché (ISO 19761 dans ce cas-ci). Donc, par incidence, le mesureur va également suivre cette évolution.

Les avantages des systèmes experts sont perceptibles lorsqu'ils ont rapport à un domaine de connaissances extrêmement vaste [55]. Comme nous pourrons le remarquer dans la revue de littérature, les articles en rapport à la création de systèmes experts dans le domaine des processus de maintenance sont inexistants. Il constitue donc un domaine de recherche fertile. La revue de littérature mettra également en évidence le fait qu'une architecture logicielle de qualité est importante et qu'une architecture générique, pour certaines classes de systèmes experts, pourrait accélérer la mise en œuvre de ces derniers. Comme nous le verrons dans la problématique ci-après, l'architecture de COSMICXpert demande à être revue et sa généralisation à un autre domaine nous apportera de nouveaux enseignements.

Comme la mesure fonctionnelle, le domaine des processus de maintenance au sein de l'industrie est devenu très important pour les entreprises. En effet, elles possèdent souvent de grandes applications stratégiques pour leur survie. Les processus de maintenance correspondent à toutes les activités de correction ou de management qui entrent en compte après la mise en fonction officielle d'un programme. Ils assurent que les logiciels continuent à répondre de la manière la plus efficiente possible aux besoins de l'entreprise.

Pour mieux comprendre les processus de maintenance et en vue de les améliorer, le milieu académique, en particulier le SEI (« Software Engineering Institute »), a créé un modèle de perfectionnement de ceux-ci du nom de « Capability Maturity Model » (CMM). En 1995, le livre « The Capability Maturity Model: Guidelines for Improving the Software Process » était la référence pour la version 1.1 de ce modèle. En 2000, CMM a subi des améliorations et a été

officiellement renommé « Capability Maturity Model Integration » (CMMI). Maintenant, la combinaison de plusieurs modèles (« Capability Maturity Model for Software (SW-CMM) v2.0 draft C », « Electronic Industries Alliance Interim Standard (EIA/IS) 731 », « Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98 ») a permis l'élaboration d'un cadre de référence CMMI qui permet de déduire un modèle CMMI personnalisé qui tient mieux compte de la structure de l'organisation et de ses processus. L'utilisation d'un modèle CMMI permet à l'entreprise d'obtenir des conseils sur l'amélioration des processus de l'organisation et sur la façon de superviser le développement, l'acquisition et la maintenance de produits ou services. Le but final est bien sûr que les processus de l'entreprise soient stables et matures. Comme nous l'avons vu au paragraphe précédent, l'importance de ces processus révèle qu'un système à base de connaissances serait d'une grande utilité dans l'accélération des réponses des responsables de la maintenance.

Le professeur Alain April de l'Ecole de Technologie Supérieure (ETS), de Montréal, a proposé, dans le cadre d'une recherche doctorale, un modèle sur base d'une liste de problèmes liés à la maintenance logicielle et reconnus comme ayant un degré de survenance élevé. Il a été baptisé « Software Maintenance Maturity Model » (**SM^{MM}**). Il permet d'améliorer les processus de maintenance à partir du moment où l'entreprise a identifié la situation dans laquelle elle se trouvait et qu'elle tient en compte les recommandations apportées. Cette façon de procéder correspond, à quelques détails près, à la façon dont procède COSMICXpert pour apporter des recommandations au niveau de la mesure logicielle. De plus, le domaine étant relativement statique, une base de connaissances conviendrait parfaitement dans le cadre d'une représentation du modèle **SM^{MM}**. A partir de ces deux derniers arguments, la création d'un système à base de connaissances était envisagée [5]. Sur base de COSMICXpert et du modèle de maintenance du professeur Alain April, le logiciel SMXpert a été créé et est une tentative d'aide à la décision pour les experts de ce domaine travaillant pour l'industrie.

1.3 Revue de littérature

Ce chapitre présente la revue de littérature liée à notre recherche. Une première section traitera des architectures logicielles, de leurs qualités, ainsi que de l'adaptabilité des architectures de systèmes à base de connaissances à un autre domaine d'expertise. Dans une

deuxième section, nous aborderons brièvement les bases de connaissances, leurs architectures possibles et l'adaptabilité de ces dernières à un autre domaine d'expertise. La Figure 1.1 schématise, à un haut niveau, la démarche de la revue de littérature.

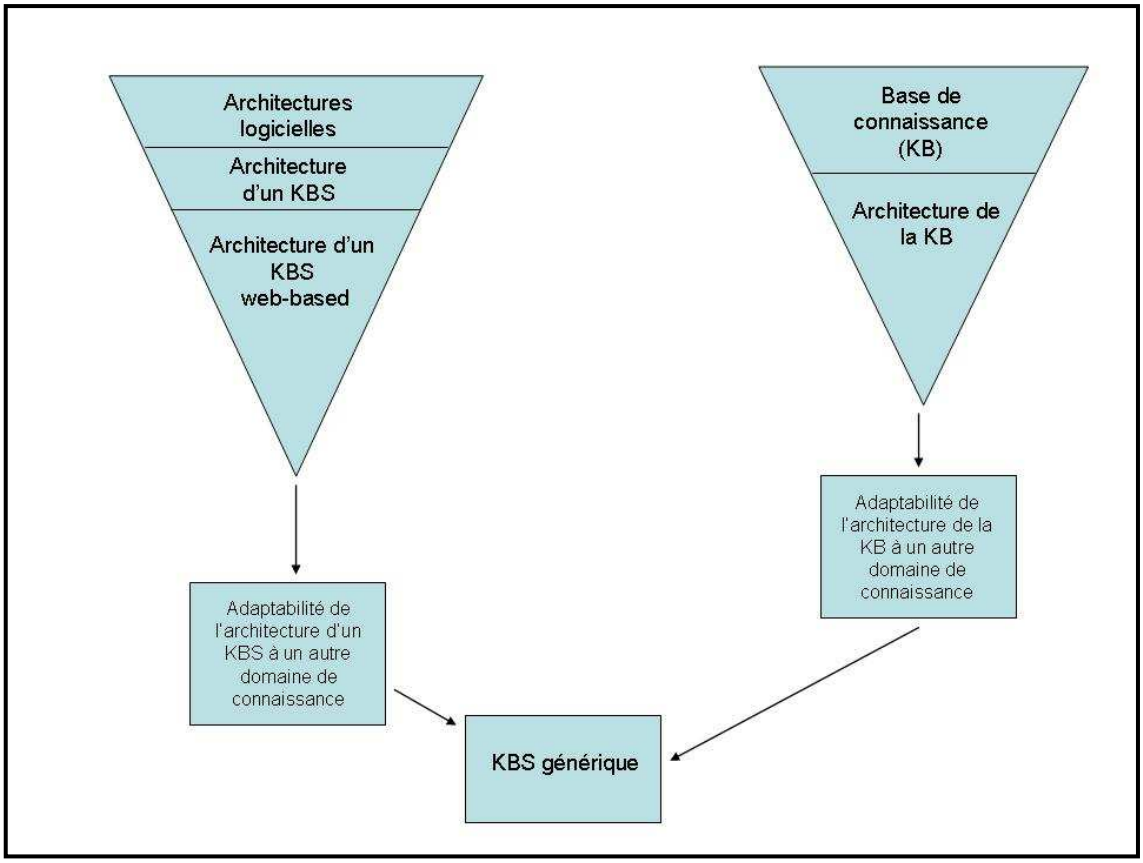


Figure 1.1 : Démarche de la revue de littérature

1.3.1 Domaine des architectures

1.3.1.1 Architectures logicielles

La notion d'architecture est un concept très important à comprendre car elle implique un nombre important d'individus lors du développement et de la maintenance d'un logiciel. De plus, il convient qu'elle soit comprise par des personnes aux compétences parfois fort différentes. Cependant, comme l'écrit Kari Smolander [6], le monde académique n'a jamais réussi à donner une définition acceptée de tous. En général, elle est trop large que pour permettre une utilisation pratique de celle-ci. Kari Smolander propose une explication à ce problème [6]:

> "A plausible explanation to the vagueness of architecture as a concept is that in practice, situations vary so much that a general and usable definition is not possible. The variation includes differences, for instance, in technology, products, development methods, business environment, organizational roles, and external stakeholders. Any strict definition of the concept of architecture would probably prove unsatisfactory at some situation."

Après avoir effectué des interviews auprès de différentes personnes ayant des contacts avec l'architecture, Smolander est arrivé à la conclusion que la notion d'architecture variait en fonction de la position de la personne et de son expérience. Par exemple, pour les ingénieurs logiciels, l'architecture équivaut à une spécification à implémenter, tandis que pour les managers, ce serait plutôt un moyen de communication.

En général, l'architecture est le réceptacle des informations dont a besoin chacun des intervenants pour spécifier le programme. Il est à noter que l'ampleur de ces besoins peut varier de manière individuelle. Ces spécifications sont liées à des concepts tels que « l'achat ou la vente du système », « l'exécution du système », « la compréhension de ses fonctionnalités », « la compréhension du scope du projet et l'estimation de son progrès », « le *design* d'une structure de haut niveau du système », « la programmation des composants du système » et à beaucoup d'autres aspects.

Selon Lakoff et Johnson [7], la plupart de nos systèmes conceptuels sont structurés de manière métaphorique. En effet, beaucoup de nos concepts sont soit abstraits, soit issus de notre expérience. Ils ne sont donc pas clairement définis. De même, nous pouvons voir l'architecture comme une métaphore qui nous permet de comprendre la structure d'un logiciel. Celle-ci peut également être décomposée en quatre métaphores qui permettent de mieux comprendre ce qu'est une architecture (perçue sous différents aspects) [7]:

- **L'architecture comme un modèle.** Elle est considérée comme étant la structure de haut niveau du système à implémenter. Les descriptions de l'architecture sont utilisées pour transférer des informations explicites entre les ingénieurs logiciels eux-mêmes, et aussi avec les programmeurs. Sa portée temporelle est sur le long terme car son but est de décrire le futur système pour permettre sa réalisation et ses modifications à venir. Cela nécessite l'utilisation d'un haut niveau de formalisme et de détails dans ses descriptions.

- **L’architecture comme littérature.** Elle réside dans la documentation des structures techniques et du transfert de l’information, de la connaissance dans le temps. Par cela, nous voulons parler de la documentation comme d’une description de l’architecture servant à aiguiller le futur lecteur en recherche d’une solution architecturale. Cela implique des niveaux de formalisme et de détails qui peuvent varier fortement mais qui, la plupart du temps, sont très élevés.
- **L’architecture comme un langage.** Elle constitue le langage pour réaliser la conception du système. Dans ce cas-ci, l’objectif principal n’est plus de transférer la connaissance mais de comprendre l’objet sur lequel on travaille. L’objectif est aussi de disposer d’un outil auquel se référer pour faciliter la communication entre les différents intervenants. C’est pour cela qu’il ne faut pas que les niveaux de formalisme et de détails soient trop élevés si on veut pouvoir être compréhensible du plus grand nombre.
- **L’architecture comme décision.** Elle représente les décisions prises pour la structure du système à implémenter. Ces décisions ont une influence sur les ressources nécessaires à la création du système (nombre d’employés, besoins de compétences spéciales, *etc*.), sur la stratégie de développement à adopter, mais aussi sur les besoins relatifs à la qualité intrinsèque de l’architecture (facilité d’utilisation, de maintenance, de compréhension et performance de celle-ci).

Kari Smolander [6] finit par conclure que ces quatre précédentes métaphores permettent aux chercheurs de mieux spécifier et expliquer l’intérêt et l’objet de leurs recherches sur l’architecture logicielle.

Il est à noter que ce sont surtout les deux premières métaphores qui sont importantes aux yeux des chercheurs et des ingénieurs logiciels. Nous pouvons le remarquer par la définition qu’en donnent Jan Bosch & PerOlof Bengtsson lors de leurs recherches sur l’évaluation de la capacité de maintenance d’une architecture [8] :

> “The software architecture defines the most fundamental design decisions and the consequent decomposition of the system into its main components and relations between these components.”

Il en est de même dans les travaux de Bonnie E. John et Len Bass [9]. Ils qualifient l’architecture de la façon suivante:

> “The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them. It is the earliest

artefact in the development lifecycle that can be analysed in any meaningful or quantitative way."

La vision de la notion d'architecture peut varier d'une personne à une autre. Dans cette optique, le « Software Engineering Institute » (SEI) a créé une page web sur laquelle ils se posent la question de savoir comment définir une architecture (« How Do You Define Software Architecture ») [Web1] et où ils rassemblent un ensemble exhaustif de définitions.

Le formalisme dans lequel l'architecture est exprimée peut varier en fonction des compétences et des habitudes des personnes qui s'occupent du *design* de celle-ci. Ce qui est important est que le méta modèle sur lequel se base ce dernier doit être compris de tous et être reconnu pour le fait qu'il ne génère pas d'incompréhensions lorsqu'il est utilisé en pratique. Le plus utilisé dans le monde de l'ingénierie logicielle est UML.

Pour ce qui est de la façon de créer une architecture à partir des besoins des utilisateurs et de l'environnement dans lequel évoluerait le système, Jacobsen, Kristensen et Nowack affirment que l'architecture est l'équivalent d'abstractions sur le domaine du logiciel [10]. Effectuer des abstractions dans un domaine revient à créer de nouveaux « concepts » et éléments dans un autre domaine, un domaine abstrait. Les éléments de ce dernier décrivent les éléments du précédent. En passant d'un domaine à un autre, on créée de la connaissance liée au domaine originel (dans notre cas, le domaine du logiciel) et ce n'est pas une simple réorganisation.

Jan Bosch & PerOlof Bengtsson ont exprimé ce processus de manière plus schématique et en donnant une place importante à la créativité des ingénieurs logiciels impliqués [8] :

"The architecture design process (…) can be viewed as a function taking a requirement specification that is taken as an input to the method and an architectural design that is generated as output. However, this function is not an automated process and considerable effort and creativity from the involved software architects is required."

Modèles selon lesquelles sont basées les architectures?

L'article de Bonnie E. John et Len Bass nous donne plus d'informations sur l'historique des développements des modèles sur lesquels se basent les architectures logicielles actuelles [9].

Vers le milieu des années 70, une étude d'IBM (réalisée par Sutton et Sprague en 1978) est arrivée à la conclusion que ce qui est le plus coûteux lors de la maintenance est l'interface utilisateur (près de 50% du budget). La communauté scientifique est alors venue à la conclusion qu'il serait préférable de séparer cette interface de la partie contenant les traitements.

Cependant, il fallait que ces deux parties puissent communiquer. C'est dans ce cadre que deux types d'architectures se sont distingués : l'architecture de type « Model-View-Controller » et celle de type « Seeheim ».

Le premier type a été décrit par Krasner et Pope en 1988 pour répondre aux besoins des développeurs en orienté objet (bien que l'architecture soit indépendante de l'implémentation). Il était initialement centré sur la présentation d'un objet à l'écran. La vue détermine l'output de l'objet, le contrôleur détermine l'input de celui-ci et le modèle détermine les actions applicables.

La question se pose alors de déterminer l'importance de l'architecture.

L'utilisation de logiciels est de plus en plus répandue. Jan Bosch et Lars Lundberg apportent une explication à ce phénomène [11]:

> "One of the reasons for the increased use of software is the increased ability to control and adapt the behaviour of systems late in the development process as well as after deployment. Another aspect of the increased use of software is the fact that the responsibility for system properties such as safety, security, reliability, flexibility and usability, has moved from mechanics and electronics to software."

De plus, comme le disent Jacobsen, Kristensen et Nowack [10], une architecture est toujours présente dans chaque logiciel. Cependant, elle n'est pas toujours explicite. Elle peut avoir existé pendant la phase de *design* sans jamais avoir été exprimée explicitement et est maintenant perdue ou intégrée en partie dans la documentation. Cela a des conséquences sur

la qualité du logiciel. En effet, Jan Bosch & PerOlof Bengtsson ont également remarqué ce problème [8] :

> "Within the community, we have become to understand the fact that the quality attributes of a software system are, to a large extent, constrained by its architecture. Consequently, the quality requirements most central to the success of the software system should drive the design of the software architecture. Unfortunately, the design of software architectures is often performed in a similar fashion as software design as a whole, i.e. as an experience-driven, implicit activity."

Ces mêmes chercheurs disent que les qualités d'un logiciel ont comme contrainte l'architecture. De plus, leurs vues sont également partagées par Jan Bosch et Lars Lundberg [11]:

> "One of the important lessons that we have learned over the last decade is that the architecture selected for a software system constrains the quality attributes. Thus, when, for instance, selecting a particular architectural style, the immediate consequence is that boundaries for one or more quality attributes have been selected as well."

Une autre remarque dans ce sens se retrouve sur un site Internet dédié aux architectures logicielles [Web2] :

> "Architectural decisions directly impact system qualities, and often a decision in favor of one quality has an impact on another. It is also important to remember that the benefits of the most applicable architecture design can be eliminated by poor implementation."

L'impact de l'architecture sur les qualités d'un système est du à la précocité du *design* de cette dernière dans le développement de l'application. En effet, il sera très difficile de les influencer, voire de modifier ces choix dans la suite du cycle de vie. D'où le besoin d'être explicite et objectif dès le départ.

Un grand nombre de logiciels continuent à évoluer après la phase de déploiement chez l'utilisateur. En effet, il est important de corriger les éventuelles erreurs qui pourraient se produirent ou d'ajouter certaines fonctionnalités. Lars Bratthall et Claes Wohlin expliquent que pour faire évoluer un logiciel il faut pouvoir comprendre son fonctionnement et que la seule lecture du code source n'est pas possible [12]. L'architecture et les descriptions du *design* sont les éléments clés qui favorisent la compréhension car ils permettent d'atteindre un niveau d'abstraction plus élevé que le code source.

Comme Jan Bosch et Lars Lundberg le spécifient, la relation entre les choix de *design* de l'architecture et les qualités d'un système est un domaine qui fait l'objet d'intenses recherches [11]. Les connaissances dans ce domaine augmentent d'années en années. Jan Bosch a lui-même proposé une méthode de *design* d'architecture qui permet d'évaluer et d'effectuer des choix par rapport aux spécifications des qualités [11]. Il existe également des styles architecturaux et des *design patterns* qui explicitent les relations qu'ils possèdent avec les qualités logicielles.

Les qualités d'un logiciel peuvent être classées en quatre catégories comme nous pouvons le voir sur ce site [Web2] : les qualités liées à l'exécution du système, les qualités non liées à l'exécution du système, les qualités commerciales et les qualités de l'architecture. En plus de celles-ci, nous pouvons tenir compte des qualités qui sont spécifiques au domaine particulier dans lequel s'exécute le système.

Les qualités liées à l'exécution du système :

Ces qualités peuvent être mesurées lorsque le système s'exécute [Web2].

- **« Functionality »**. La fonctionnalité d'un système équivaut à sa capacité de traiter le travail pour lequel il a été conçu. Par là, nous entendons qu'il respecte bien le cahier des charges et donc les spécifications de départ.
- **« Performance »**. La performance est liée au temps de réponse, l'utilisation et le comportement du système.
- **« Security »**. La sécurité est liée à la capacité du système de résister à des tentatives de modifications non autorisées de l'usage ou du comportement du système pendant qu'il continue à servir des utilisateurs légitimes.
- **« Availability »**. La disponibilité du système est une mesure du temps pendant lequel il est actif et s'exécute correctement. On prend en compte également les intervalles entre les *crashes* et le temps nécessaire à la réparation du système.
- **« Usability »**. Selon Bonnie E. John et Len Bass [9], l'utilisabilité d'un programme reprend des notions comme l'efficacité, sa facilité d'apprentissage et la satisfaction que son utilisation procure à l'utilisateur. Contrairement aux qualités

précédentes, celle-ci est relative aux qualités d'un programme. Cependant, les auteurs de l'article précisent que même si le *design* de l'interface et des fonctionnalités d'un système est correct, l'utilisabilité d'un système peut être compromise si l'architecture de base n'a pas pris en compte les besoins humains autres que ceux liés à sa facilité de la maintenir. Par exemple, s'il faut pouvoir prendre en compte l'affichage des différentes langues, alors il faudra qu'il existe un ou plusieurs composants dans l'architecture pour pouvoir gérer cet aspect.

- **« Interoperability »**. L'interopérabilité est la capacité d'un système d'interagir avec un autre système.

Les qualités non liées à l'exécution du système :

Ces qualités ne peuvent pas être mesurées lorsque le système s'exécute [Web2].

- **« Modifiability »**. Une des principales qualités d'un système est la facilité d'effectuer de la maintenance, c'est-à-dire de le modifier. Elle est fortement liée à l'architecture car la façon dont a été décomposé le système a un impact sur les composants ciblés par une modification. Dans l'article de Jan Bosch & PerOlof Bengtsson, la « maintainability » est définie par la facilité avec laquelle on peut exécuter des tâches de maintenance de différentes sortes [8]. En général, elle est associée à un effort, un coût pour effectuer les tâches précédentes. En effet, il est très important de pouvoir facilement modifier le système car cela permet de rendre le logiciel qui s'y rapporte plus évolutif. Imaginons que l'architecture d'un programme demande 10 ingénieurs logiciels à temps plein pendant un an pour effectuer la maintenance, les chefs de projet privilégieront les architectures alternatives qui permettent de diminuer ce coût en personnel. Par incidence, cela engendrera une réponse plus rapide aux nouveaux besoins des utilisateurs.
- **« Understandability »**. La compréhension que l'on peut avoir d'un système va de pair avec la qualité précédente. En effet, il est plus facile d'effectuer des modifications lorsque l'on sait exactement où elles se situent. Cela est possible grâce au plus grand niveau d'abstraction que procure l'architecture. La manière dont elle est exprimée joue également un grand rôle dans la compréhension.
- **« Portability »**. La portabilité d'un système est la capacité de celui-ci dans différents environnements logiciels. Ces derniers peuvent se différencier au niveau du hardware ou du software, et est généralement une combinaison des deux.
- **« Reusability »**. La réutilisabilité est la capacité d'utiliser un système dans d'autres applications.
- **« Integrability »**. L'intégrabilité d'un système ou d'un composant de celui-ci est la capacité de travailler correctement avec d'autres systèmes/composants qui ont été développés séparément.
- **« Testability »**. La testabilité correspond à la facilité de tester un logiciel. Mikael Svahnberg nous en donne une définition plus précise [13]: *Testability involves the ability to test and debug the parts of the system individually and/or as a group, the ability to extract test data from the system and the ability to add data measurement points when needed, but also the ease by which these things can be done.*

Les qualités commerciales [Web2]:

- **« Cost and Schedule »**. Il faut comprendre que cette qualité inclus le prix du système en respect au « time to market », le durée de vie attendue du projet et la planification de l'utilisation de systèmes dits « ancêtres ».

- **« Marketability »**. Cela correspond à l'utilisation du système en accord avec la compétition sur le marché.
- **« Appropriateness for Organization »**. Disponibilité du facteur humain, de l'expertise et d'une structure en rapport au logiciel.

Les qualités de l'architecture [Web2]:

- **« Conceptual Integrity »**. L'intégrité de l'architecture globale qui est composée de plus petites structures architecturales.
- **« Correctness »**. Une architecture correcte correspond au fait que le logiciel effectue ses fonctionnalités comme prévu, conformément aux spécifications de ces dernières. De manière plus directe, c'est le degré avec lequel le système traite les fonctionnalités demandées.

En conclusion des articles précédents, on peut légitimement dire que la qualité dépend de ceux qui créent l'architecture. Mikael Svahnberg pointe du doigt le fait que les décisions sont souvent prises par intuition [13], généralement en se basant sur l'expérience de seniors de l'ingénierie logicielle. Cela peut conduire à des logiciels devant entrer dans un moule, qui peut ne pas convenir au mieux pour répondre aux besoins du domaine du logiciel.

Cependant, il est encore difficile à l'heure actuelle de pouvoir évaluer les qualités d'une architecture au moment de sa création, ou encore l'impact qu'elle aura sur les qualités du système. La communauté des ingénieurs logiciels en est consciente et des travaux, comme ceux de Jan Bosch & PerOlof Bengtsson, se développent pour tenter d'évaluer ces qualités dans la phase de *design* du logiciel. Malgré tout, il existe d'autres techniques d'analyse des architectures logicielles. Elles se classent en deux catégories : les techniques « architecture oriented » et les autres « quality attribute oriented ». En fait, les premières sont basées sur des évaluations par des équipes d'ingénieurs logiciels (ou d'autres intervenants) à partir de leur expérience. Elles sont effectuées après la phase de *design* de l'architecture. Les secondes sont basées sur la comparaison de deux architectures bien précises d'une qualité bien précise et sur des prévisions quantitatives comme l'évaluation du comportement en temps réel. Elles peuvent être réalisées itérativement lors de la phase de *design* de l'architecture.

1.3.1.2 Architectures des systèmes à base de connaissances (KBS)

Après avoir développé le domaine des architectures logicielles, cette partie va se concentrer sur les travaux en rapport aux architectures des systèmes à base de connaissances. Ce dernier

groupe est un sous-ensemble des premières et, donc, l'entièreté de ce qui a été exposé au point précédent est valable.

Mais avant tout développement approfondi, il est important d'expliciter ce que l'on entend par système à base de connaissances afin que toute référence ultérieure à cette notion soit partagée par le lecteur et les auteurs. Dans la suite, nous nous référerons aux systèmes visés en tant que « systèmes experts » ou, de façon interchangeable, en tant que « système à base de connaissances ».

Avant de donner des définitions en rapport aux systèmes experts et à base de connaissances, il faut clarifier les esprits sur la distinction entre ces deux-ci. Prince, citée par J.-M. Desharnais [15, p. 39], écrit à ce sujet [14]:

> "[…] des raisons de pertinence, de vérité et de situation d'échec dans les organisations ont entraîné un abandon progressif de cette dénomination de système expert, aussi bien dans les milieux de la recherche que dans les structures industrielles. Celles qui tend à la remplacer est celle de système à base de connaissances initialement adopté pour des raisons de marketing et qui finit par avoir une authenticité académique."

De part sa nature évolutive, le domaine des systèmes experts fournit pléthore de définitions pour son objet principal. Nous pouvons trouver une définition générale de système expert dans Jackson [18, p. 2]:

> "An expert system is a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problem or giving advice."

D'autres définitions viennent étoffer cette première classification. Nous en trouvons notamment dans [19]:

> " La connaissance des experts est souvent rare et précieuse. Les systèmes experts sont des programmes d'ordinateurs qui capturent une partie de cette connaissance et permettent la diffusion de celle-ci à d'autres.

Un système expert (sur la base de connaissances) est un système de résolution de problèmes et d'aide à la décision basé sur des connaissances d'un domaine particulier et sur des règles logiques ou sur des procédures pour utiliser ces connaissances. Les connaissances comme la logique sont toutes deux issues de l'expertise d'un spécialiste du domaine.

Un système expert est un programme qui émule l'interaction qu'un utilisateur pourrait avoir avec expert humain, pour résoudre un problème. L'utilisateur fournit les données. Le programme va poser des questions jusqu'à ce qu'il arrive à une conclusion. La conclusion peut être une solution unique ou une liste de solutions possibles. Le système peut expliquer en langage naturel comment il est arrivé à cette conclusion et pourquoi."

Janet Efstathiou, Ani Calinescu et Guy Blackburn de l'université d'Oxford proposent une très bonne définition [20] :

"The key feature of an expert system is the distinction between the suite of software and the knowledge that it manipulates. Whereas traditional programs embedded the domain knowledge within the program, expert systems represent the knowledge in a separate knowledge base (KB). The KB contains the representation of the domain knowledge, i.e. the knowledge from a particular domain of expertise that is used to solve the problems presented to the expert system. The program that manipulates the domain knowledge and combines it with data taken from the world to produce a conclusion is known as the inference engine. When an expert system interacts with the world via a user, the system is being used in consultative mode. The alternative is to operate as a real-time program, receiving data directly from sensors."

Mais encore, Christine W.J. Chee et Margaret A. Power nous apprennent que [21]:

"A knowledge-based system is a heavily symbolic computer program which has a separate declarative knowledge base and inference engine. An expert system is a specific kind of knowledge-based system, usually built using expert system tools, technology, and languages. Expert systems best solve specific complex problems, and do so quickly and skilfully using a human expert's knowledge and problem-solving ability."

M. De la Sen, J.J. Miñambres, A.J. Garrido, A. Almansa et J.C. Soto distinguent bien deux types de systèmes experts [22]:

> "The term expert system was originally used to denote systems using a significant amount of expert information about a particular domain in order to solve problems within that domain [2,15,25,28]. Due to the important role of knowledge in such systems, they have also been called knowledge-based systems (see [15,16]). However, since the terminology has been applied to so many diverse systems, it has essentially evolved into two different uses of the term. First, the term is often used to describe any system constructed with special kinds of "expert-systems" programming languages and tools, including production systems, rule-based systems, frame-based systems, "blackboard" architectures, and programming languages such as Lisp or Prolog. The other important feature is that, since they are usually non-deterministic, a large number of modules may be "applicable" (candidates for activation) at any given moment. Thus, a criterion is needed to determine how to select which of the applicable modules must be executed next, and what to do after selection. This second is the more appropriate job of an expert system in the sense that it is a system that "reasons" about a problem in much the same way humans do."

Le système est articulé autour d'une base de connaissances qui sera abordée à la Section 1.3.2.

Le fait que le système développé soit un système expert a des impacts sur la façon d'atteindre certaines qualités. Nous nous concentrerons sur la facilité à maintenir un logiciel. En effet, Christine W.J. Chee et Margaret A. Power ont participé à l'élaboration d'un processus de *design* de systèmes experts permettant de tenir compte de l'aspect « maintainability » [21]. Selon elles :

> "Many traditional software engineering techniques may successfully apply to expert systems. However, some techniques have a greater impact on expert systems design than in conventional software. Expert systems differ from conventional software in two significant ways: program structure, and application. These

differences affect which software engineering principles enhance expert system maintainability."

Suite à ces lectures, nous concluons que les différences, ayant un impact sur la maintenabilité, se retrouvent au niveau :

- De l'existence d'une base de connaissances qui engendre une documentation supplémentaire
- D'une documentation des raisonnements humains que le système tente de reproduire
- D'erreurs dans la base de connaissances engendrant des réponses incorrectes par le système qui fonctionne parfaitement
- D'un mécanisme d'informations de l'utilisateur sur le raisonnement qu'a eu le moteur d'inférence pour en déduire la réponse (ceci n'est pas présent dans tous les systèmes à base de connaissances)
- De la nécessité d'obtenir l'aide d'un expert du domaine lorsque l'on développe le système
- De l'utilisation d'outils de développement de systèmes experts comme des « shells ». Christine W.J. Chee et Margaret A. Power affirme que cela aide à la maintenance car les changements dans le code source ne tiendront pas en compte des complexes mécanismes d'inférence [21].
- Du peu de modularité des composants de l'architecture dont font l'objet les systèmes à base de connaissances. Ce point est généralement prépondérant dans l'échec d'un système expert [21].

Il faut alors s'intéresser à l'architecture d'un système à base de connaissances.

Les systèmes à base de connaissances sont différents sur certains points mais dans la pratique sont des logiciels à part entière.

Dans les types d'architectures communément utilisées lors de la création de systèmes à base de connaissances, on retrouve l'architecture de type « blackboard ». Par exemple, Xiaoyi Chi, Ma Haojun, Zhao Zhen et Peng Yinghong ont proposé celle-ci pour l'architecture d'un système expert dans le domaine du « blanking technology design » [23]. L'avantage de ce type d'architecture est qu'elle est capable de gérer des sources de connaissances différentes, qui peuvent de ce fait être représentées de manières différentes. Voilà ce qu'ils en disent dans [23]:

"The blackboard model provides a very flexible control structure for judging and solving problems. In the blackboard model, information in different processes (or at

different levels) is described on the blackboard, then recommending/reference information is output."

Comme on peut le voir dans la figure ci-dessous, l'architecture de type « blackboard » est constituée de trois composants principaux [23]:

- La connaissance utilisée pour résoudre les problèmes est divisée en différents composants de type « Knowledge Source » qui sont indépendants les uns des autres.
- Le composant « Blackboard data structure » représente le « blackboard ». Ce dernier est l'équivalent d'une base de données globale qui enregistre et organise les données intermédiaires à la résolution d'un problème. Les composants de type « Knowledge source » génèrent des changements au niveau du « blackboard » qui permettent d'arriver à la solution d'un problème de manière incrémentale. Cette technique constitue donc le seul moyen de communication entre ces composants.
- Le composant « Controller » contrôle les changements apportés au « blackboard » et, en fonction de ceux-ci, décide de quelles seront les prochaines actions. En fait, il peut être localisé dans les composants de type « Knowledge source », dans le « blackboard » et dans les composants isolés ou bien dans une combinaison de ces trois derniers.

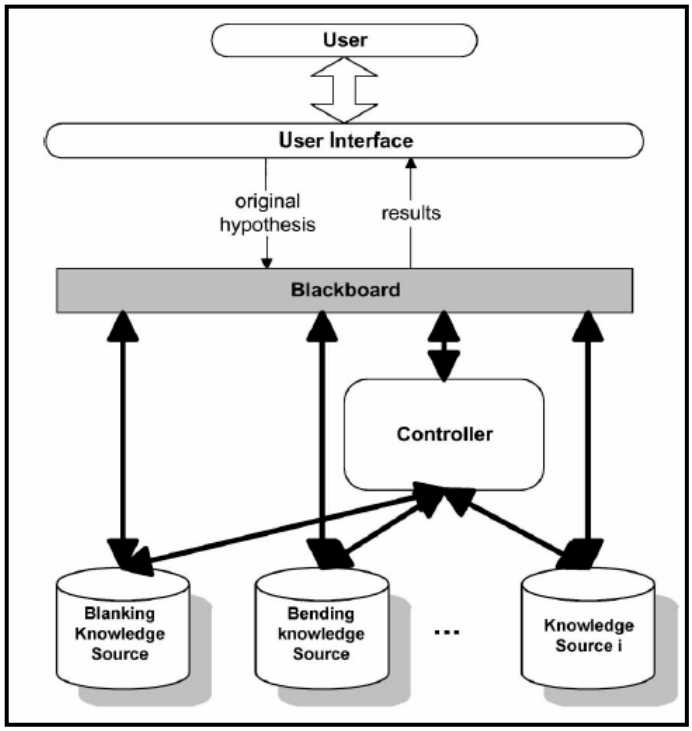


Figure 1.2 : Architecture de type « blackboard » [23]

Evelyne Tropper et Sylvain Béland montrent dans leur article intitulé « A New Expert System Architecture For Decision Support » que l'architecture de type « Blackboard » est celle qui est la plus adéquate [24].

Giovanna Avellis, L. Borzacchini et A. Cavallo ont développé un système expert du nom de « Software Maintenance Expert System » (SMES) destiné à aider l'ingénieur logiciel à facilement modifier un système. L'équipe s'est également dirigée vers ce type d'architecture car [25]:

> "Our development of a blackboard architecture was motivated by the need for integration of different knowledge sources in software maintenance."

L'architecture logique de type « Model – View – Controller » est également utilisée dans certains systèmes à base de connaissances. Curtis H.K. Tsang et Chris Bloor présentent un cadre de référence générique pour les systèmes experts médicaux complexes basé sur une architecture inspirée de celle de type MVC [26].

> "The rapid prototype of an expert system can be achieved by to three major responsibility of system components. They are namely the Object Static Component, Presentation Components and Inference Components. This idea is borrowed from the Model View and Control (MVC) approach."

Il propose une explication plus precise de la variante de l'architecture MVC [26]:

> "The medical expert system consist of three distinct responsibilities namely: the representation system component for man-machine interaction role, the static object component for data and its transformation role and the rule-set system component for intelligent inference role. The communication among these system components are through message passing among objects and images."

Lien entre l'architecture et la base de connaissances

Lors de la création d'un système expert, Alan T. Demmin et Du Zhang ont remarqué que le *design* de l'architecture avait une grande importance dans la formalisation de la base de connaissances. En effet, comme expliqué en [27]:

> "UML class diagrams were developed for the input/out object model and the high-level interface design. This object model was then used for the term definitions within the rulebase. One of the first design considerations of any software system is the identification of data elements that will be used as input to the system, and similarly, the data elements that will be returned or updated by the system. When designing rule-based systems, these data elements are the terms and facts that are part of the knowledge base. The first step in designing a rulebase is to identify all of the data elements or terms within the knowledge and assign appropriate term names. The importance of this effort must not be overlooked. A term name can make a significant difference in the comprehension of the rules that utilize these terms."

1.3.1.3 Architectures des systèmes à base de connaissances « web-based »

Les exemples suivants illustrent les différents choix posés en matière d'architectures logiques et physiques lors de la création de systèmes à base de connaissances « web-based ».

Web-based expert system for vehicle registration

Alan T. Demmin et Du Zhang ont travaillé à la création d'un « web-based expert system » pour l'enregistrement de véhicules avec lequel les utilisateurs pourront connaître les prix en vigueur. Ils ont fait le choix d'utiliser une interface accessible via Internet , comme expliqué en [27]:

> "Recent e-business initiatives are requiring companies to provide "web-enabled" systems in order to maintain a competitive edge in their marketplace."

Dans leur article [27], ils présentent leurs motivations et leurs conclusions :

> "The purpose of this study is to investigate the feasibility of utilizing an inference capability to administer the business logic for an enterprise within a web-based

architecture. Our work has shown that using expert system technology to implement a web-based, business-driven solution is certainly a viable and promising option."

Le système a été développé sur base du modèle de référence d'une application serveur de Sun en J2EE. Les diagrammes de classes UML ont été créés pour le modèle objet entrée/sortie et le *design* des interfaces de haut niveau. A partir de ces diagrammes, ils ont pu identifier les « data elements » qui sont utilisés en entrée et en sortie du système. Comme cela a été expliqué précédemment, cette étape est très utile pour pouvoir formaliser la base de connaissances. Au final, l'architecture logique était composée de plusieurs objets [27]:

FeeTransaction - the type of transaction being requested
Feevehicle - the vehicle that is involved in the transaction
FeeOwner - the owner of the vehicle that is involved in the transaction
FeeResult - the resulting fee list and related information for the transaction
FeeRequest and FeeResponse - (...) to encapsulate all of the functionality necessary to translate data between a serialized and deserialized format, to match requests with responses, and to designate which rule service will be accessed within the rules engine.

Quant à l'architecture physique, c'est une architecture de type « multi-tier ». La logique d'affaires et le contrôleur se trouvent du côté serveur de l'application. La base de règles est, quant à elle, localisée dans un *repository* séparé. Pour finir, l'interface client est représentée par ce qui permet d'accéder aux services définis via les composants de la logique d'affaires.

Pour ce qui est de l'implémentation, le système expert a été développé en Java. Les servlets ont été utilisés pour le contrôleur et les « Entreprise Java Beans » pour certains composants de la logique d'affaires. Il est intéressant de noter que Alan T. Demmin et Du Zhang ont fait l'usage d'un moteur d'inférence configurable : « Blaze Advisor ». Cela leur a permis de ne pas devoir implémenter un nouveau moteur raisonnant sur base de règles.

Les performances liées au choix d'une interface web sont également exposées (cf. [27]):

"The performance of the web application has a response time within one second. The rules engine itself consumes two hundred to three hundred milliseconds, and the

network traffic can affect the final response time. There is a startup cost associated with the rules engine in the form of loading external and internal table information and setting up the Rete network. This generally takes about fifteen seconds during the first invocation of the EJB. Once the EJB is initialized, it remains in memory for the EJB container to process future requests."

Digital Services Test Sytem (DSTS)

R. Rong, D. Brooks, G. Fu et E. Eichen ont créé un système expert qui permet de savoir si une ligne téléphonique peut être le support d'une technologie de connexion Internet de type *Digital Subscriber Line* (*DSL*) [28]. En effet, la longueur des lignes téléphoniques, le type de services convoyés par les lignes cohabitant avec les premières et le type de câblage sont des variables qui entrent en jeu dans les possibilités de connexion offertes.L'architecture physique du système client/serveur est de type « 3-tier ».

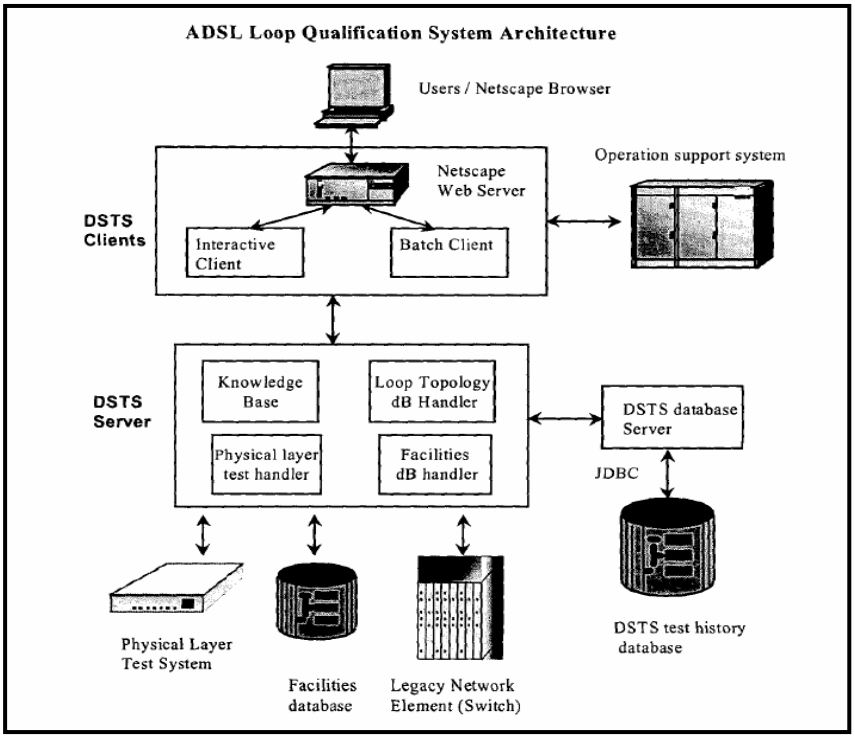


Figure 1.3 : Architecture physique du *DSTS* [28]

Comme on peut le voir sur la Figure 1.3 : le premier « tier » se présente à l'utilisateur via un navigateur qui lui permet d'effectuer des requêtes directement (« Interactive Client ») ou de manière différée (« Batch Client »), le « tier » du milieu (il reçoit les requêtes des clients,

rassemble les informations provenant des systèmes externes, implémente la logique d'affaires et renvoie les résultats) est principalement composé de la base de connaissance et du moteur d'inférence, le troisième « tier » est formé par les bases de données des détails sur les lignes téléphoniques et les anciens systèmes fournissant également des données et offrant une capacité de stockage.

Un des avantages de cette découpe est que le client ne doit jamais communiquer avec le troisième « tier », il ne fait que recueillir, transmettre les requêtes des utilisateurs et afficher les résultats. La logique d'affaires et les systèmes externes leur sont totalement étrangers et inutiles. Ce type d'architecture permet plus de flexibilité et de « scalability ». En effet, le client, le serveur et la base de données peuvent se situer à des endroits géographiques différents et des modifications peuvent être effectuées de manières indépendantes.

Web-based expert system for food dryer selection (DrySES)

Haitham M.S. Lababidi et Christopher G.J. Baker ont travaillé sur le développement d'un système à base de connaissances (*DrySES*) ayant pour but de conseiller un utilisateur sur le type de machine de séchage de nourriture qu'il devait choisir en fonction de ses besoins [29]. En effet, chaque machine de ce type ne peut pas nécessairement traiter la même quantité en une même période et la façon dont le processus est exécuté a un impact assez important sur la qualité du produit final.

L'architecture logique est de type « blackboard ». Elle est dotée d'un mécanisme central qui permet la coordination des modules indépendants relatifs aux connaissances et la gestion du processus d'inférence. Comme on le voit dans la figure ci-dessous, l'architecture est composée de quatre composants principaux :

- « Knowledge Sources » : Connaissances qui permettent la résolution des problèmes de l'utilisateur.
- « Blackboard » : Moyen de communication entre les différentes sources de connaissances. Il a l'avantage d'intégrer divers composants de la base de connaissances.
- « Scheduler » : Il permet la communication entre les différents constituants du système et guide les raisonnements de celui-ci. D'une part, il récupère les données des applications, les enregistre sous forme de données, de faits ou d'évènements ou répond aux demandes de données des applications. D'autre part, il sélectionne les évènements enregistrés qui proviennent du « Blackboard » et les exécute.
- « User Interface »

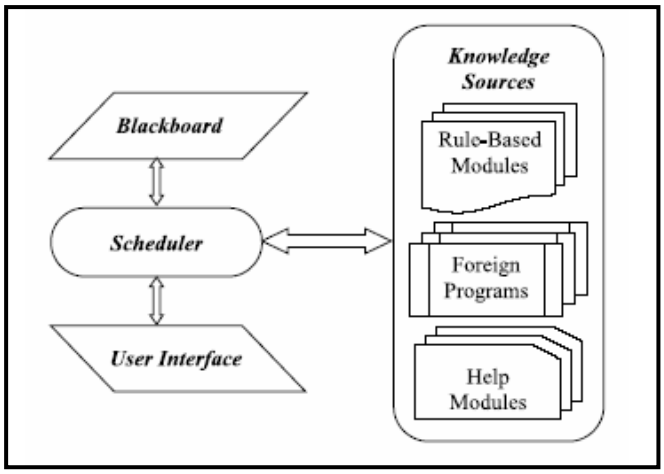


Figure 1.4 : Architecture logique du DrySES [29]

Quant à l'architecture physique, Haitham M.S. Lababidi et Christopher G.J. Baker disent que [29]:

> "DrySES has been implemented as a three-tier distributed application (Deitel & Deitel, 1999). Such applications consist of a user interface, business logic and database access. This approach has been commonly used in developing web-based database applications. In the present study, it has been modified to accommodate the knowledge sources used by DrySES. In this respect, the scheduler is the middle tier (business logic), while the knowledge sources and the blackboard represent the third tier (database access)."

Pour ce qui est de l'implémentation de cette architecture, le choix du langage de programmation s'est porté sur Java et sur des technologies liées au web [29]:

> "Java was used as the main programming language for this purpose and to dynamically create the HTML forms required by the GUI. Java servlets and other Java and Internet tools were successfully employed to integrate the diverse applications on the web in a seamless, user-friendly environment."

Fish-Expert

Daoliang Li, Zetian Fu et Yanqing Duan ont travaillé sur un système expert « web-based » (*Fish-Expert*) permettant d'aider les pisciculteurs à identifier les maladies dont leurs poissons sont affectés []. Ils expliquent le contexte technologique dans lequel est né *Fish-Expert* [30]:

> "With the rapid development of the Internet, more web-based ESs are beginning to emerge. Unfortunately, WWW was originally conceived simply as a document distribution infrastructure (Riva et al., 1998) and was not created with applications such as expert systems in mind (Huntington, 2000). Any attempt to use it for distributing expert systems must cope with certain difficulties (Huntington, 2000; Riva et al., 1998). Until recently, there seems to be a lack of easy-to-use supporting tools for developing web-based ESs. At the time the Fish-Expert system was developed, no suitable tool for web-based ESs was available. Thus, Fish-Expert was developed using a mixture of Internet techniques and SQL programming languages. DHTML (Dynamic Hypertext Markup Language), Java Script, Java, VB script and ASP (Active Server Page) were used in the programming."

Pour ce qui est de l'architecture physique, ils ont adopté une architecture à trois couches de type Client/Serveur/Serveur.

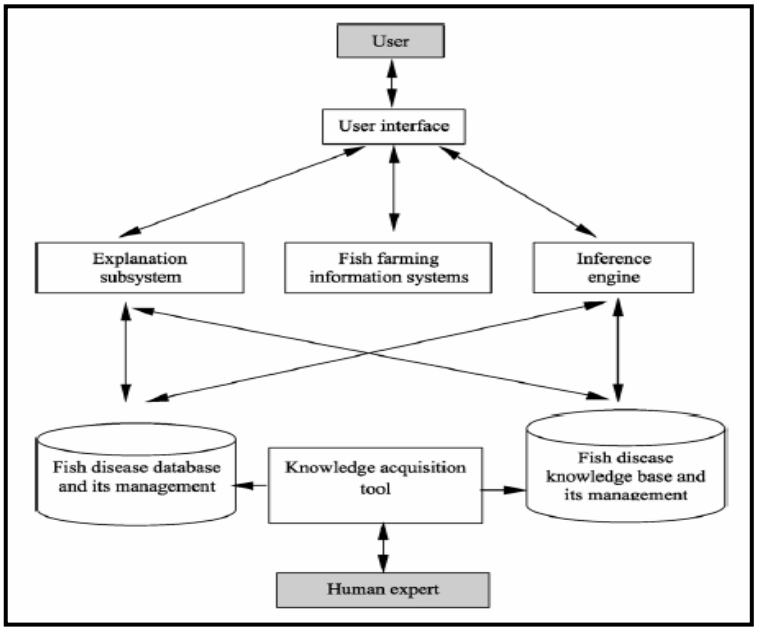


Figure 1.5 : Architecture physique de *Fish-Expert* [30]

The medical knowledge acquisition system

Un des aspects les plus importants des systèmes experts est la possibilité de pouvoir gérer sa base de connaissances. En d'autres termes, il doit être possible d'ajouter, supprimer et modifier des connaissances en fonction des avancées dans le domaine d'application. Hongmei Yan, Yingtao Jiang, Jun Zheng, Bingmei Fu, Shouzhong Xiao et Chenglin Peng ont travaillé à la création d'un système « web-based » permettant la gestion d'une base de connaissances médicales [31]. Cela permet de centraliser les connaissances de milliers de praticiens, dans l'idée de créer des systèmes experts de qualité.

L'architecture physique de ce système est désignée comme ceci [31]:

> "The medical KA system is built upon a distributed client/server architecture consisting of three tiers (Figure ci-dessous): (i) server tier, (ii) middle tier, and (iii) client tier. The three-tier design supplies a tier of distribution separate from the front-end presentation and the back-end data access. Such arrangement has many advantages over traditional two-tier or single-tier designs in terms of ease of load balancing, performance, flexibility, robustness, and scalability."

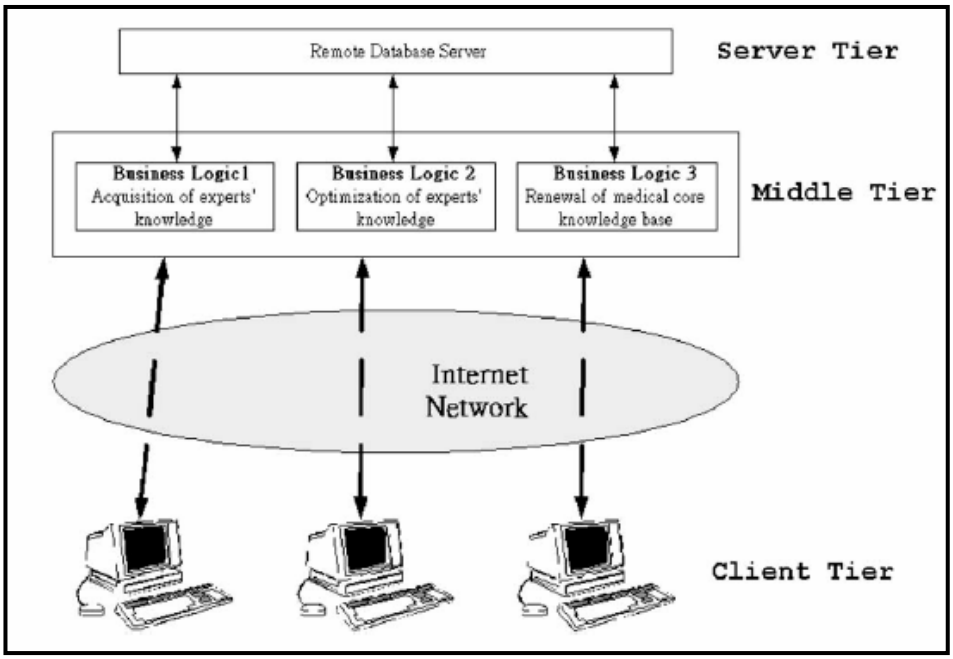


Figure 1.6 : Architecture physique du *medical KA system* [31]

Le « Server Tier » traite les requêtes des clients (entrées ou sorties), gère la sécurité lors de l'accès aux bases de données et à ses informations, contrôle le flux correspondant au nombre d'utilisateurs connectés ainsi qu'aux échanges de données.

Le « Middle Tier » gère les droits d'accès des clients aux bases de données, le « load balancing » des requêtes clientes, la sécurité (liées à des données utilisateur comme son niveau professionnel), ainsi que l'intégrité et la validité des informations médicales introduites. Il constitue en fait un tampon entre le client et le serveur.

Le « Client Tier » correspond à l'interface présentée aux experts médicaux pour accéder à la base de données et permettre la manipulation des informations qui y sont stockées.

Conclusion

Il existe de plus en plus de systèmes à base de connaissances possédant une interface Web. Selon Alan J. Thomson et Ian Willoughby, ce choix permet d'effectuer des corrections et des ajouts dans la base de connaissances sans qu'il y ait à modifier quoi que ce soit au niveau de l'utilisateur [32]. Cela facilite également l'accroissement du nombre d'ordinateurs qui peuvent avoir accès à un système expert, quelque soit leur localisation géographique.

Suite aux exemples précédents de systèmes à base de connaissances « web-based », nous pouvons remarquer que le fait d'avoir un logiciel accessible via Internet influe sur l'architecture physique et, de temps à autres, logique. En effet, l'architecture physique est généralement du type « multi-tier », ce qui permet de séparer la partie cliente, les serveurs et les systèmes de stockage de données (bases de données, …). Pour ce qui est de l'architecture logique, la tendance est plutôt à la séparation entre la vue (grâce au navigateur), le contrôleur qui dirige les requêtes des clients et le modèle qui implémente la logique d'affaires et qui se trouve sur le deuxième « tier » composé par les serveurs.

Nous pouvons conclure que COSMICXpert fait effectivement partie de ces systèmes à base de connaissances « web-based ». Il sera exposé en détails dans le Chapitre 2 de cet ouvrage.

1.3.1.4 Adaptabilité de l'architecture d'un KBS à un autre domaine de connaissances

L'existence de systèmes experts de type « shell », qu'il faut configurer pour obtenir un système à base de connaissances fonctionnel, permet de dire qu'il est possible de réutiliser la même architecture pour des systèmes différents. Cependant, ces coquilles ne sont pas encore courantes pour les systèmes à base de connaissances « web-based ». De plus, Joel D. Riedesel nous dit qu'elles peuvent s'avérer restrictives car les créateurs de celles-ci tendent à privilégier leur rapidité au dépend de la représentation adéquate du domaine de connaissances [33].

Malgré l'existence des « shells », nous avons pu remarquer précédemment que la plupart des équipes de recherche ont développé l'entièreté de l'application. Cependant, certains chercheurs ont opté pour l'intégration dans leur système d'un moteur d'inférence préexistant comme, par exemple, Jess (Java Expert System Shell) [Web3]. Dans chacun des cas, cette démarche les a, malgré tout, amené à se poser des questions sur l'architecture logique et physique la plus adéquate.

1.3.2 Domaine des bases de connaissances

1.3.2.1 Base de connaissances (KB)

Les systèmes à base de connaissances sont parfois synonymes de systèmes experts, tel que déjà mentionnée par Prince [14]. Un tel système s'articule autour de ladite base de connaissances, manipulée afin d'en extraire une connaissance adaptée et utile à l'utilisateur. Citons à cet égard [34]:

> "Expert systems consist of mainly two parts. One is the problem-solving part, which utilizes human expert knowledge. Knowledge is usually represented declaratively in task-specific forms, such as rules, decision trees, decision tables and object relation graphs. The other part combines conventional functions, such as main flow description, database query and graphical user interface. This part is usually represented procedurally."

L’ontologie d’un domaine est indissociable de la base de connaissances. La défintion de l’ontologie est la suivante, pour Guarino [35]:

> “An ontology is a logical theory accounted for the intended meaning of a formal vocabulary, i.e. its ontological commitment to a particular conceptualization of the world. ”

Ensuite, Guarino, cité par Brigitte Biébow and Sylvie Szulman [36], fournit également une définition du concept de base de connaissances [35] :

> “[…] Guarino’s definition of a knowledge base as being obtained by specialization of an ontology to a particular state of the world.”

La mise sur pied d’une base de connaissances est un processus évolutif qui mérite une attention particulière. Le caractère incrémental et évolutif de ce processus de conception apparaît plus clairement dans l’exemple de conception donné par Margaret A. Hahn, Richard N. Palmer, M. Steve Merrill et Andrew B. Lukas dans [37] :

> “The knowledge base of an expert system is developed through a process of ‘‘knowledge acquisition.’’ Knowledge was acquired and incorporated into [the system] by domain familiarization and interviewing sewer infrastructure experts, operators, and managers. The knowledge acquisition process was facilitated with a rapid prototyping process that entailed developing a simplified prototype of the tool after each information gathering session **(McGraw and Harbison-Briggs 1989)**. This facilitates rapid feedback of the domain experts’ suggestions and effective testing of the accuracy of the knowledge base.”

*1.3.2.2 Architecture de la base de connaissance*s

En dernier lieu, il nous faut expliciter les raisons qui nous ont poussé à conserver une vision dissociée et duale du système. Pour se faire, rappelons la citation Koseki et *al.* [34],

> "Expert systems consist of mainly two parts. One is the problem-solving part, which utilizes human expert knowledge. […] The other part combines conventional functions, such as main flow description, database query and graphical user interface."

Cette description d'un système à base de connaissances met en lumière la dichotomie sans laquelle notre tâche de restructuration n'aurait pas été réalisable. En effet, il est possible de traiter les deux parties du système de façon indépendante, permettant ainsi d'agir sur l'un et l'autre parallèlement. Dans la suite, nous nous réfèrerons à ces deux parties de manière différente afin de mieux marquer cette dichotomie. Par convention, nous désignerons par « *moteur* » la partie fonctionnelle du système tandis que l'expression « *base de connaissances* » sera réservée à la partie traitée par le premier. Nous dénommerons *« système »* l'entièreté du logiciel, moteur et base de connaissances compris.

Cette simple convention permet de mieux exprimer notre vision du système avec le moteur comme outil de traitement et la base de connaissances comme greffon interchangeable. La possibilité s'offre alors à nous de modifier l'architecture du moteur tandis qu'une nouvelle base de connaissances, à greffer ultérieurement, peut être mise sur pied par les experts du domaine vers lequel nous devrons migrer. Cette vision des choses est encore renforcée par le schéma (cf. Figure 1.7) d'architecture proposé dans [34].

Dans le cadre de COSMICXpert, l'architecture de la base de connaissances du domaine de la mesure a été définie et présentée par J.-M. Desharnais, A. Abran, A. Mayers, L. Buglione et V. Bevo dans l'article « Knowledge Modeling for the Design of a KBS in the Functional Size Measurement Domain » [38].

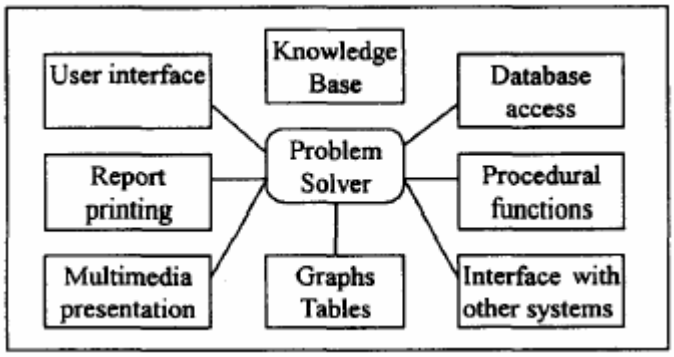


Figure 1.7 : Architecture générale des composants d'un KBS hybride

1.3.2.3 Adaptabilité de l'architecture d'une KB à un autre domaine de connaissances

Les documents en rapport à l'adaptabilité de l'architecture d'une base de connaissances sont rares. Cependant, il existe deux principales sources d'informations à ce sujet :

- la méthodologie CommonKADS décrite dans [70] et dans [71].
- van Heijst [39] car un de ses objectifs est la réutilisation des différentes théories des ontologies.

A ce sujet, ce dernier aborde la réutilisation des concepts entre différents domaines [39]:

> "Reuse of domain-specific concepts across domains. At first glance, the reuse of domain-specific concepts across domains seems a contradiction in terms. However, domain-specificity is not a dichotomy: some concepts are obviously more domain specific than are others. […] This observation can be used to organize the library in such a way that more reusable concepts are put in other theories than less reusable concepts. To do this, the notion of domain specificity must be operationalized. One candidate for this operationalization is the notion of abstraction level: definitions that specify less detail are often less domain specific than definitions that specify more detail."

Cependant, Van Heijst précise qu'il existe des problèmes en rapport à cette réutilisation [39]:

"Firstly, the relation between more abstract and less abstract definitions is a many to many relation. A concept which is specified in detail can have multiple abstractions, depending on the point of view that one takes. This makes it difficult to specify the inclusion relations between theories which contain detailed definitions and theories which contain abstract definitions. (…) A second problem with an organization according to the level of abstraction is that this dimension does not discriminate between concepts on the same level of abstraction".

1. 4 Problématique

Comme nous l'avons vu dans la revue de littérature, la qualité du logiciel prend différents aspects et est fortement influencée par l'architecture, tant logique que physique. La problématique étudiée dans ce document étant celle de *la qualité du logiciel* (domaine vaste et diversifié concernant tout acteur du génie logiciel), cette constatation est très utile. Elle est vérifiée à condition que la documentation contienne le *design* de l'architecture, exposé *de manière explicite et sans incompréhension*.

La nécessité d'une architecture rigoureuse venant supporter le développement d'une application est un sujet auquel les professeurs nous ayant dispensé leur savoir n'ont cessé de nous sensibiliser durant notre apprentissage. Sur ce même point, la revue de littérature nous apprend qu'elle reflète les décisions de *design* prises, ce qui la rend incontournable. Face à l'omniprésence de cette contrainte, le besoin de traiter cet aspect du développement d'un logiciel s'est imposé d'emblée. En effet, la migration de la base de connaissance de **COSMICXpert Web Edition** (projet qui nous a été confié dans le cadre de notre stage à l'Ecole de Technologie Supérieure à Montréal) s'est rapidement révélée un terrain adéquat pour une étude plus approfondie des méthodes de gestion de la qualité du logiciel. C'est avec ce projet qu'est apparue l'opportunité de traiter sous un angle différent les problèmes de maintenance, *design* d'architecture, phase de test d'un système à base de connaissances et qualité du code.

D'un point de vue conceptuel, une problématique majeure de ce mémoire est de déterminer si la migration d'un système à base connaissance vers un autre domaine que celui pour lequel il a été conçu peut s'effectuer harmonieusement et avec facilité. Cette migration concerne autant l'architecture du logiciel que l'architecture de la base de connaissances, deux sujet ayant été abordés dans la revue de littérature (cf. adaptabilité). En effet, la tâche qui nous a été confiée était de développer un prototype de système expert dont la base de connaissance concerne la maintenance, en réutilisant l'architecture de **COSMICXpert.**

Les modèles issus de la littérature indiquent que la mise en œuvre de ce passage de **COSMICXpert à SMXpert** suppose quelques questions :

- ✓ Le nouveau domaine de connaissance peut-il être mis sous forme d'une ontologie cohérente par rapport à celle d'origine ? En d'autres termes, les concepts manipulés dans le nouveau domaine peuvent-ils aisément être ramenés à des concepts semblables inhérents au domaine d'origine ? Cette question est la clé de voûte de tout raisonnement à venir.
- ✓ L'application originelle présente-t-elle toutes les fonctionnalités nécessaires au traitement de la base connaissance nouvelle ? Plus clairement, la migration de la base de connaissance est-elle juste un remplacement des fichiers adéquats ou nécessite-t-elle vraiment une restructuration du système qui la traitera ?
- ✓ L'étendue du nouveau domaine de connaissance entre-t-elle en compte ? Concrètement, les performances de l'application d'origine sont-elles suffisantes dans le cas où le nouveau domaine – et par-là, la nouvelle base de connaissances – serait plus vaste ? L'application originelle n'est-elle pas sous-exploitée si le nouveau domaine est plus restreint ?
- ✓ La manipulation des connaissances, telle que pensée dans le système d'origine, est-elle adaptée à la connaissance du nouveau domaine ? Les interfaces sont-elles en adéquation avec les besoins des utilisateurs coutumiers du nouveau domaine ?
- ✓ Le passage d'un domaine à l'autre peut-il dégager des principes généraux applicables à toute forme de migration future et à n'importe quel domaine de connaissances ?

Ces différentes réflexions forment la base des raisonnements suivis lors de l'élaboration de **SMXpert Web Edition** et constitue en quelque sorte un fil conducteur du présent document.

D'un point de vue purement technologique, **COSMICXpert Web Edition** offre une intéressante diversité. En effet, l'on voit cohabiter au sein du système, pas moins de quatre langages différents (*HTML, XSL, Javascript, JAVA,…*).

Cette particularité de l'application engendre les problèmes suivants :

- ✓ La validation automatique du code avec des outils tels que ***IBM Rational Sitecheck*** est éminemment difficile à mettre en œuvre. En effet, les logiciels de ce type – pensés pour une technologie donnée – ne prennent que peu ou pas en compte la variété en question. Par exemple, si ***Sitecheck*** prend en compte les liens *hypertextes* dans son calcul de dépendances, il ne tiendra pas compte des liens implicites lovés au sein du code *Javascript*.
- ✓ La communication entre les technologies peut poser problème lorsqu'il s'agit de manipuler des données communes. On observe alors la nécessité de recourir à des *« bricolages »* peu esthétiques, qui rendent le système moins efficient, voire plus sensible à des brèches de sécurité.
- ✓ La diversité, au niveau du code source, rend sa maintenance et son appropriation plus complexe. Ce problème est particulièrement marqué du côté *client* de l'application, c'est-à-dire dans le code des pages Web de l'interface utilisateur.

Cette hétérogénéité technique se justifie cependant pleinement, chaque langage utilisé apportant une spécificité nécessaire à l'application. Mais il serait bénéfique de revoir leur utilisation de manière plus spécialisée, évitant ainsi une trop grande mixité au sein du code.

Notons également que **COSMICXpert** est le fruit d'héritages successifs et que ce relais de développeur en développeur accentue cette hétérogénéité du code. Non seulement, de nouvelles technologies ont été ajoutées depuis la version originelle mais l'hétérogénéité se marque aussi au niveau des *astuces* algorithmiques utilisées. De ce fait, il n'est pas rare de voir deux tâches identiques implémentées de façons différentes, en deux endroits distincts du code.

Au niveau de la qualité du logiciel, la problématique dégagée par nos travaux s'exprime à plusieurs degrés. D'une part, la question se pose de savoir si un projet passé de main en main perd automatiquement en qualité ou si les générations successives tendent à l'améliorer. Cet aspect est intimement lié à la phase de test ayant, comme nous le verrons, marqué le début de nos travaux. D'autre part se dégage l'intéressante question de l'amélioration de la qualité du logiciel à travers sa restructuration. En effet, le passage d'un système vers son successeur n'offre-t-il pas les étapes nécessaires à un contrôle et une amélioration de sa qualité ?

1.5 Objectifs

Dans ce mémoire nous faisons l'hypothèse que la phase de restructuration d'un système à base de connaissances s'inscrit dans le cadre de la gestion de la qualité du logiciel et que de ce fait les analyses nécessaires au passage à un nouveau système à partir d'un système existant, comprend toutes les phases de la maintenance. C'est ce que nous allons faire avec la restructuration de **COSMICXpert Web Edition**[2].

En terme d'objectifs, cette démarche permet d'améliorer la qualité des mécanismes de traitement des connaissances, de la prise en compte des exigences et du code. Cette amélioration reste valable aussi bien pour le logiciel à base de connaissances actuel (**COSMICXpert WEB Edition**) que pour le nouveau (**SMXpert Web Edition**).

Plus concrètement, nous visons l'accomplissement des objectifs suivant :

- ✓ Fournir un système à base de connaissances (**SMXpert Web Edition**) opérationnel et présentant les fonctionnalités spécifiques à la nouvelle base de connaissance.
- ✓ Améliorer la qualité de **COSMICXpert WEB Edition** au niveau des résultats attendus. Cet objectif concerne les exigences liées aux fonctionnalités du système. En clair, nous corrigerons les erreurs restantes afin d'obtenir un système le plus opérationnel possible.
- ✓ Fournir deux systèmes à base de connaissances plus performant. Cet objectif s'axe donc sur les exigences non fonctionnelles, les performances en terme de rapidité, sécurité, confort d'utilisation, etc.
- ✓ Présenter la démarche qui a été suivie pour réaliser ces améliorations et tenter d'en déduire une plus formelle et systématique, c'est-à-dire un cadre de référence. Celui-ci sera limité aux systèmes à base de connaissances « web-based ».

A contrario, toute la problématique entourant les interfaces *homme-machine* ne sera pas abordée dans ce travail. Ainsi, aucune considération esthétique ou ergonomique ne sera mise en oeuvre afin de modifier ou améliorer lesdites interfaces graphiques.

[2] Pour plus de détails sur l'application, le lecteur pourra consulter le Chapitre 2.

Chapitre 2 : De COSMICXpert à SMXpert.

Comme nous l'avons déjà évoqué dans le chapitre précédent, modifier le domaine de connaissances de **COSMICXpert Web Edition** afin de créer un nouveau système dédié au domaine de la maintenance était notre objectif premier. Le résultat de ce travail est le logiciel à base de connaissances **SMXpert Web Edition**. Il est actuellement à l'essai sur une base de connaissances réduite à quelques concepts, elle-même en attente d'experts prêts à l'étoffer. Dans ce chapitre, nous allons évoquer plus en détail l'un et l'autre logiciel, afin de mettre en exergue les différences fondamentales avec lesquelles nous avons du composer.

2.1 Présentation générale des logiciels

2.1.1 COSMICXpert Web Edition

Ce logiciel est à la base de notre démarche de restructuration. La compréhension de ses mécanismes et de son fonctionnement était une condition préalable à toute autre tâche. A cet égard, notre premier travail aura été de prendre en main le système, de nous familiariser avec lui tant du point de vue utilisateur que technique. C'est par le biais d'une phase de tests et de correction de ce dernier que nous aurons pu acquérir la compréhension nécessaire. Nous verrons l'importance de cette première étape dans les réflexions menées par la suite (cf. Chapitres 3, 5 et 6). Afin d'épargner au lecteur ce long travail de familiarisation et de rendre la compréhension des développements ultérieurs plus aisée, nous commencerons par décrire sommairement les divers aspects du logiciel (cf. Section 2.2 et 2.3). Pour une explication exhaustive et détaillée du fonctionnement de **COSMICXpert Web Editon**, le lecteur pourra se reporter à [15], [2] et [3].

2.1.2 SMXpert Web Edition

SMXpert[3] Web Edition est l'aboutissement de notre travail de restructuration sur COSMICXpert. Comme précisé plus haut, ce logiciel est actuellement fonctionnel et à l'essai sur une base de connaissances réduite à quelques concepts aisés à manipuler. J.-M. Desharnais et A. April en parlent dans [5] et le présentent comme un outil de diagnostique:

SMXpert a bénéficié de deux traitements distincts. D'une part, son domaine de connaissances a été changé par rapport à celui de COSMICXpert. Il s'applique désormais à l'évaluation de la maturité des processus de maintenance de logiciel. Cette partie de la restructuration constituait notre objectif premier. D'autre part, notre logiciel a bénéficié d'une restructuration de son architecture. Celle-ci a été repensée et adaptée aux nouveaux besoins accompagnant le changement de base de connaissances. De même, SMXpert se veut une version apportant une amélioration de performances comparativement à COSMICXpert. Toutefois, nous verrons par la suite qu'il est possible d'espérer améliorer la qualité du logiciel originel également, au travers de cette démarche de restructuration. Les Sections 2.2 et 2.3 décrivent l'application dans sa globalité.

2.2 Domaines de connaissances et ontologies

La présentation de l'ontologie de la connaissance traitée par un système à base de connaissances est une phase indispensable lors de son développement. En effet, une telle application structure celui-ci pour pouvoir raisonner dessus.

2.2.1 COSMICXpert

Comme l'explique le Chapitre 1, un des objectifs premiers de **COSMICXpert Web Editon** est d'offrir un système d'apprentissage de la méthode de mesure logicielle *COSMIC-FFP*. De ce fait, la base de connaissances du logiciel repose sur les liens tissés entre une ontologie du génie logiciel et une ontologie de la mesure logicielle *COSMIC-FFP*.

[3] SM est en fait un acronyme pour Software Maintenance

L'ontologie choisie pour le génie logiciel se devait de rassembler autour d'elle le plus grand consensus possible. Les définitions et structures retenues à cet effet pour les concepts propres au génie logiciel sont issues du SWEBOK (Software Engineering Body of Knowledge) [40].

La référence retenue pour tous les concepts propres à la mesure logicielle est évidemment le manuel de mesure (dans sa version 2.2) de *COSMIC-FFP* [68]. Cette méthode de mesure permet d'estimer la taille fonctionnelle d'un logiciel. Elle s'appuie sur la méthode des Points de Fonctions et possède, à ce titre, de nombreuses parentes. Depuis sa création, l'utilisation des Points de Fonctions a été généralisée à de multiples méthodes de mesures, l'étendant et lui apportant leurs caractéristiques propres. Sans prétendre être exhaustif, citons parmi ces 'cousines' des méthodes telles que *Function Point Analysis* [41], *MKII Function Points* [42], *Feature Points* [43] ou *3D Function Points* [44].

Comme bien des méthodes étendant les Points de Fonctions, *COSMIC-FFP* trouve son origine dans la nécessité de pallier aux nombreux manques de ces derniers. Elle voit le jour en septembre 1997 dans les laboratoires de génie du logiciel et de métrique de l'UQAM patronné par le Laboratoire de Métriques Appliquées en Gestion Du Logiciel (LMAGL inc.). Tout en conservant les spécificités des Points de Fonctions traditionnels, la méthode présente une variante de leur utilisation corrigeant nombre de lacunes et s'adaptant particulièrement au monde de l'industrie. Respectant également les principes du Common Software Measurment International Consortium, la méthode est, depuis 2002, une norme ISO (19761) [1].

Concrètement, *COSMIC-FFP* présente tous les aspects propres à une méthode de mesure. En guise de résultat à une série de règles et procédures appliquées au logiciel mesuré, la méthode fournit une valeur représentant sa taille fonctionnelle. L'objet de la mesure est considéré de son point de vue utilisateur et la partie technique du système n'est donc pas visée. Une autre caractéristique de la méthode est son champ d'application [69]. En effet, *COSMIC-FFP* possède un large champ d'application englobant :

- le domaine des logiciels d'affaires, tels que les applications bancaires, de production ou de facturation, *et cetera*
- le domaine des logiciels en temps réel conçus pour garder le contrôle des évènements du monde réel, tels que les applications liées aux télécoms, à l'aviation ou encore les systèmes d'exploitation.

- Les logiciels tenant des deux catégories précédentes, tels que les logiciels de réservations en temps réel d'avions et d'hôtels.

De plus, la méthode s'applique indépendamment des décisions d'implantation du logiciel mesuré, suivant plusieurs points de vue différents et à toutes les phases de construction du logiciel, contrairement aux mesures techniques qui ne peuvent être appliquées que lorsque le code est réalisé.

Toutefois, la méthode de mesure *COSMIC-FFP* possède quelques limitations. Comme la plupart des mesures reposant sur les Points de Fonctions, elle n'est pas encore entièrement au point pour tenir compte de la taille fonctionnelle des logiciels caractérisés par un grand nombre d'algorithmes mathématiques et autres règles complexes, ou spécialisées. Ainsi n'est-elle pas adaptée aux systèmes experts, aux logiciels de simulation, et autres. De même, les logiciels traitant des variables de processus continus, tels les sons ou les vidéos, sortent de son domaine d'application.

Parallèlement aux règles et procédures applicables, *COSMIC-FFP* est définie par un ensemble de concepts et de définitions représentant un vocabulaire qui lui est propre. Le lecteur pourra se référer à [45] afin d'obtenir une définition exhaustive de ce vocabulaire. Les concepts n'étant pas isolés, il existe des relations entre les différents concepts.

Ci-dessous, la Figure 2.1 (cf. [15]) illustre l'ensemble des concepts présents dans COSMIC-FFP et des liens entre eux. Dés lors, les concepts de mesure logicielle utilisés au sein de COSMICXpert sont ceux que l'on peut retrouver dans le manuel de *COSMIC-FFP*. Il constitue à ce titre la référence pour l'ontologie de la mesure logicielle.

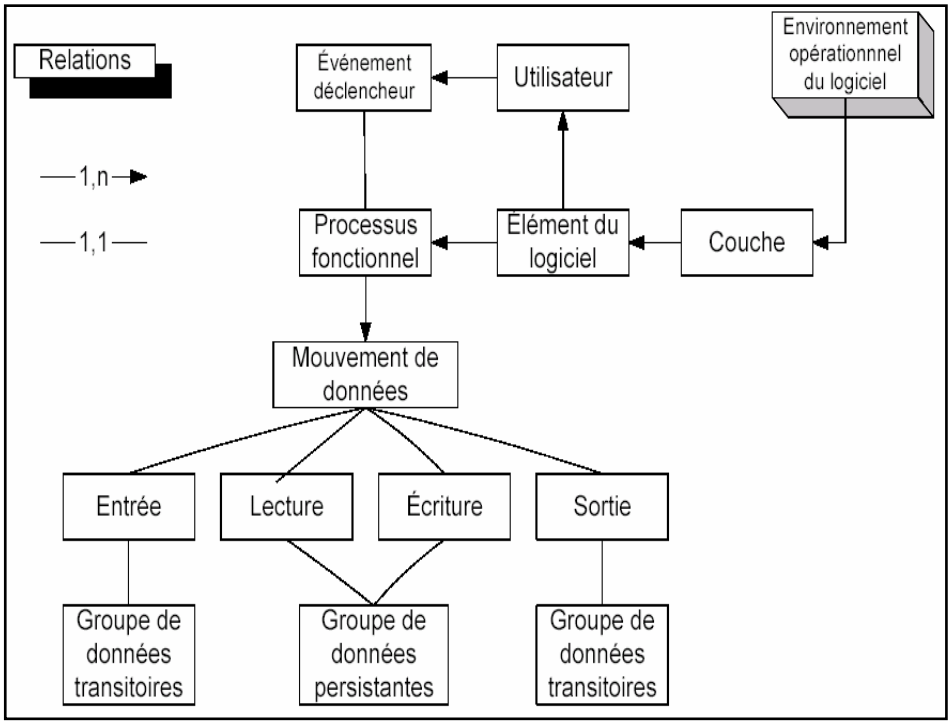


Figure 2.1 : Ontologie de COSMIC-FFP [15]

2.2.2 SMXpert

SMXpert Web Edition a pour but premier d'aider un utilisateur à évaluer la maturité de processus de maintenance et de résoudre les problèmes liés à ceux-ci. De par l'étendue de ce domaine de connaissances, SMXpert s'adresse à un large panel d'utilisateurs. En effet, l'évaluateur pourra aussi bien être un utilisateur final souhaitant apprécier la qualité des processus de maintenance de son entreprise, qu'un développeur impliqué dans lesdits processus, et désirant résoudre un problème les concernant.

Concrètement, et comme le rappelle [5], SMXpert est un logiciel à base de connaissances destiné à supporter l'usage du Modèle d'Évaluation de la Capacité à Maintenir le Logiciel (**SM^{MM}**) décrit dans [49]. Ce modèle a été élaboré pour répondre aux questions concernant les insuffisances des modèles existants en matière de maintenance du logiciel, ainsi qu'aux besoins d'un modèle de l'amélioration spécifique à la maintenance du logiciel. Il hérite directement du CMMi pour les activités similaires à un processus de développement de logiciel. Il en adapte d'ailleurs plusieurs pratiques. A ce sujet, J.-M. Desharnais et A. April donnent plus d'informations dans [5]:

> "The **SM**$^{\mathbf{CMM}}$ is designed to complement the maturity model developed by the SEI of Carnegie Mellon University in Pittsburgh [2] by the addition of software maintenance specific practices. The architecture of the model locates the most fundamental practices at a lower level of maturity, however the most advanced practices are located at a higher level of maturity. An organization will typically mature from the lower to the higher level as it improves."

Le modèle **SM**$^{\mathbf{MM}}$ est structuré en niveaux [5]. Du plus élevé au plus bas, nous avons :

- *Process domains*: ce sont les connaissances pertinentes et principles pour un ingénieur de la maintenance. Il existe 4 domaines : *Process Management*, *Maintenance Request Management*, *Software Evolution Engineering*, *Support to Software Engineering Evolution*
- *Key Process Areas*
- Roadmaps
- Best Practices: elles sont nombreuses et centrées selon un point de vue unique et sur les activités spécifiques de la maintenance du logiciel. A titre d'exemple, ce modèle peut être utile pour évaluer un fournisseur en maintenance du logiciel.

Pour plus d'informations sur la structure, le lecteur peut consulter [59] et [60]. Cependant, pour avoir une idée de la grandeur du modèle, il faut se rendre compte qu'il comprend 4 domaines, 18 zones de processus clés, 74 plans de route, 443 meilleures pratiques [5].

SMXpert adapte le raisonnement d'un expert utilisant le **SM**$^{\mathbf{MM}}$ et, par conséquent, son domaine de connaissances repose sur l'ontologie de celui-ci. J.-M. Desharnais et A. April vont dans ce sens également dans [5]:

> "The ontology represents the Domains, Key Process Areas and Roadmaps suggested by the SMCMM to describe the software maintainers' body of knowledge. Based on this ontology, it is possible to analyze the tasks required to build a KBS to support the maintainers in their quest for best practices."

L'ontologie de la maintenance a été définie par Kitchenham [56] et Ruiz [57]. De taille très importante, seule une partie de celle-ci a été prise en compte lors du développement du système à base de connaissances [55]. Elle reprend la plupart des concepts impliqués dans les questions en rapport au premier problème de Dekleva sur la gestion des demandes rapides de maintenance [58]. Le fait d'avoir choisi ce sous-ensemble est compréhensible car la maintenance est fortement basée sur les évènements [55]. En effet, certaines activités de celle-ci peuvent interrompre le travail habituel. La représentation de cette sous partie de l'ontologie est illustrée à la Figure 2.2 [56].

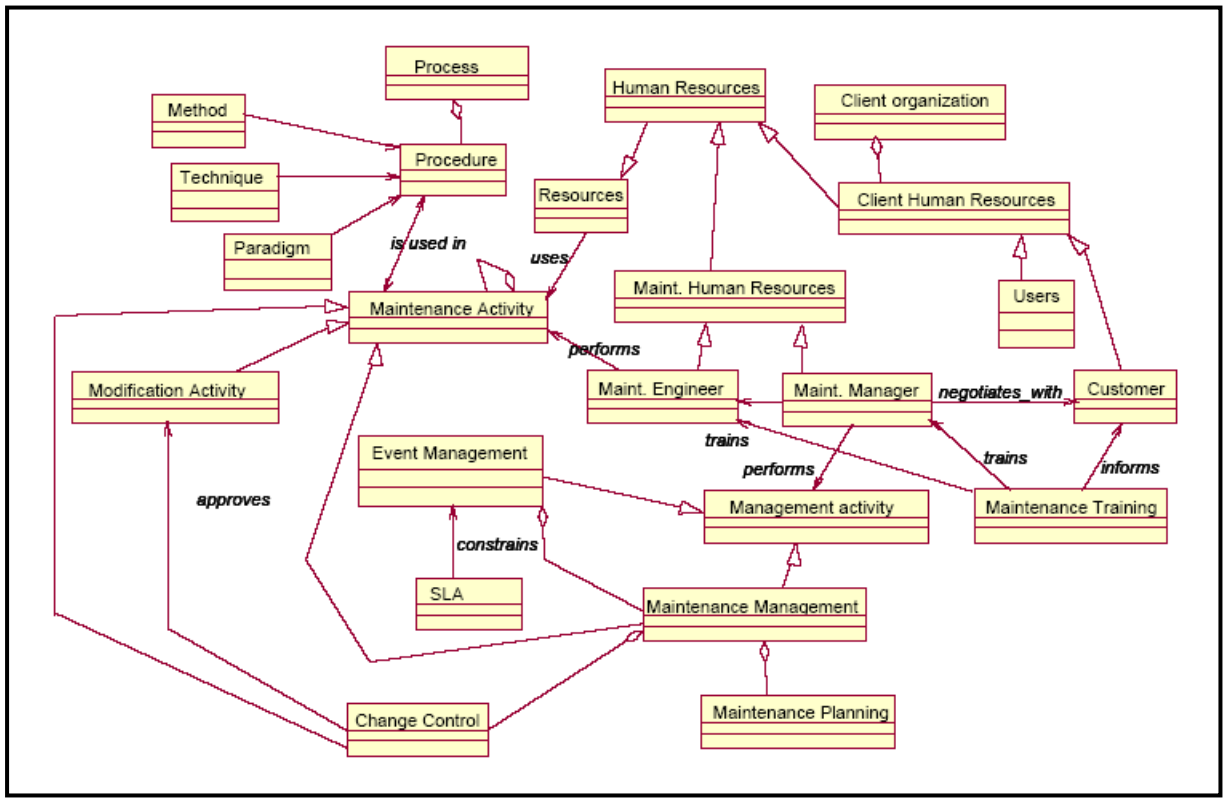


Figure 2.2 : Sous-ensemble de l'ontologie de la maintenance logicielle

A. April, J.-M. Desharnais et R. Dumke [55] indiquent que le choix d'une ontologie réduite de la maintenance est lié à un but pédagogique. En effet, celui-ci est de montrer l'utilité réelle d'un système à base de connaissances dans le domaine de la maintenance logicielle.

2.2.3 Obervations

Nous pouvons remarquer immédiatement que :

- Les deux domaines de connaissances sont différents par la taille. L'ontologie de la maintenance est beaucoup plus vaste dans ses concepts que celle de COSMIC-FFP. La Figure 2.1 représente l'entièreté de l'ontologie (COSMIC-FFP) tandis que la Figure 2.2 n'illustre qu'un sous-ensemble de l'ontologie de la maintenance.
- Les deux domaines de connaissances sont différents par le contenu. La mesure se concentre sur les produits des différentes phases du cycle de vie de développement de logiciel. Quant à la maintenance logicielle, elle se focalise sur les processus du même nom (et dans ce cas-ci, plus particulièrement sur les processus de gestion des demandes rapides de maintenance).

Du fait des remarques précédentes, nous pouvons conclure que les ontologies présentées ne se rapprochent aucunement. A ce stade, rien n'indique que l'objectif d'adaptation du logiciel au nouveau domaine sera possible. Nous devons encore analyser sous quelles formes vont se matérialiser les ontologies dans chacun des systèmes à base de connaissances.

2.3 Schémas conceptuels

Nous allons présenter les schémas conceptuels en rapport aux bases de connaissances des deux logiciels. Ils représentent les concepts qui permettent de rendre l'ontologie manipulable par l'application, ainsi que leurs relations. La base de connaissances étant centrale dans les systèmes à base de connaissances, ces concepts jouent un grand rôle dans son architecture et dans la possibilité du logiciel à effectuer des raisonnements sur les connaissances.

Le formalisme adopté pour les schémas est le formalisme "entité - association" (ERA) fréquemment utilisé dans la conception des bases de données. Nous en faisons une utilisation détournée dans la mesure où nous l’utilisons pour une autre finalité. Dans la pratique, la base de connaissances peut s'apparenter à une base de données car elle est le réceptacle d'une partie ou de l'ensemble des connaissances provenant de l'ontologie.

Chaque schéma ERA est complété d'un "dictionnaire des termes" donnant, pour chaque terme utilisé dans le schéma (et modélisé par une entité), une définition la plus précise et complète possible. Cela permet d'atteindre un plus haut niveau de précision qu'un simple graphique.

2.3.1 COSMICXpert

Sur base des concepts de l'ontologie de *COSMIC-FFP,* un schéma conceptuel propre à COSMICXpert peut être généré. Ce travail a été effectué dans [15] et [2] et les connaissances introduites ont fait le sujet d'une vérification et d'une validation [61]. Une série de concepts en forment le cœur et leur compréhension, par l'utilisateur du logiciel, est indispensable. Nous décrivons ci-dessous une liste de ces concepts, issue de [2], tandis que la Figure 2.2, tirée de la même source, fournit le schéma conceptuel, mettant en exergue les différents liens entre ceux-ci :

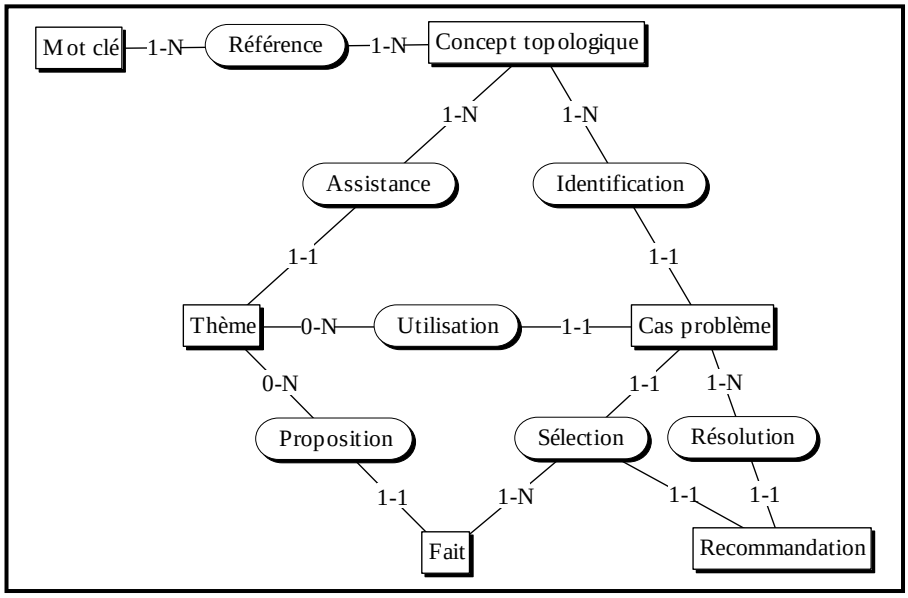


Figure 2.3: Schéma conceptuel de COSMICXpert

Les concepts topologiques sont des concepts qui servent à établir des liens entre les concepts de l'ingénierie du logiciel et ceux de *COSMIC-FFP*. Ils sont définis dans l'ontologie de COSMIC-FFP.

Un mot clé est n'importe quel mot du domaine du génie logiciel ou de COSMIC-FFP pouvant être associé à un concept topologique.

Un cas problème est un cas pour lequel le mesureur doit utiliser un processus de diagnostic pour identifier correctement un concept de l'ontologie COSMIC-FFP (ex. : identifier un groupe de données ou encore identifier un processus fonctionnel).

Les thèmes sont les pistes de réflexion à suivre par le mesureur pour trouver une solution aux cas problèmes.

Un fait est le constat du mesureur par rapport à un thème et au cas problèmes relié à ce thème.

Une recommandation est la proposition de solution fournit par le système ou des indications permettant au mesureur de raffiner sa solution (par exemple la référence d'un autre cas problème).

2.3.2 SMXpert

La première difficulté de la démarche de restructuration réside dans la définition d'un schéma conceptuel de SMXpert suffisamment proche de celui de COSMICXpert. Avec l'aide du Professeur April et de Mlle Fadila Oudjehane, nous avons pu dégager un nouveau schéma conceptuel. Celui-ci résulte du lien tissé entre l'ontologie du Modèle d'Évaluation de la Capacité à Maintenir le Logiciel et l'ontologie de la maintenance du logiciel. La Figure 2.4 illustre ce schéma et met en exergue les différents liens entre les concepts définis par la suite.

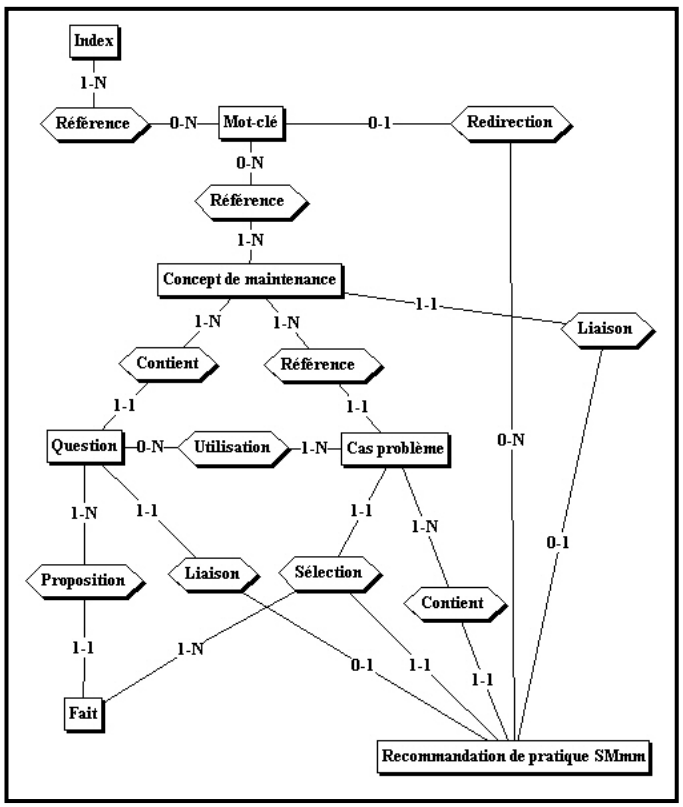


Figure 2.4 : Schéma conceptuel de SMXpert

Les concepts de maintenance sont des concepts inhérents à la problématique de la maintenance du logiciel. Ils établissent un lien entre les problèmes rencontrés par les utilisateurs ou développeurs et les connaissances fondamentales de la maintenance du logiciel.

Un mot-clé est n'importe quel mot du domaine de la maintenance du logiciel pouvant être associé à un concept de maintenance.

Un mot d'index est un terme du génie logiciel plus générique et donc moins précis qu'un mot-clé mais permettant d'orienter l'utilisateur vers le mot-clé adapté à sa question.

Un cas problème est un cas pour lequel l'évaluateur doit utiliser un processus de diagnostic afin d'identifier correctement une pratique de l'ontologie de **SM^{MM}**.

Les questions sont les pistes de réflexion à suivre par l'évaluateur pour trouver une solution aux cas problèmes.

Un fait est la réponse de l'évaluateur en rapport à une question et au cas problèmes relié à cette question.

Une recommandation est la proposition de solution fournie par le système ou des indications permettant à l'évaluateur de raffiner sa solution (par exemple la redirection vers un autre mot-clé).

Enfin, il est à noter que de plus fortes liaisons existent entre les recommandations de pratique et les concepts de maintenance, ainsi qu'avec les questions.

2.3.3 Observations

Contrairement aux ontologies des deux domaines, nous pouvons remarquer que les schémas conceptuels des deux logiciels sont assez semblables. C'est cette proximité qui rend notre démarche car les concepts du schéma sont au centre des logiciels. En effet, leur nature de système à base de connaissances fait qu'ils sont centrés sur cette même base.

2.4 Caractéristiques des logiciels

Comme pour une phase de spécification des exigences fonctionnelles, nous allons présenter les profils d'utilisateur interagissant avec le logiciel.

De plus, nous allons analyser les tâches qui structurent l'application. En effet, selon van Heijst [39], c'est la première activité se déroulant lors du développement d'un systèmes à base de connaissances. Nous les représenterons graphiquement. Le formalisme utilisé correspond à peu de choses près à un diagramme de flux sans l'être vraiment. Nous avons adopté ce formalisme pour rester cohérent avec les publications antérieures [2] [15].

2.4.1 COSMICXpert

D’un point vue utilisateur, **COSMICXpert Web Edition** présente trois modes d’interaction avec le système : le mesureur, l’expert et l’administrateur. Ces trois modes allouent des droits croissants à l’utilisateur.

- **Le mode administrateur :** permet à l’utilisateur travaillant dans celui-ci d’enregistrer, de supprimer et de modifier les droits d’un utilisateur du système. L’*administrateur* fixe les droits des autres utilisateurs, et donc les modes qui leur sont accessibles. A ce titre, il peut conférer à d’autres utilisateurs les mêmes droits que les siens. Il peut également modifier le mot de passe d’un utilisateur ainsi que la date d’expiration éventuelle de son enregistrement. De plus, l’*administrateur* dispose d’accès aux interfaces propres aux deux autres modes de fonctionnement.

- **Le mode expert :** permet à l’utilisateur travaillant dans celui-ci d’ajouter, supprimer ou modifier des connaissances à la base. Plus précisément, l’*expert* peut ajouter, supprimer ou modifier une donnée à la base de connaissances. Le type d’une de ces donnée doit appartenir à la liste de concepts vus *supra* et constituant l’ontologie du système. Il est à noter que l’application ne permet pas de modifier ou de supprimer des données de type **concept topologique**. De plus, l’*expert* dispose d’un accès à l’interface propre au mode mesureur.

- **Le mode mesureur :** permet à l’utilisateur travaillant dans celui-ci de consulter la base de connaissances afin de l’assister dans une tâche de mesure d’un logiciel. Ce mode constitue la finalité du système puisqu’il permet de mesurer un logiciel en suivant la méthode *COSMIC-FFP* et permet donc l’apprentissage de celle-ci par la pratique. Le *mesureur* dispose de l’accès à une seule interface : celle propre au mode mesureur. Cette interface permet de mener à bien les tâches successives du processus suivi par un mesureur afin de bien interpréter les règles de *COSMIC-FFP*. L’enchaînement de celles-ci est illustré dans [2]. Cependant, il correspond à la Figure 2.5 ci-après sans les éléments de couleur bleue et en remplaçant le terme « concept de maintenance » par celui de « concept topologique ».

Une caractéristique de COSMICXpert, mise en évidence dans la figure 2.5, est sa nature hybride [38]. En effet, cette figure illustre bien le fait que COSMICXpert repose à la fois sur des raisonnements par cas et des raisonnements basés sur des règles. Pour citer [2] :

> “La première partie du processus (figure avec fond blanc) définit une approche du type raisonnement par cas tandis que la seconde partie (figure avec fond gris) définit une approche ressemblant à un système basé sur des règles.”

Sans prétendre donner une explication exhaustive, nous citons ci-dessous les différences fondamentales entre ces deux formes de raisonnements. Le lecteur intéressé par l’approfondissement de la question pourra à nouveau se reporter à [15].

Dans [46], nous pouvons trouver une explication du raisonnement à base de cas :

> “Le raisonnement à base de cas permet de résoudre un nouveau problème en se souvenant d’un problème passé similaire et en réutilisant l’information et la connaissance de cette situation.”

Ce type de raisonnement s’applique particulièrement bien lorsque la connaissance traitée relève d’une catégorie imparfaitement cernée par l’application de règles. Il constitue en quelque sorte, un raisonnement bâti sur une bibliothèque de référence. A cet égard, selon [47], un système de raisonnement par cas est efficace dans un domaine de connaissances à la condition que certaines situations reviennent avec régularité dans ce domaine. COSMIC-FFP présente cette particularité. Il est possible de se référer à des cas problèmes déjà résolus pour en résoudre de nouveaux.

Le raisonnement à base de règles est d’un genre beaucoup plus systématique et inflexible. On trouve dans [15], cette définition :

> “***Le raisonnement à base de règles*** (SI…ALORS RÈGLE) permet entre autres l’abduction, c’est-à-dire que l’on peut partir d’un résultat pour en retrouver les causes. C’est une des plus anciennes techniques pour représenter le domaine de la connaissance dans un système expert. Il permet de traiter des problèmes précis et connus des experts. Les règles sont formalisées par des experts et peuvent servir à résoudre plus d’un type de problème.”

On comprend dés lors aisément que la complémentarité de ces deux modes de raisonnements permet de cerner des domaines de connaissances plus vastes et plus complexes.

2.4.2 SMXpert

La réalisation de **SMXpert Web Edition** a nécessité de s'adapter aux exigences propres au domaine de la maintenance. En effet, comme nous le voyons à la Section 2.2, le domaine de connaissances visé s'avère plus étendu que le domaine de la mesure. Ce changement suppose des adaptations au niveau de la gestion des données, et des performances résultantes. Une autre composante qu'il nous a fallu prendre en compte est l'interaction plus importante avec l'utilisateur final. En effet, l'utilisateur cherchant à évaluer la maturité d'un processus ou résoudre un problème lié à celui-ci se verra orienté de façons différentes suivant son profil et celui de son entreprise. Un critère à prendre en compte est donc la versatilité des réponses du système, en fonction de l'évaluateur.

D'un point vue utilisateur, **SMXpert Web Edition** présente trois modes d'interaction avec le système similaires à son prédécesseur : l'évaluateur, l'expert et l'administrateur. Ces trois modes présentent les mêmes caractéristiques que les modes équivalents de COSMICXpert à quelques exceptions près. Nous verrons dans la suite de ce document, dans quelle mesure ces objectifs ont pu être réalisés.

- **Le mode administrateur :** permet, outre les opérations vues *supra*, d'éditer un profil d'utilisateur. Ce dernier offre la possibilité à l'utilisateur d'aborder la base de connaissances suivant un point de vue adapté.
- **Le mode expert :** permet, outre les opérations vues *supra*, de calibrer les connaissances afin qu'elles soient présentées uniquement à un utilisateur dont le profil (en réalité celui de son entreprise) est adapté.
- **Le mode évaluateur :** permet toujours à l'utilisateur travaillant dans celui-ci de consulter la base de connaissances. Toutefois, ce mode l'assiste désormais dans une tâche d'évaluation de la maturité d'un processus. L'*évaluateur* dispose de l'accès à une seule interface : celle propre au mode évaluateur. Cette interface permet de mener à bien les tâches successives du processus suivi par un évaluateur afin de

bien interpréter les pratiques de **SM**$^{\mathbf{MM}}$. L'enchaînement de celles-ci est illustré dans la Figure 2.5. Il est à noter qu'un trait spécifique à SMXpert est le fait que cette interface soit sensitive au profil de l'utilisateur. Ainsi, les connaissances présentées le seront en fonction de la calibration citée *supra* et du profil de l'utilisateur.

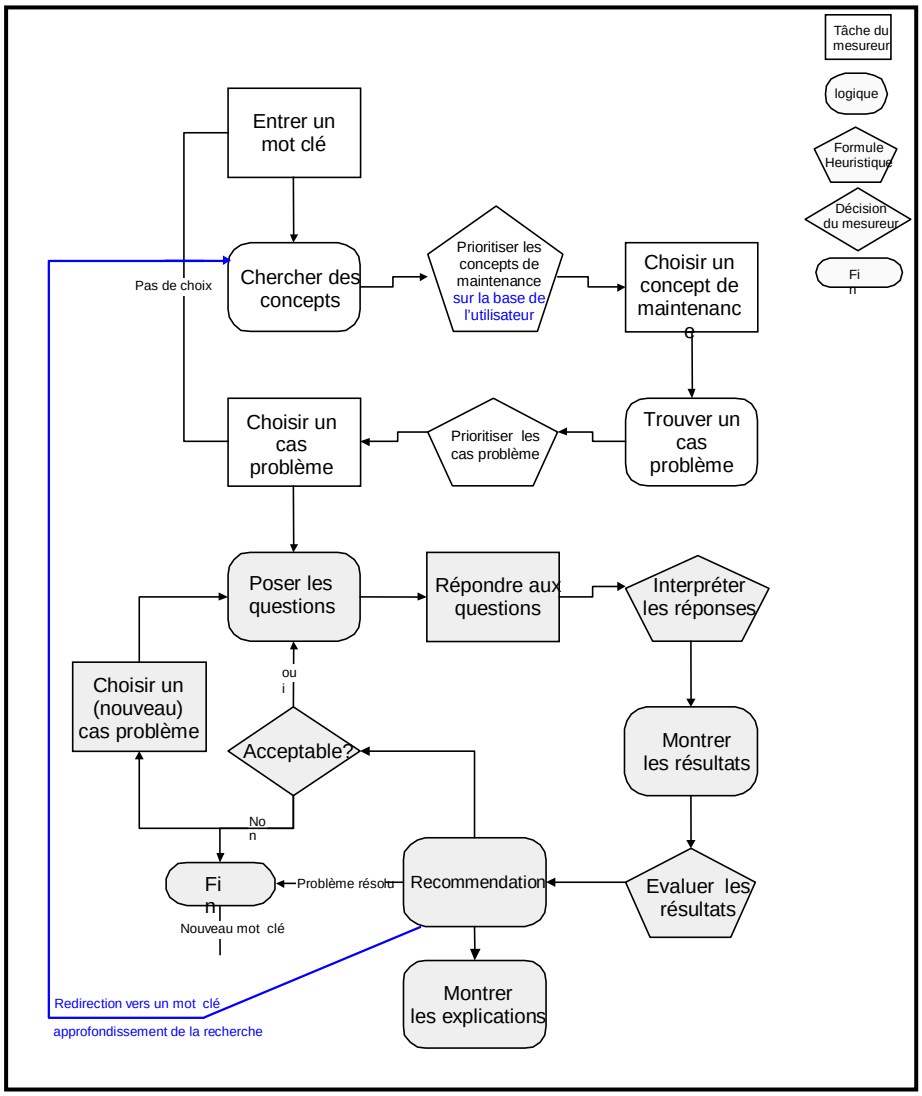


Figure 2.5 : Enchaînement des tâches de l'évaluation de la maturité dans SMXpert

Comme l'illustre également la Figure 2.5, SMXpert partage la même spécificité que son parent : il s'agit d'un système hybride [38]. Les premières étapes du raisonnement, jusqu'à l'affichage des questions, relèvent du raisonnement par cas. Tandis que le choix de la bonne recommandation sur bases des réponses aux questions dénote d'un raisonnement par application de règles.

2.4.3 Observations

La comparaison des enchaînements de tâches de COSMICXpert et SMXpert met en évidence leur similitude. Nous pouvons observer que les seuls détails supplémentaires sont un filtre et une possibilité de redirection (en bleu dans la figure 2.5).

Il est utile de remarquer que les tâches portent sur les concepts nécessaires au logiciel pour pouvoir raisonner sur l'ontologie. En effet, la section 2.3.3 nous a permis de conclure que ces concepts, utilisés par chaque logiciel, sont proches.

Ces constatations nous renforcent dans l'idée que le passage de COSMICXpert à SMXpert est possible sans trop d'efforts.

2.5 Caractéristiques techniques

2.5.1 COSMICXpert

D'un point de vue purement technique, COSMICXpert présente un certain nombre de caractéristiques notables apportant leurs avantages et inconvénients respectifs.

En premier lieu, **COSMICXpert Web Edition** est, comme son nom l'indique, un logiciel à base de connaissances *basé sur Internet*. Cette forme d'application présente l'avantage d'être facilement et largement accessible. Toutefois, elle soulève, de part cette large diffusion, la question cruciale de la sécurité. De même, la question du temps de réponse reste un point à ne négliger sous aucun prétexte. Enfin, le fait d'avoir une application WEB implique une mixité de langages et technologies qui peut, le cas échéant, amoindrir les performances du logiciel et, surtout, la facilité de maintenance.

Ce dernier point est fort marqué dans le prototype de **COSMICXpert Web Edition** sur lequel nous avons commencé notre travail. Le langage Java Server Page permet une utilisation plus diversifiée de technologies. En effet, les pages JSP alliant les possibilités de n'importe qu'elle page HTML à la puissance du langage Java, beaucoup de technologies peuvent cohabiter dans une même application. Dans notre cas, outre le langage JSP, il est fait usage de classes Java du côté serveur. Les interfaces, quant à elles, se constituent d'un

mélange de Javascript, XHTML et XSLT et de code Java inclus dans les pages elles-mêmes. Cette hétérogénéité ne joue évidemment pas en faveur d'une compréhension aisée du code et ne facilite donc pas la maintenance de celui-ci. Un autre composante qui vient rendre plus complexe encore cette compréhension est le fait que COSMICXpert ait été hérité par plusieurs générations de développeurs possédant chacun leur propre méthode de codage. Dés lors, même s'il était presque complètement fonctionnel au début de notre travail, « rentrer dans le code » s'est avéré une tâche parfois ardue.

C'est avec ces quelques faiblesses en tête que nous avons entamé notre travail de restructuration.

2.5.2 SMXpert

D'un point de vue strictement technique, un grand nombre de modifications ont été apportées par rapport à COSMICXpert. *Sensu lato*, SMXpert constitue le même type d'application. Il s'agit bien d'un logiciel à base de connaissances basé sur Internet. A ce titre, les remarques formulées préalablement pour COSMICXpert restent valables. Toutefois, SMXpert se distingue de son parent par une architecture physique fortement modifiée. Cette architecture nouvelle se veut plus adaptées aux inconvénients en question. Les problématiques du temps de réponse et de la sécurité y sont abordées de façon plus consciente.

De même, toujours sur base des remarques évoquées *supra*, SMXpert représente une tentative de gérer plus harmonieusement le mélange technologique. En effet, grâce au concours du *cadre de référence* Struts, un effort a été produit sur la séparation des technologies utilisées côté serveur et côté client.

2.6 Questionnement sur la démarche

La présentation des deux logiciels permet immédiatement de souligner certains aspects abordés dans la problématique, ainsi que dans les objectifs (cf. Chapitre 1). L'écart entre les deux systèmes nous oblige à nous questionner sur des points particuliers.

Tout d'abord, il nous faudra nous interroger sur la nature même du travail effectué. La première tâche sera de déterminer si les modifications permettant d'aboutir à SMXpert en partant de COSMICXpert relèvent du domaine de la restructuration ou de la ré ingénierie (cf. Chapitre 3). De même, il nous faudra définir si les ontologies respectives sont suffisamment proches que pour être modélisées au moyen des mêmes structures de données. L'exploitation de l'ontologie de SMXpert suppose un questionnement sur les mécanismes nécessaires à celle-ci. En effet, les concepts de cette ontologie pourraient ne pas être manipulables similairement à ceux de COSMICXpert. Cette dernière question renvoie également au problème de la mise en œuvre des nouvelles fonctionnalités exigées par SMXpert. En dernier lieu viendra le problème de l'étendue de la réutilisation des composants techniques existants.

Ensuite se posera la question de l'amélioration de la qualité. Nous devrons trouver des pistes indiquant que la restructuration d'un logiciel à base de connaissances est un moment adapté à l'amélioration de sa qualité. Le questionnement portera notamment sur le type d'améliorations que nous pouvons attendre, et l'importance de celles-ci. Nous examinerons, notamment, ce problème du point de vue de l'architecture du logiciel, de ses performances ainsi que de sa maintenabilité.

La dernière question de notre démarche sera de tenter de dégager des principes généraux permettant de traiter la restructuration des logiciels à base de connaissances. Nous pourrons notamment vérifier si les étapes suivies dans notre approche expérimentale sont suffisantes dans tous les contextes, ou si d'autres étapes seraient nécessaires afin de restructurer et améliorer tout logiciel à base de connaissances. Les résultats de cette interrogation pourraient alors s'inscrire dans une esquisse de *cadre de référence.*

C'est à l'ensemble de ces questions que nous tentons de répondre dans les prochains chapitres.

Chapitre 3 : Restructuration et explication de la démarche architecturale.

La clé de voûte de notre démarche de restructuration repose sur une conviction : les étapes nécessaires à la restructuration d'un logiciel à base de connaissances devraient offrir la possibilité d'augmenter la qualité de celui-ci. Dans ce travail, nous avons tenté d'apporter des indices théoriques et également issus de l'expérience, permettant d'étayer cette conviction.

3.1 Restructuration de COSMICXpert

3.1.1 Restructuration versus Réingénierie

Un premier élément tendant à appuyer notre conviction est la définition même de la restructuration d'un logiciel, donnée par Dumont [50] citant McClure [72].

> ***La restructuration*** est le processus de transformation d'une représentation d'un système en une autre forme. Habituellement, l'ancienne représentation et la nouvelle ont le même niveau d'abstraction. Cette transformation doit préserver le comportement externe du système, c'est à dire que le système doit conserver la même fonctionnalité. Pour effectuer le travail de restructuration, les analystes utilisent certaines approches de la rétro ingénierie des logiciels. […] La restructuration est souvent utilisée comme une forme de maintenance préventive permettant d'améliorer la qualité du système en respectant certains standards. Finalement, la restructuration du code peut permettre d'effectuer la maintenance adaptative du système.

Le fait d'utiliser la restructuration comme « maintenance préventive » implique que ce procédé permet d'améliorer la qualité. Toutefois, cette définition générale n'est pas suffisante pour s'appliquer à notre cas précis. En effet, notre projet de restructuration de COSMICXpert est une démarche un peu particulière. Si l'ouvrage effectué sur COSMICXpert s'apparente à de la restructuration dans une large mesure, il n'en présente pas moins une dimension de réingénierie puisque le produit final sera un logiciel dont le moteur et la base de connaissances présente des caractéristiques distinctes de COSMICXpert. Cette particularité

est due à la nature de COSMICXpert. En tant que logiciel à base de connaissances, il permet de maintenir et modifier de façon indépendante l'architecture et le contenu de sa base de connaissances (cf. Chapitre 1). Au final, on peut obtenir un logiciel qui semble totalement différent alors que seul le contenu de la base de connaissances a été modifié.

La question que nous devons résoudre est de savoir si le travail que nous fournissons est bien du domaine de la restructuration. Si tel est le cas, l'indice concernant l'amélioration de qualité sera valable. Afin de mieux cerner la démarche appliquée à COSMICXpert, citons [50] :

> ***La réingénierie*** est le processus d'analyse ou de modification d'un système dans le but de le reconstruire sous une nouvelle forme afin d'améliorer sa qualité et sa maintenabilité. […] la phase d'analyse de ce processus correspond au processus de rétro-ingénierie. De plus, la réingénierie est un processus qui fait appel à des notions de l'ingénierie des logiciels et de la restructuration du code source. Finalement, il est important de remarquer que la réingénierie permet la modification du système pour qu'il comble de nouveaux besoins, des besoins que le système original ne comblait. La Figure 3.1 présente le processus de réingénierie des logiciels.

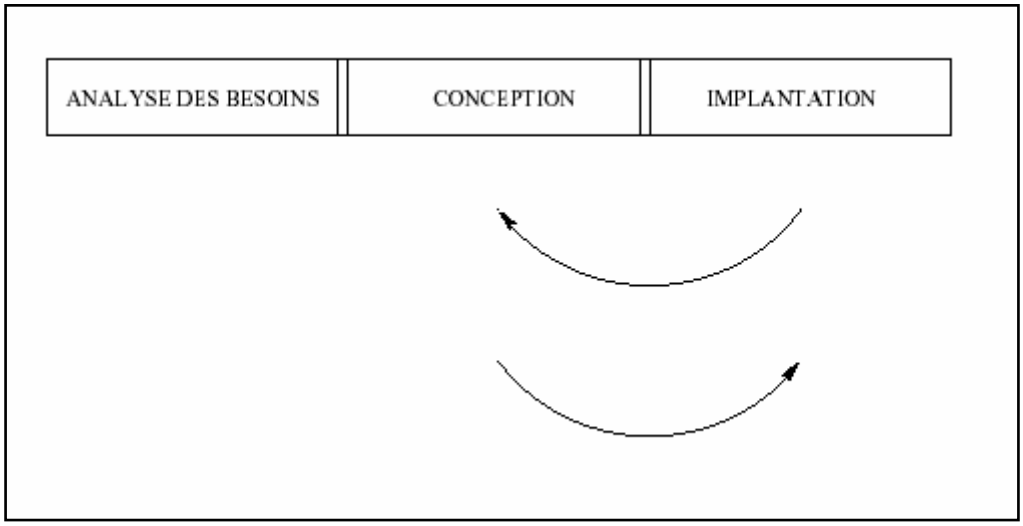


Figure 3.1 : Processus de la réingénierie.

Afin de rendre plus claires les définitions précédentes, nous pouvons ajouter cette dernière, toujours dans [50] :

La rétro-ingénierie est le processus par lequel on analyse un système dans le but d'identifier ses composantes et leurs relations. De plus, ce processus crée une représentation du système sous une autre forme ou à un niveau d'abstraction plus élevé. Contrairement à la réingénierie, la rétro-ingénierie est uniquement un processus d'analyse et non un processus de modification. La Figure 3.2 présente les processus de rétro-ingénierie des logiciels.

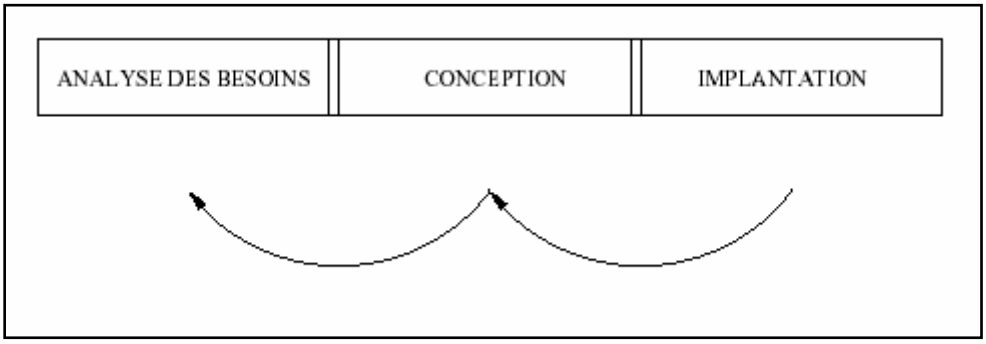


Figure 3.2 : Processus de la rétro-ingénierie.

Une autre définition est proposée par Nelson [73] :

"Reverse Engineering: The process of identifying software components, their interrelationships, and representing these entities at a higher level of abstraction. Reverse engineering by itself involves only analysis, not change. Program comprehension and program understanding are terms often used interchangeably with reverse engineering Four specializations of reverse engineering are offered, in increasing level of impact:

Redocumentation: Perhaps the weakest form of reverse engineering, this involves merely the creation (if none existed) or revision of system documentation at the same level of abstraction.

Design Rediscovery: Redocuments, but uses domain knowledge and other external information where possible to create a model of the system at a higher level of abstraction.

Restructuring: Lateral transformation of the system within the same level of abstraction. Also maintains same level of functionality and semantics.

Reengineering: The most radical and far reaching extension. Generally involves a combination of reverse engineering for comprehension, and a reapplication of

forward engineering to reexamine which functionalities need to be retained, deleted or added."

Bien que structurées de manières différentes, les deux approches concordent sur les trois points qui nous importent.

Trois éléments tendent à prouver que le passage de COSMICXpert reste bien du domaine de la restructuration et non de la réingénierie :

- Premièrement, le niveau d'abstraction du nouveau système reste inchangé. Les modifications apportées à l'architecture logique et décrites dans la suite de ce chapitre sont du domaine du réarrangement plutôt que de la conception pure. C'est donc l'architecture physique qui se voit la plus transformée et non l'architecture logique De ce fait, on garantit un niveau d'abstraction inchangé.
- Deuxièmement, les fonctionnalités principales proposées par SMXpert, bien que s'appliquant à d'autres connaissances, restent fondamentalement inchangées par rapport aux fonctionnalités principales de COSMICXpert.
- Troisièmement, les nouvelles fonctionnalités proposées par SMXpert ne constituent pas des changements majeurs mais sont de l'ordre des modifications propres à un cycle de maintenance. Ces fonctionnalités ajoutées n'impliquent pas une re-conception de l'application depuis sa genèse. Il ne s'agit pas de changements d*e rupture* par rapport à l'existant mais de changements *incrémentaux*[4].

Ces éléments nous confirment que la démarche suivie s'inscrit bien au niveau de la restructuration de logiciel. Le fait de pouvoir produire un logiciel restructuré « indépendant » du logiciel originel provient de cette possibilité de modifier le contenu de la base de connaissances en profondeur.

D'autres éléments tendent à renforcer la conviction citée plus haut. Ainsi, nous pouvons remarquer que le cycle de restructuration mené sur SMXpert (détaillé au Chapitre 6) partage un certain nombre de caractéristiques avec une phase de maintenance traditionnelle, qui par nature amène une amélioration de qualité.

[4] Les termes utilisés ici font référence, par analogie, aux termes désignant les innovations dans la théorie Schumpetérienne.

Tout d'abord, une phase de tests préliminaire permet, outre la familiarisation avec le logiciel, une mesure de sa qualité sur différents aspects tant fonctionnels que non fonctionnels. Cette partie de la restructuration est adaptée à l'utilisation de diverses méthodes de métrologie et permet de mettre en lumière des problèmes de *correction*. Puisqu'il serait inepte d'avoir connaissance de divers problèmes sans les résoudre, une partie de la restructuration sera consacrée à des modifications *correctives* et, donc, propres à une activité de maintenance. De plus, cette tâche est réalisée par des personnes différentes des développeurs et portant donc un regard plus objectif sur le logiciel, libre de toute considération personnelle ou de toute déformation due au temps consacré au développement du logiciel. Nous pouvons raisonnablement penser que cette caractéristique est indispensable à la bonne marche d'une maintenance de logiciel.

Ensuite, les réflexions nécessaires au changement du domaine de connaissances forcent à s'interroger sur la qualité intrinsèque des composants existants, et sur l'architecture (au niveau le plus concret) de ceux-ci. Le fait de devoir analyser en détail le fonctionnement du logiciel originel offre un recul permettant de détecter de nouvelles faiblesses. De plus, le fait de devoir prendre en compte des données telles que le changement de la taille (si les connaissances sur le domaine cible[5] sont plus nombreuses) ou le changement de structure de données, amène aussi une réflexion sur l'adéquation de l'architecture supportant l'application. En effet, des faiblesses au niveau des performances, jusqu'alors passées inaperçues en raison de la petitesse de la base de connaissances, peuvent apparaître lorsque la base s'élargit de manière sensible.

3.1.2 Remarques sur la méthodologie suivie

La méthodologie choisie pour le développement des nouvelles fonctionnalités reprend l'essentiel des étapes d'un cycle de vie en cascade. Afin de nous appuyer sur une méthodologie rigoureuse, nous avons choisi comme guide le document de Méthodologie v1.0 décrit par Naji Habra, Vincent Englebert, François Vermaut et Gautier Dallons dans le cadre du Laboratoire d'Ingénierie du Développement de Logiciels.

[5] Pour des raisons de clarté nous nommerons *domaine* cible le nouveau domaine à assigner au logiciel et *domaine* source celui du logiciel existant.

Ce chapitre traite du travail effectué en amont de la programmation proprement dite. A cet égard, il aborde des aspects d'ingénierie des besoins (cf. Section 3.2.1 et 3.2.2) et des aspects d'architecture logique abstraite (Section 3.2.4). Ces aspects sont globalement traités suivants l'optique du document cité plus haut (Laboratoire M.D.L.).

Toutefois, le fait de s'inscrire dans une restructuration possède deux avantages. D'une part, les concepts à haut niveau d'abstraction ne subissent que des modifications mineures (cf. Section 3.2.4). D'autre part, une documentation assez complète (particulièrement des concepts avec un bas niveau d'abstraction) existe pour le logiciel source et reste donc valable pour le logiciel cible.

Nous prenons donc quelques libertés vis-à-vis de la méthodologie de développement afin de la rendre plus souple. Notamment, nous ne produirons pas systématiquement tous les artefacts qui se devraient d'apparaître dans un développement depuis zéro.

3.2. Explication de la démarche architecturale.

3.2.1 Architecture de la base de connaissances.

Restructurer un logiciel à base de connaissances permet de modifier l'architecture du logiciel indépendamment de la structure de la base de connaissances qu'il manipule. Notre objectif étant de modifier le domaine de connaissance du logiciel, nous avons commencé par la restructuration de la base de connaissances. Toute la difficulté de la démarche réside dans le fait de définir un schéma conceptuel du logiciel cible qui soit suffisamment proche du schéma conceptuel du domaine source. Toutefois, cette ressemblance entre schémas n'exclut pas certaines caractéristiques propres à chacun d'eux. Dès lors, même si les structures de données prévues pour modéliser les connaissances de COSMICXpert sont adaptées à la formalisation de la connaissance de SMXpert, certains ajouts ont dû être apportés afin de satisfaire aux exigences du nouveau domaine. Nous détaillons ces ajouts dans la suite tandis que l'Annexe 3 présente l'entièreté des structures de données propres à SMXpert. La section

suivante décrira en détails comment les fonctionnalités reposant sur ces structures de données additionnelles ont été mises en œuvre.

3.2.1.1 Prise en compte d'un profil et d'un filtre utilisateur

Nous avons vu au chapitre précédent que SMXpert comptait parmi ses exigences la possibilité d'éditer un profil utilisateur selon lequel la connaissance présentée à l'évaluateur sera filtrée. Le profil utilisateur reflète le niveau actuel de maturité des processus de l'utilisateur/évaluateur. Ce niveau de maturité est exprimé par le biais de quatre attributs reflétant chacun un aspect de la maturité des processus auquel l'utilisateur est confronté:

- La **Catégorie d'Utilisateur** : cet attribut reflète, comme son nom l'indique, la catégorie à laquelle appartient l'utilisateur. Un utilisateur peut être un simple **client/utilisateur** ou un **développeur/manager**.
- Le **Lieu de Maintenance** : cet attribut s'attache à voir si le processus de maintenance que l'utilisateur met en question se déroule dans ses locaux ou à l'extérieur. A cet effet, l'attribut peut prendre deux valeurs : **sous-traitée** ou **sur place**.
- La **Taille de l'Organisation** : cet attribut permet de cibler quantitativement à quel genre d'utilisateur le système s'adresse. Les valeurs possibles correspondent à trois intervalles distincts allant de **un ou quelques individus** à **plus de 30 individus** et en passant par **moins de 30 individus.**
- Le **Niveau de Maturité** : cet attribut reflète le niveau de maturité de l'organisation de l'utilisateur, basé sur l'échelle de maturité d'un logiciel décrite dans le ***Modèle d'Évaluation de la Capacité à Maintenir le Logiciel*** des Professeurs April et Abran. Cette échelle va donc de **1** (maintenance initiale) jusqu'à **5** (maintenance en continu).

Chacun des attributs précédents peut, en outre, être associé à une valeur neutre, dite *unknown*. Cette valeur permet de laisser des entrées de la base connaissances libres d'accès à tout type d'utilisateur (cf. Section 3.2.2.1). Ainsi, si l'expert rédige une **question** associée à une **catégorie d'utilisateur** *unknown*, les utilisateurs de toutes catégories pourront se voir proposer la question.

Cette exigence a un impact double sur les structures de données. Elle nécessite en effet de modifier deux types de documents. Le document *users* doit recueillir les informations sur le profil tandis que chaque **question** appartenant à un document *concept de maintenance* doit désormais préciser à quel profil utilisateur elle convient.

A cette fin, le document *users* s'est vu ajouter un élément *filter* comprenant lui-même quatre éléments représentant les quatre attributs cités plus haut. Ces éléments sont respectivement : *type_user*, *where_maintenance*, *size_organisation* et *maturity_level*. De même, chaque élément *question* d'un document *concept de maintenance* se voit ajouter, en guise de descendants, les mêmes cinq éléments.

3.2.1.2 Prise en compte des recherches itératives

Une autre caractéristique de SMXpert : son aspect itératif. En effet, la possibilité est donnée à l'évaluateur d'approfondir une recherche n'ayant pas abouti à une recommandation finale en continuant celle-ci avec un autre mot-clé. Du point de vue des données, ce mécanisme nécessite l'ajout d'un lien entre une **recommandation** et le **mot-clé** par lequel l'approfondissement de la recherche peut être effectuée. Concrètement, chaque élément *REC* d'un document *cas problème* est muni d'un nouvel attribut *redir* contenant l'identifiant du **mot-clé** en question au sein du document *glossary_KW*. Ce nouvel attribut peut être laissé vide si la **recommandation** ne prévoit pas de redirection. La Section 3.2.2.5 décrit comment la fonctionnalité associée a été mise en œuvre sur base de cette nouvelle structure de données.

3.2.1.3 Prise en compte d'un lien entre question et recommandation

Comme nous le décrirons à la Section 3.2.3, le moteur d'inférence de SMXpert s'appuie sur les fonctionnalités de lien d'une recommandation à une question. De façon similaire au mécanisme de la section précédente, les structures de données ont été modifiées de manière à autoriser un tel lien. Concrètement, chaque document *concept de maintenance* se voit ajouter un élément *REClink* délimitant une série d'élément *reco*, définissant à leur tour la recommandation par défaut, par cas problème, pour ce concept de maintenance. Chaque

élément *TH*[6] possède également son élément *REClink*. Pareillement, cet élément renferme autant d'éléments *reco* qu'il existe de **cas problème** pour ce **concept**. Chaque *reco* possède l'identifiant de la **recommandation** liée à cette **question** ainsi que le nom du document *cas problème* dans lequel on peut la trouver. L'utilisation de ces nouvelles structures permettra de mettre en œuvre les fonctionnalités décrites aux Sections 3.2.2.3 et 3.2.2.4.

3.2.1.4 Prise en compte de la taille du domaine

Le domaine de la maintenance étant vaste, nous avons du prendre en compte la possibilité de trouver un grand nombre d'entrées **d'index** et de **mots-clés**. La version actuelle de la base de connaissances, contenant la quasi-totalité desdites entrées, dénombre 549 mots d'**index** et 71 **mots-clés,** contre 19 et 17 pour COSMICXpert**.** Afin de prendre en compte cette taille plus élevée, nous avons scindé le document *glossary* en deux documents spécialisés *glossary_KW*, reprenant l'ensemble des **mots-clés**, et *glossary_INDEX* contenant les mots de l'**index**. Cette division devrait nous empêcher de supporter des coûts trop importants en matière de temps de traitement.

3.2.1.5 Maintien de la compatibilité arrière

La règle d'or adoptée pour la restructuration de la base de connaissances est de maintenir un maximum de compatibilité. Nous avons donc conservé des éléments dans les diverses structures qui n'ont pas de rôle apparent pour le fonctionnement de SMXpert. Citons par exemple, l'attribut *cf* d'un élément *fact* (dans un document *concept de maintenance***)** qui n'est plus nécessaire au fonctionnement du moteur d'inférence (cf. Section 3.2.3). Ce choix se justifie par notre volonté de garantir un niveau élevé de compatibilité arrière entre les structures des bases de connaissances. Ainsi, les structures de données résultantes permettent de modéliser les connaissances du domaine de SMXpert *et* du domaine de COSMICXpert.

[6] Rappelons qu'un élément *TH* représente une question liée à concept de maintenance donnée. Bien que contre intuitive, cette appellation se justifie par deux faits distincts. La première cause est notre volonté de réutiliser au maximum les structures de données existantes. D'autre part, le « th » provient de l'équivalence entre **thème** et **question** au niveau des ontologies. Pour plus de détails sur les structures de données, le lecteur pourra se référer à l'Annexe 3.

Nous verrons au Chapitre 6 que ce choix permet, dans certains cas, d'améliorer la qualité du logiciel originel.

3.2.2 Nouvelles fonctionnalités

Sur base des exigences formulées par le Professeur April, expert dans le domaine cible, notre démarche de restructuration s'est accompagnée de l'ajout de quelques fonctionnalités importantes pour le fonctionnement de SMXpert. Comme énoncé précédemment, ces fonctionnalités ne constituent pas des modifications majeures mais sont plutôt de l'ordre de la modification *adaptative*. Nous listons ci-dessous les fonctionnalités en question. Pour toutes informations supplémentaires, le lecteur pour se référer aux *cas d'utilisation* de l'Annexe 2.

3.2.2.1 Création d'un profil et d'un filtre utilisateur

Les discussions avec le Professeur April ont mis en lumière que l'orientation d'un client vers une bonne solution concernant ses processus de maintenance est intimement liée au type du client visé. Ainsi, contrairement au domaine de la mesure logicielle, les connaissances traitées et dispensées par SMXpert devront prendre en compte le niveau de maturité des processus de l'utilisateur. Nous avons opté pour l'ajout d'une fonctionnalité **filtre utilisateur** afin de traiter cet aspect dynamique, inexistant jusqu'alors. Ce filtre permet donc de trier les connaissances de façon transversale et dynamique. Concrètement, l'expert possède un outil supplémentaire permettant d'associer un niveau de maturité pertinent à chaque **question** introduite dans la base de connaissances. Ce niveau de maturité est exprimé par le biais des quatre attributs cités dans la Section 3.2.1.1.

L'utilisation du **filtre utilisateur** se marque à trois niveaux distincts :

Premièrement, l'expert doit avoir conscience que toute opération qu'il effectue sur la base de connaissances est toujours liée à une occurrence de filtre, instancié avec une valeur pour chaque attribut cité. Par défaut, la valeur du filtre qu'il applique à chaque **question** de la base de connaissances reste constante durant toute la session de travail de l'expert. Au lancement de la session, elle est instanciée à *unknown* pour chaque attribut. Toutefois, l'expert peut

modifier la valeur de ce filtre à tout moment. Pour ce faire, il dispose d'une option **Set Filter** apparaissant dans le menu d'édition de la Page Expert (cf. Figure 3.3).

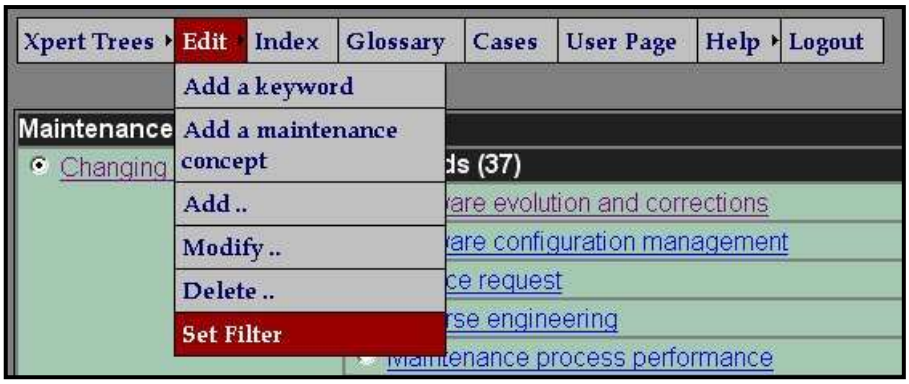


Figure 3.3 : Option **Set Filter** dans le menu d'édition

Cette option commande l'apparition d'un panneau flottant permettant de visualiser la valeur actuelle du **filtre utilisateur** et de paramétrer chacun de ses attributs (cf. Figure 3.4). Une fois paramétré, cette instance de filtre sera associée à chaque nouvelle entrée de la base de connaissances. Il est à noter que l'expert peut alors entrer à deux reprises une **question** portant le même intitulé mais avec des instances de filtre différentes. Ainsi l'identifiant d'une **question** dans la base de connaissances prend-il en compte les valeurs d'attributs du filtre et non pas le seul intitulé de la question. Cette flexibilité permettra donc de poser une même question sous des formes différentes et adéquates par rapport au type d'utilisateur consultant la base de connaissances.

Figure 3.4 : Menu de paramétrage du **filtre utilisateur**

Le deuxième aspect de l'utilisation d'un **filtre utilisateur** est évidemment le fait d'associer une instance de filtre à chaque utilisateur, révélant à quel niveau de maturité ce dernier est associé et, donc, quel type de connaissances il est pertinent de lui présenter. Nous nommerons également cette instance de filtre, le **profil utilisateur**. À cet effet, le panneau d'ajout d'un utilisateur a été agrémenté de quatre questions supplémentaires concernant le niveau de maturité de l'utilisateur que l'administrateur enregistre dans le système (cf. Figure 3.5).

Figure 3.5 : Menu d'ajout d'un utilisateur.

Cependant, le niveau de maturité d'un utilisateur est loin d'être fixe au cours de son existence. Le simple fait qu'il utilise SMXpert est une preuve de sa volonté de changer ce niveau. De ce fait, il est indispensable de permettre l'édition de son profil dans le temps. Cette opération est rendue possible par l'ajout d'une option au menu d'administration des utilisateurs du système. A côté des options existantes dans COSMICXpert, un bouton proposant d'éditer le profil de l'utilisateur a été ajouté (cf. Figure 3.6).

Figure 3.6 : Options d'édition des informations d'un utilisateur.

Le bouton **Edit User Profile** commande l'apparition d'un panneau flottant permettant de modifier les valeurs du filtre associé à un utilisateur (cf. Figure 3.7). Ce menu propose de modifier la valeur de chaque attribut. Une fois l'opération effectuée, l'utilisateur sera enregistré avec un niveau de maturité conforme aux valeurs introduites et se verra proposer, dans l'interface utilisateur, les connaissances adaptée à son nouveau niveau.

Figure 3.7 : Menu d'édition du profil utilisateur

Enfin, la dernière exigence pour que la fonctionnalité **de filtre utilisateur** soit implémentée de manière efficace est la possibilité de mettre en relation les deux derniers aspects. Ainsi, l'interface utilisateur sera-t-elle modifiée de façon à trier en temps réel les connaissances accessibles à l'utilisateur en fonction de son profil. Le principe de ce tri est simple. Une **question** n'est affichée à l'écran que si les valeurs des quatre attributs cités plus haut sont équivalentes dans l'instance de filtre liée à la question et dans le profil de l'utilisateur en cours. Ces trois mécanismes permettent d'assurer qu'un tri transversal basé sur le type d'utilisateur est effectué tandis que le travail d'orientation vers une recommandation correcte continue de manière « longitudinale ».

3.2.2.2 Fixer et modifier la position d'une question dans son Concept de Maintenance

Il est également apparu que la possibilité de placer les **questions** dans un ordre précis aurait un impact sur la manière de traiter les connaissances de la base. Les raisons de ce choix seront plus amplement détaillées dans la section consacrée au moteur d'inférence. L'impact sur les fonctionnalités est double. Il faut en effet pouvoir fixer la place, et donc la priorité, d'une **question** par rapport à son **concept de maintenance** mais, de plus, il faut pouvoir modifier cet ordre si le besoin s'en fait sentir.

Deux pages ont donc été ajoutées pour implémenter ces deux opérations.

Une première page vient en préambule de l'ajout d'une **question** et un menu y propose de placer la nouvelle **question** relativement à ses pairs (cf. Figure 3.8).

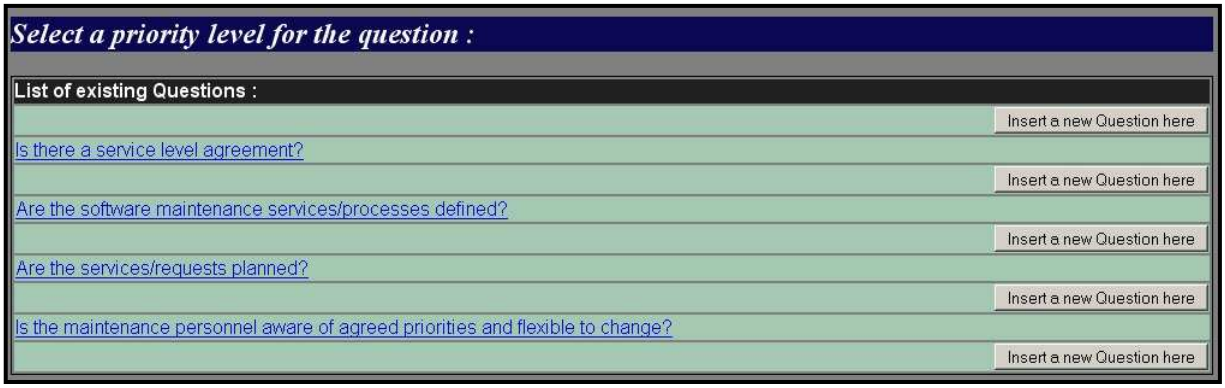


Figure 3.8 : Menu de positionnement d'une nouvelle question.

De même, la première page consacrée à la modification d'une **question** est désormais dédiée à un menu permettant de déplacer la **question** visée (cf. Figure 3.9). Il est à noter que pour des raisons de facilité, d'ergonomie et d'esthétique, ce menu propose bel et bien de *permuter* la **question** avec une autre présente dans la liste. Evidemment la possibilité de laisser la position inchangée est laissée à l'utilisateur.

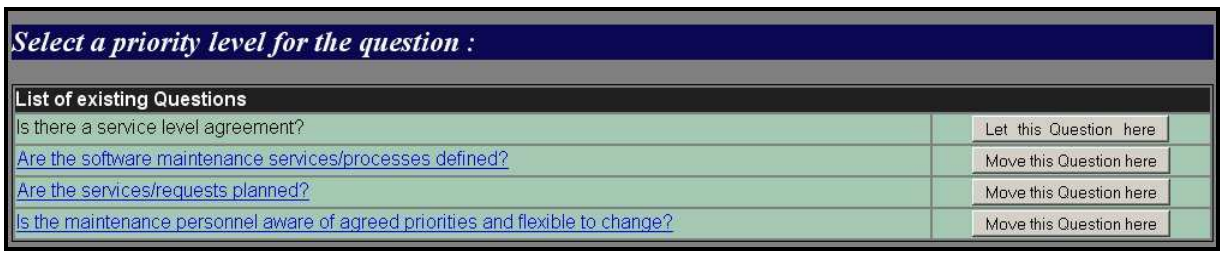


Figure 3.9 : Menu de modification de la position d'une question

Il est a noter que, si elle présente un intérêt primordial dans la réalisation de SMXpert (cf. *infra*), la fonctionnalité de positionnement/déplacement d'une **question** reste d'une faible complexité. En effet, la démarche ne consiste en rien de plus qu'un changement d'identifiant dans la base de connaissances. En d'autres termes, le changement de priorité de la **question** s'accompagne d'une modification réelle de sa place dans la base de connaissances, aisée à implémenter.

3.2.2.3 Lier une recommandation à une question

Encore une fois, il s'agit d'une fonctionnalité dérivée d'une exigence primordiale du domaine cible et de la manière de raisonner sur celui-ci. Comme expliqué dans la section précédente, les nouvelles structures de données obligent de lier une recommandation à une question en particulier. Ce lien prendra tout son sens par la suite, dans la manière dont le moteur d'inférence va gérer ces raisonnements. Il est à noter que les **questions** présentes dans la structure de données de SMXpert sont plus intimement liées à un **cas problème** qu'à leur **concept de maintenance**. Cette différence avec COSMICXpert se marque donc par la nécessité qu'une question possède un lien avec une **recommandation** de chaque **cas problème** pour un **concept de maintenance**.

Afin d'implémenter cette nouvelle fonctionnalité, un nouveau menu sera nécessaire. Celui-ci apparaîtra à la dernière étape des procédures d'ajout et de modification d'une **question**. Il consistera simplement en une liste de **recommandations**, groupées selon leurs **cas problèmes** respectifs, et invitera l'utilisateur à sélectionner une et une seule **recommandation** dans *chaque* groupe. Les **recommandations** sélectionnées seront liées à la **question** pour le **cas problème** visé. La Figure 3.10 illustre la dernière étape de l'ajout (ou de la modification) d'une **question** et le menu décrit *supra*. Dans cette illustration, nous pouvons voir que la **question** va être ajoutée (ou modifiée) et que cette **question** sera liée à la deuxième **recommandation** du premier **cas problème** tandis qu'elle sera liée à la première **recommandation** du second **cas**.

Cette fonctionnalité permet donc d'introduire des connaissances compatibles avec les structures décrites à la section précédente.

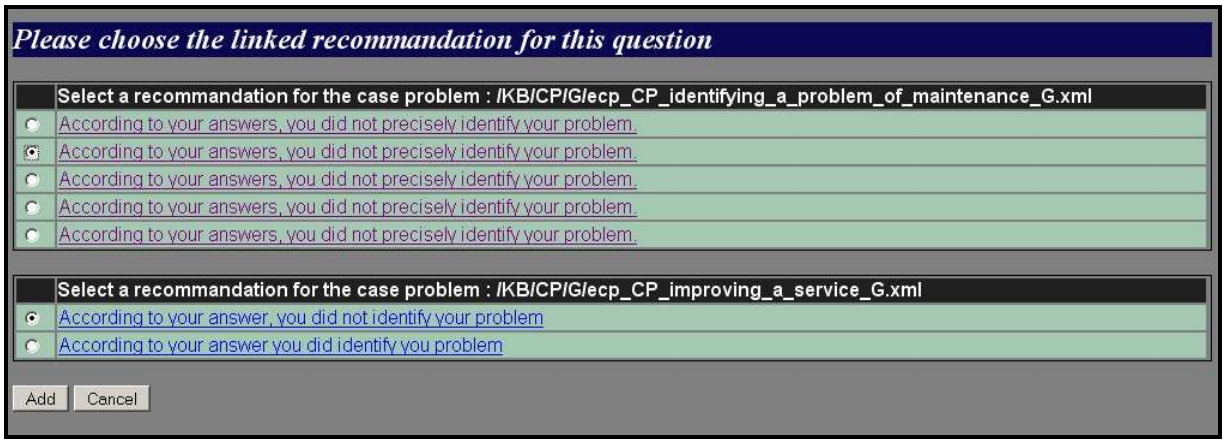


Figure 3.10 : Menue de sélection des recommandations liée à une question.

3.2.2.4 Lier une recommandation à un concept de maintenance

Dans la même optique que la fonctionnalité précédente, la possibilité de lier une **recommandation** à un **concept de maintenance** permet d'assigner une « recommandation par défaut » pour un **concept de maintenance** donné. Nous verrons par la suite qu'il s'agit d'une fonctionnalité essentielle au bon fonctionnement du moteur d'inférence. La structure étant pensée pour supporter ce lien entre **recommandation** et **concept de maintenance**, il nous faut encore ajouter une fonctionnalité permettant d'ajouter ce lien ou de le modifier. Pour ce faire, une étape est ajoutée en fin de la procédure d'ajout ou de modification d'un **cas problème**. On y trouve un menu invitant l'utilisateur à sélectionner une **recommandation** par défaut pour ce **concept de maintenance** (cf. Figure 3.11). Une fois, sélectionnée, la **recommandation** sera liée au **concept de maintenance**.

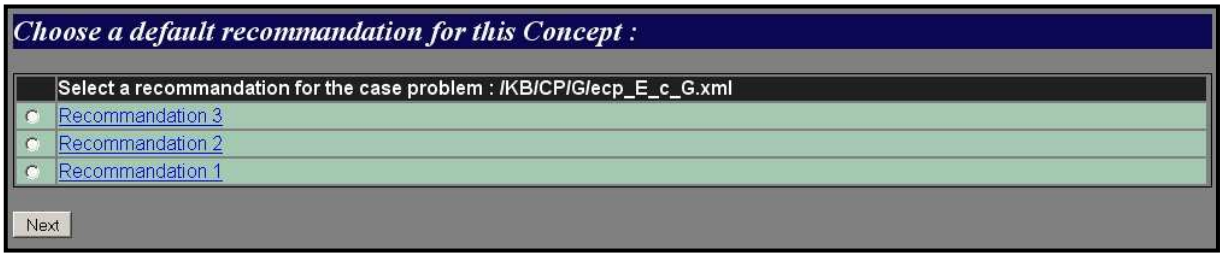


Figure 3.11 : Menue de sélection d'une recommandation par défaut pour un cas problème

3.2.2.5 Assigner une redirection à une recommandation

Une des principales différences entre le fonctionnement SMXpert et celui de son prédécesseur est son caractère itératif. En effet, l'utilisateur peut être guidé vers une **recommandation** proposant de relancer une recherche sur base d'un autre **mot-clé,** apparaissant au système comme plus approprié pour le problème de l'utilisateur. Cette différence vient du fait que le processus d'aide à la décision en matière de maintenance est bien souvent un processus itératif. Un outil automatisé implémentant ce processus se doit donc d'être lui-même itératif. Nous avons vu précédemment que la structure de données d'une **recommandation** a été agrémentée d'une **redirection**. C'est grâce à cet attribut que cette fonctionnalité est implémentée.

Concrètement, le processus d'ajout ou de modification d'une **recommandation** comprend désormais une option permettant de lier un **mot-clé** à celle-ci. L'expert pourra laisser cette option sur une valeur par défaut « *no redirection* » qui, comme sont nom l'indique, postule que la **recommandation** ajoutée (ou modifiée) est terminale et qu'aucune démarche supplémentaire n'est nécessaire ou possible pour résoudre le problème de l'utilisateur. Au contraire, si la **recommandation** introduite par l'expert n'est qu'une étape dans une recherche plus longue, il pourra assigner un **mot-clé** constituant le début d'une nouvelle recherche. Cette option prend la forme d'une liste déroulante comprenant tous les **mots-clé** de la base de connaissances ainsi que la valeur par défaut *no redirection* (cf. Figure 3.12). Une fois sélectionné, le **mot-clé** sera lié à cette **recommandation** en tant que redirection pour l'utilisateur.

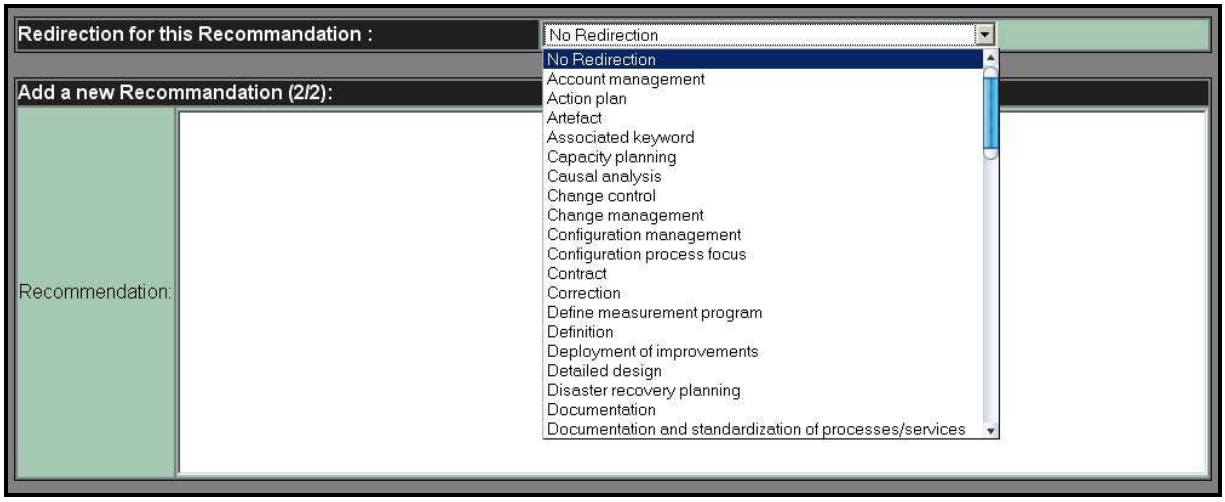


Figure 3.12 : Menu de redirection d'une recommandation.

Du point de vue de l'utilisateur, cette nouvelle fonctionnalité se marque par un nouveau bouton à son attention. Lorsque l'interface utilisateur affiche la **recommandation** déduite par le moteur d'inférence, un bouton invitant l'utilisateur à continuer ses recherches s'affiche dans la réponse de la **recommandation**, si celle-ci n'est pas la dernière (cf. Figure 3.13). Ce bouton actionne le redémarrage d'une recherche avec comme **mot-clé**, le mot-clé associé suivant la méthode vue *supra*, et propose à l'utilisateur la liste des **concepts de maintenance** liés à ce mot. La procédure est en tout point équivalente à une recherche lancée par l'utilisateur sur le **mot-clé** en question, à l'exception près que la première étape est ici commandée automatiquement par l'application. Ces deux mécanismes assurent que l'aspect itératif du processus d'aide à la maintenance soit pris en compte dans la structure de données.

Figure 3.13 : Bouton de redirection vers une recherche additionnelle.

3.2.3 Evolution du moteur d'inférence et nouvelles fonctionnalités.

Il nous faut désormais examiner quels raisonnements sont applicables à la manipulation des concepts de la nouvelle base de connaissances. Ce travail pourrait s'avérer d'une rare complexité puisque le moteur d'inférence est le centre névralgique du logiciel. Mais dans le cas de la restructuration de COSMICXpert, cette tâche n'a pas été la plus complexe.

Tout comme son prédécesseur, **SMXpert Web Edition** présente la particularité de poser deux types de raisonnement : basés sur les cas et basés sur les règles. Les raisonnements basés sur des cas prennent place lors de la première partie du travail de l'évaluateur. Le travail effectué pour générer une ontologie basée sur le Modèle d'Évaluation de la Capacité à Maintenir le Logiciel [49] suffisamment proche de celle de COSMICXpert, rend cette partie du moteur d'inférence originel déjà adaptée. Le raisonnement par cas ne nécessite donc pas d'effort de restructuration.

A contrario, le moteur d'inférence manipulant les règles a du être retravaillé afin de correspondre aux besoins de SMXpert. Comme nous l'avons cité plus haut, ce moteur adapte la façon le raisonner d'un expert utilisant le **SM^{MM}**. La méthode de raisonnement d'un expert est intimement basée sur les questions qu'il pose à l'utilisateur, et l'ordre dans lequel celles-ci

sont posées. Nous avons vu dans la précédente section que les structures de données ont été modifiées de manière à supporter de nouvelles associations, le moteur d'inférence de SMXpert profitera donc de cette nouvelle possibilité pour appliquer les raisonnements du **SM^{MM}**.

Ces raisonnements utilisent donc les **questions** et les réponses à celles-ci pour orienter l'utilisateur vers une **recommandation** adéquate. Cette **recommandation** peut à son tour amener l'utilisateur à relancer une phase d'inférence (cf. Section 3.2.2.5). Concrètement, le raisonnement effectué par le moteur d'inférence s'apparente à un arbre de décision simple ou chaque réponse à une question oriente l'utilisateur vers une nouvelle recommandation. Un arbre de décision exposant cette méthode est présenté à la Figure 3.14.

Connaissances :

Questions : Q = {Q_1, Q_2,…, Q_N}

Recommandations : Rc = {$Recom_0$, $Recom_1$,…, $Recom_N$}

Redirections : Rd = {$Redir_1$, $Redir_2$,…, $Redir_N$}

Arbre de décision :

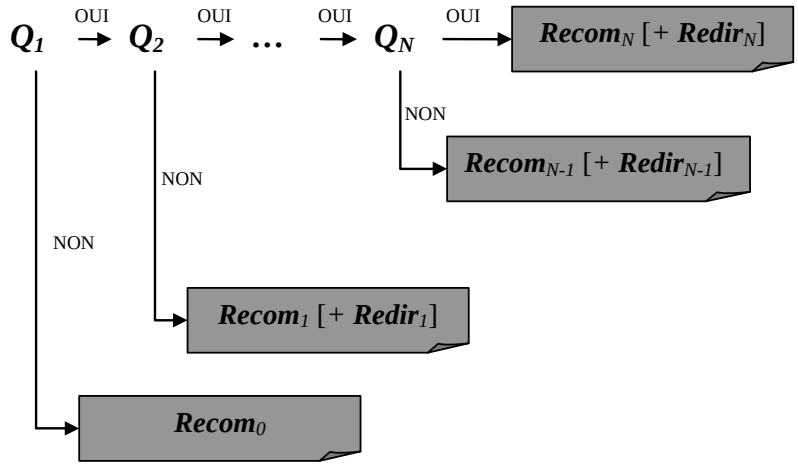


Figure 3.14 : Schéma de la méthode de décision du moteur d'inférence

Il est à noter que nous sommes faces à un ensemble de **questions** hiérarchisées. En effet, la première question mène à une **recommandation** ne comportant aucune redirection vers

d'autres recherches. Ainsi la première question est-elle la question « de base ». Elle symbolise le cas où le problème de l'évaluateur trouve une solution immédiate à son problème. Les *n-1* **questions** suivantes sont des interrogations destinées à raffiner un problème trop complexe. Toutefois, chaque redirection présente dans chaque feuille de l'arbre de décision reste un élément *facultatif*. Ainsi, rien n'oblige l'expert à ne posséder qu'une **question** à réponse immédiate. Enfin, on notera que l'ensemble des redirections n'est, concrètement, qu'un ensemble de mots-clés vers lesquels la recherche peut être relancée. Par conséquent, si l'ensemble des mots-clés est représenté par Mc, nous obtenons :

$$\mathbf{Rd} \subseteq \mathbf{Mc}$$

Il est à noter que cette hiérarchisation de l'ensemble **Q** ainsi que l'ajout des fonctionnalités citées plus haut rend le travail du moteur d'inférence beaucoup plus léger. Cette simplification de la tâche du moteur déplace le problème au niveau de l'expert humain qui, lui, devra veiller à ce que les **questions** soient bien ordonnées de façon hiérarchique. De même, c'est sur l'expert humain que repose à tâche d'orchestrer correctement les redirections d'une **recommandation** vers un nouveau **concept de maintenance**. On pourrait donc conclure que la restructuration de SMXpert rend, en quelque sorte, le système *moins* expert. Toutefois, il faut rester conscient qu'il s'agit là d'un choix issu d'une réflexion sur le niveau d'automatisation applicable au domaine de la maintenance. Il ne s'agit donc pas d'une limitation liée à la restructuration elle-même, ce qui laisserait croire à tort que celle-ci amène une perte de qualité et non une amélioration.

Il reste enfin à évoquer le rapport existant entre les raisonnements à base de cas et les raisonnements à base de règles. En effet, ces deux types de raisonnements ne sont pas indépendants l'un de l'autre, sinon ils perdraient tout intérêt. Ainsi les raisonnements de la première partie, basés sur les cas, vont-ils orienter les possibilités de réponse du moteur manipulant les règles. Par analogie, on pourrait voir dans les raisonnements susnommés des aiguillages permettant de diriger la voie suivie par un raisonnement global vers la recommandation attendue par l'évaluateur. Concrètement, l'arbre de décision présenté plus haut sera donc différent en fonction du **cas problème** sélectionné au préalable, dans la mesure où le lien entre une **question** et sa **recommandation** varie avec le **cas problème**. Les Figures 3.15 et 3.16 présentent deux variantes d'une même recherche et illustre l'influence de la recherche d'un cas problème sur les réponses du système aux questions.

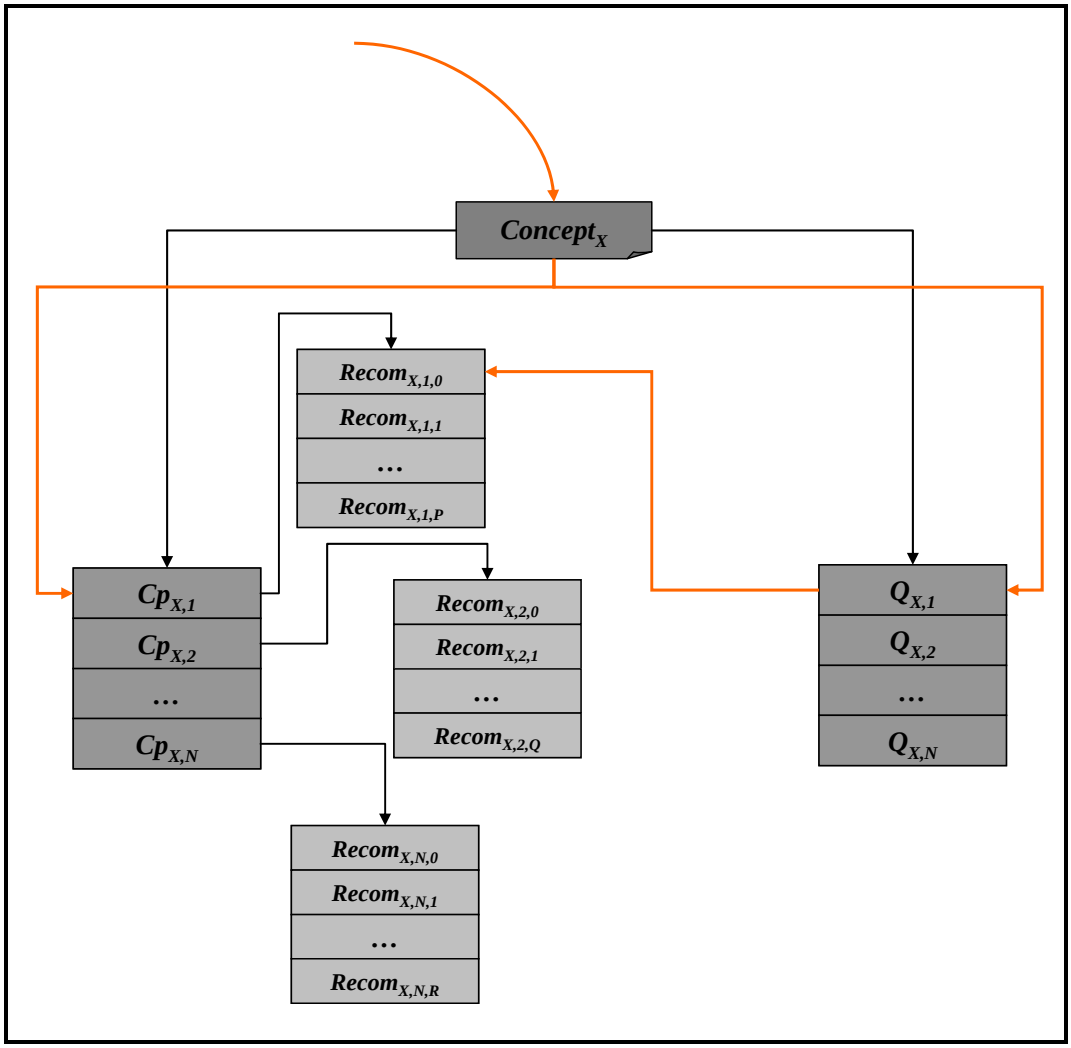


Figure 3.15 : Influence du cas sélectionné sur le comportement du moteur d'inférence (première variante)

Dans la figure ci-dessus les traits noirs symbolisent les liens existants entre les différents concepts, tandis que les traits de couleur représentent le cheminement d'un évaluateur au sein de ceux-ci. Plus précisément, cette figure présente le cheminement d'un évaluateur qui, par le biais des mécanismes de recherches par **index** et par **mot-clé**, arrive à identifier le **concept de maintenance** $Concept_X$ comme répondant à son problème. On voit que le $Concept_X$ est lié à N **cas problèmes**. Le **cas problème** répondant aux besoins de l'utilisateur est $Cp_{X,1}$, possédant lui-même P recommandations. Nous voyons alors que l'évaluateur répond négativement à la première question qui se présente à lui, soit $Q_{X,1}$. Comme le prévoit son arbre de décision, le moteur d'inférence oriente l'évaluateur vers la **recommandation** par défaut $Recom_0$. Or le choix du **cas problème** provoque une indirection de ce choix vers la **recommandation** par défaut *spécifique* à ce dernier, soit $Recom_{X,1,0}$.

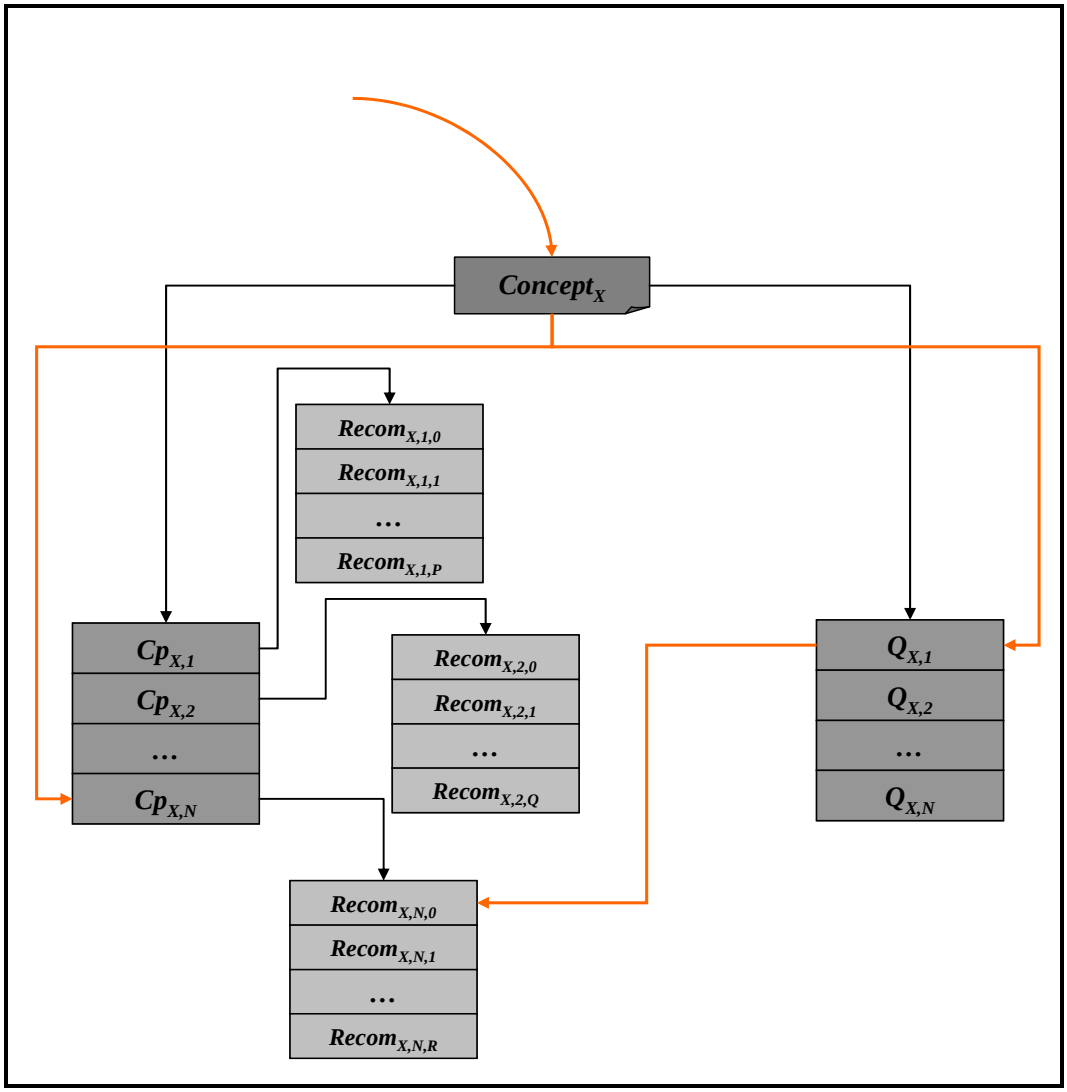


Figure 3.16 : Influence du cas sélectionné sur le comportement du moteur d'inférence (deuxième variante)

La figure ci-dessus reprend la situation de la Figure 3.15 mais dans le cas où l'évaluateur identifierait le **cas problème** $Cp_{X,N}$ comme répondant à ses besoins. On voit alors que, ayant répondu pareillement à la même question, l'évaluateur se voit proposer la **recommandation** $Recom_{X,1,0}$ en guise de solution à son problème. Ces deux figures illustrent donc parfaitement l'analogie de l'aiguillage et, donc, le rapport entre les raisonnements à base de cas et les raisonnements à base de règles dans le fonctionnement global du moteur d'inférence.

3.2.4 Architecture du logiciel

3.2.4.1 Motivation et adéquation avec la démarche de restructuration

Repenser l'architecture de COSMICXpert constitue une étape originale pour une démarche de restructuration. En effet, un point essentiel distinguant une restructuration d'un processus de réingénierie postule que le système doit garder le même niveau d'abstraction (cf. Section 3.1.1). Il paraît donc contradictoire de réviser l'architecture du logiciel, particulièrement à haut niveau. Toutefois, cette étape apparaissait comme indispensable à plusieurs égards.

Tout d'abord, une documentation hétérogène et, plus particulièrement *l'absence* de documentation sur l'architecture logique de COSMICXpert constituaient une faiblesse pour la maintenabilité de ce dernier. En effet, les générations successives de développeurs ont, à chaque étape de l'évolution, gardé les spécifications du système à un niveau très proche de l'implémentation physique. Aucune représentation des différents modules à un haut niveau n'a jamais été produite. Dans cette optique, nous avons trouvé pertinent de dégager rigoureusement ladite architecture, et de la restructurer légèrement pour les besoins de SMXpert. Notre tâche n'a donc pas été de recréer l'architecture à haut niveau de COSMICXpert mais de la préciser. Ensuite, comme expliqué précédemment, le travail de modification logique de COSMICXpert relève de l'affinage plutôt que de la reconstruction. C'est l'implémentation physique qui a subi les plus profonds changements. Le niveau d'abstraction est donc conservé.

D'autre part, la restructuration des différentes couches de l'architecture, et plus particulièrement les couches de bas niveau, s'inscrit parfaitement dans notre démarche d'amélioration de la qualité. En effet, le travail apporté à la gestion des nouvelles exigences permet de devoir manipuler les concepts de haut niveau et, donc, l'architecture logique tout comme les préceptes qui l'accompagnent. Parallèlement, l'implémentation de ces nouvelles exigences permet de manipuler des concepts de bas niveau et d'apporter à cette occasion une modification des architectures physique et concrète. Ces deux axes de la restructuration permettent au final d'avoir un gain de performances et de qualité, comme nous l'exposons dans la section suivante.

3.2.4.2 Ligne directrice et effet escompté

Dans cette section, nous allons présenter les lignes directrices ayant présidé à nos modifications sur l'architecture. Dans un premier temps, nous exposons les faiblesses que nous désirons corriger. Ensuite, nous détaillons les préceptes qui devront régir la restructuration architecturale (tant logique que physique). Et enfin, nous exposons en quoi ces lignes directrices sont sensées nous apporter une réponse aux faiblesses précitées.

Parmi les faiblesses du système (cf. Chapitre 2) que notre restructuration tentera de corriger, citons :

- **La maintenabilité :** cette insuffisance de maintenabilité se marque par un manque de documentation à haut niveau d'abstraction.
- **Problèmes de lisibilité du code :** ce dernier est dû au mélange technologique, au grand nombre de traitements placés dans l'interface (pages JSP) ainsi qu'au 'style' de programmation des différentes générations de développeurs.
- **Le couplage élevé :** ce problème est lié à la quantité de traitements intégrés aux pages elles-mêmes. Il devient alors difficile de modifier une interface utilisateur sans toucher à la logique d'affaires, et vice versa.
- **Les performances :** cette faiblesse des performances (particulièrement du temps de réponse) provient également de la localisation des traitements. Le code présent dans les pages ralentit les opérations dans la mesure où le côté client n'est pas adapté pour effectuer les traitements propres à la logique d'affaires.
- **La sécurité :** même si cet aspect n'est pas primordial dans notre cas, la présence de code propre à la logique d'affaires au sein de l'interface reste une faille de sécurité importante.

Conscients de ces problèmes, nous avons tenté de redessiner une architecture qui prennent en compte ces derniers. Nous avons défini une série de préceptes afin de nous guider dans ce processus de restructuration. Ainsi, à tout moment de l'étape de restructuration, une grande attention aura été donnée à :

- **La modularité :** nous veillerons à concevoir ou transformer les composants tout en gardant une grande indépendance entre ceux-ci, garantissant ainsi une modularité élevée.
- **La réutilisation :** afin de ne pas sortir de notre démarche de restructuration et garantir une continuité avec le passé, tout code préexistant réutilisable sera conservé. Nous veillerons à trouver le moyen de maximiser cette réutilisation.
- **L'emplacement des traitements :** le code propre aux traitements lourds de la *logique d'affaires* devra se situer du côté serveur. Techniquement, nous veillerons donc à minimiser le code Java intégré aux pages JSP.
- **La lisibilité du code :** nous veillerons à uniformiser autant que possible les méthodes de codage présentes dans le système.

Par l'application de ces préceptes et la clarification de l'architecture logique, nous espérons voir ressortir les avantages suivants :

- **Une maintenabilité accrue du système :** l'architecture logique étant clarifiée, il sera plus facile d'appréhender le système dans sa globalité pour de futurs mainteneurs**.** D'autre part, l'amélioration de la lisibilité du code permettra une compréhension plus rapide de celui-ci à l'avenir. Une des particularités de notre démarche de restructuration tient donc au fait qu'elle rendra toute restructuration ultérieure plus aisée et efficace.
- **Un début de généricité du système :** sans prétendre (loin de là) créer un système expert générique, la nouvelle architecture permettra de modifier aisément le système afin de l'adapter à un autre domaine, d'autres utilisateurs ou une autre logique d'affaires.
- **De meilleures performances :** le temps de réponse devrait être amélioré par la nouvelle architecture physique.
- **Une sécurité accrue :** aucun traitement de la logique d'affaires ne sera plus accessible du côté client, seul les résultats de ceux-ci apparaîtront.

3.2.4.3 Présentation de l'architecture logique abstraite

Dans cette dernière sous section, nous présentons l'architecture logique de SMXpert. La clarification de l'architecture de COSMICXpert est présentée dans l'Annexe 1. L'architecture physique et son implémentation feront, quant à elles, l'objet du chapitre suivant. Pour tous détails supplémentaires, le lecteur pourra se référer à l'Annexe 7.

La Figure 3.17 illustre l'architecture globale de SMXpert. On y distingue deux types d'acteurs : les utilisateurs et le système à base de connaissances (SBC). Chaque type d'utilisateur dispose d'une interface qui lui est propre (cf. Chapitre 2). Ces interfaces ne sont que des vues permettant d'interagir avec le système. La seule complexité à ce niveau provient du fait que chaque interface est responsable de la correction syntaxique des données qu'elle fournit au système, d'où l'existence d'un module *validation syntaxique*.

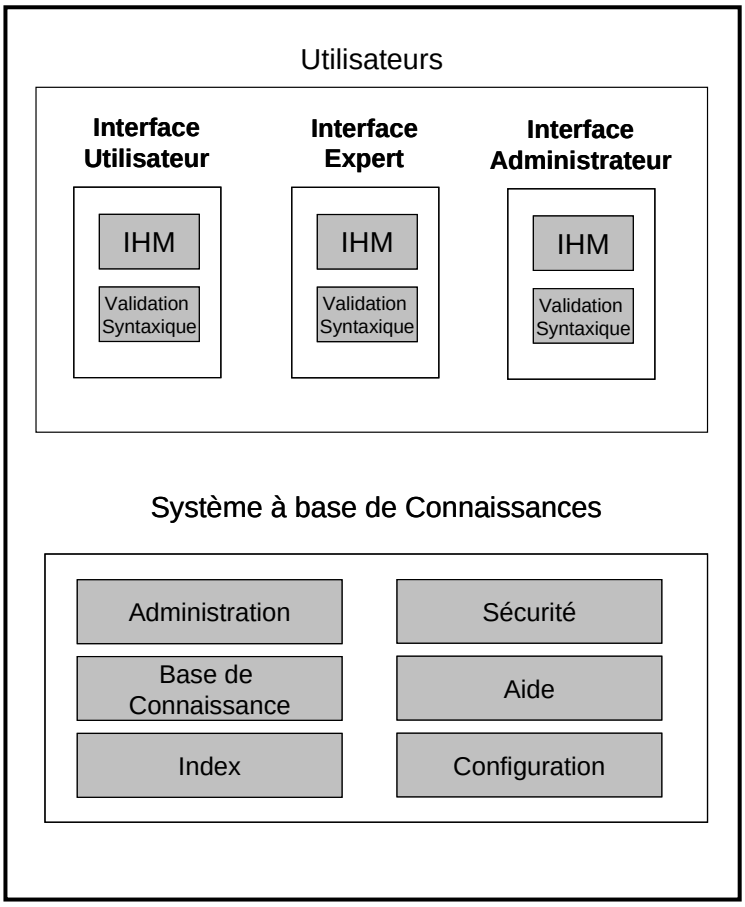


Figure 3.17 : Schéma de l'architecture logique de SMXpert.

Le **SBC**, quant à lui, comporte six composants dont il convient de détailler les rôles et les tâches dans le fonctionnement du système. La Figure 3.18 illustre ces six composants.

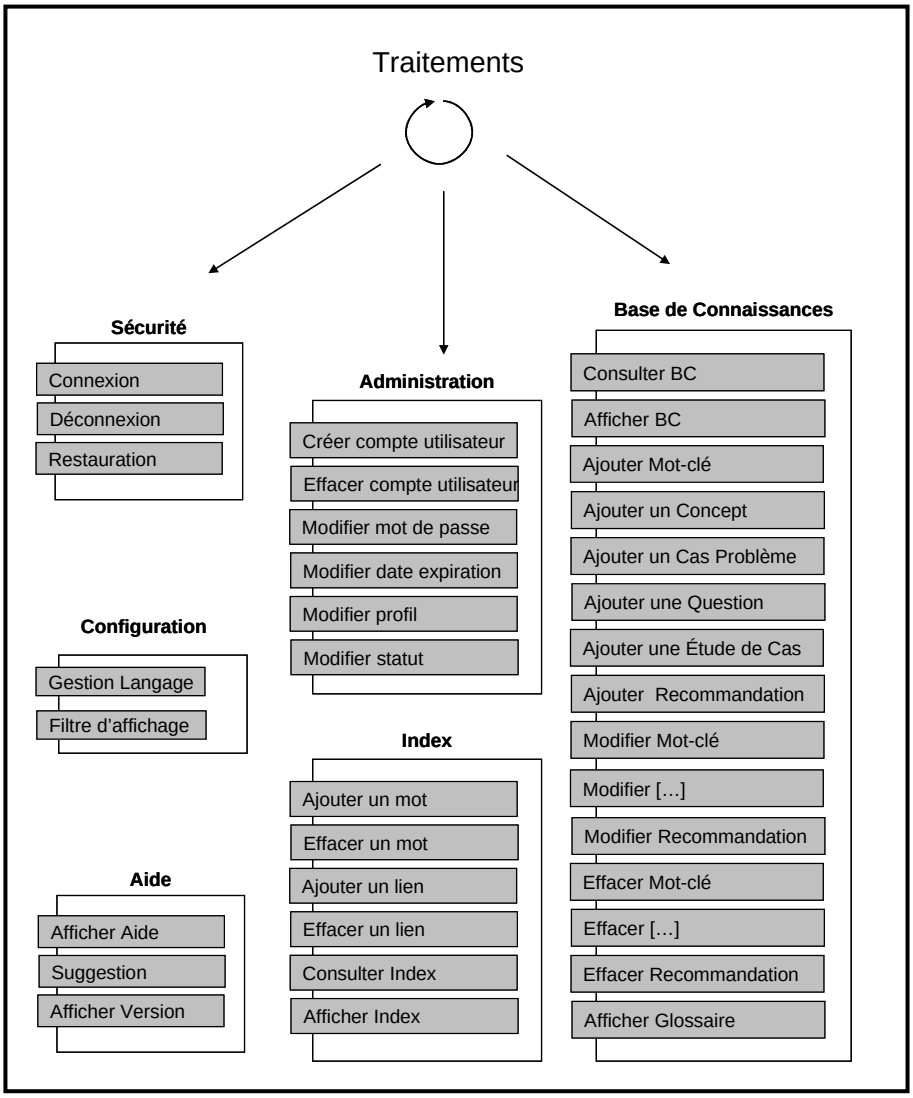


Figure 3.18 : Schéma des composants du SBC et leurs tâches

- **Le module Sécurité :** regroupe toutes les fonctionnalités garantissant une certaine sûreté du système. On y trouve la gestion des sessions, ainsi que des connexions et déconnexions à ces dernières. Une fonctionnalité de restauration de la base de connaissances est aussi intégrée à ce module, afin de garantir une dimension de récupération de désastres.
- **Le module Configuration :** englobe les fonctionnalités permettant de paramétrer certaines caractéristiques du système. Les caractéristiques paramétrables du logiciel sont la langue dans lequel les instructions et menus apparaissent et le filtre utilisateur (cf. Section 3.2.2).

- **Le module Aide :** regroupe les fonctionnalités propres à guider l'utilisateur dans sa compréhension du logiciel.
- **Le module Administration :** regroupe toutes les fonctionnalités liées à la manipulation des comptes utilisateur. Celles-ci couvrent notamment l'ajout et la suppression d'un compte. De plus, le module regroupe les fonctionnalités de *modification* desdits comptes.
- **Le module Index :** regroupe les fonctionnalités propres à l'ajout et la suppression d'un mot de l'index. Il renferme également les opérations de lien entre lesdits mots et les mots-clés vers lesquels ils sont sensés orienter l'utilisateur.
- **Le module Base de Connaissances :** le plus important et le plus conséquents de modules du **SBC.** Il regroupe en effet les fonctionnalités liées à la manipulation des concepts de l'ontologie de SMXpert. On retrouve donc dans ce module les fonctionnalités d'ajout, de suppression et de modification des concepts de maintenance, mot-clés, cas problèmes, questions (incluant les **faits** qui lui sont associés) et recommandations. Ce module renferme également les opérations de consultation de la base de connaissances, ainsi que les fonctionnalités d'affichage de celle-ci[7].

Certaines fonctionnalités présentes dans les modules ci-dessus sont inexistantes dans COSMICXpert. Ces dernières ont vu le jour sur base des exigences propres au fonctionnement de SMXpert et les sections suivantes les abordent avec plus de détails. Pour plus d'informations sur les fonctionnalités non spécifiques à SMXpert, le lecteur pourra se référer à [21], ainsi qu'en Annexe 2.

[7] La différence entre consultation et affichage de la base de connaissances peut paraître ténue mais se justifie pourtant pleinement. Par *consultation*, nous entendons toute opération nécessitant une action de l'utilisateur produisant un résultat tandis que nous définissons *affichage* par la simple mise en forme des connaissances de la base.

Chapitre 4 : Application de la démarche architecturale[8]

Suite à la création de l'architecture logique, il faut définir une architecture physique qui est en accord avec la première. Comme expliqué précédemment, le but de la création de SMXpert n'est pas seulement le développement d'un système à base de connaissances permettant de prendre des décisions en rapport aux processus de maintenance, mais également l'amélioration de la qualité du logiciel de départ. Dans cette optique, nous avons privilégié la réutilisation des mêmes technologies (servlets, JSP dans le cadre d'une plateforme Java) et également le maximum du code source. Nous verrons par la suite les résultats en rapport à l'amélioration et à la réutilisation effective.

4.1 Définition d'une application Web

Une application Web est un programme situé sur un serveur Web. Ce dernier produit des pages statiques et dynamiques dans un langage de balises (généralement du HTML) en réponse à la requête d'un utilisateur. Lorsqu'une application Web utilise des technologies comme les servlets et le langage JSP, la conjonction des composants formant celle-ci respecte l'architecture physique globale de la Figure 4.1.

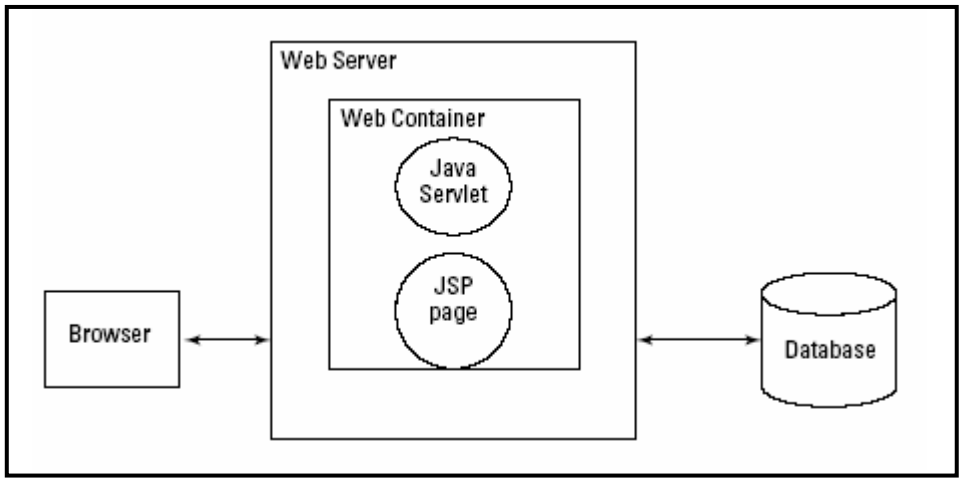


Figure 4.1 : Architecture physique d'une application web [54]

[8] Les principales références pour ce chapitre sont les livres [52] [53] [54]. Les sites Internet contenus dans la section « Liens Chapitre 4 » des références ont également été d'une grande utilité.

Nous sommes donc partis de l'application existante **COSMICXpert Web Edition**. Celle-ci étant une application Web, il était logique que SMXpert le soit aussi vu l'optique de restructuration. Comme nous l'avons vu dans la revue de littérature du Chapitre 1, le *design pattern* qui convient le mieux pour une application Web correspond au « Model – View – Controller ».

4.2 Outils et technologies utilisées :

Avant d'aborder tout autre sujet en rapport à SMXpert, nous allons présenter les différentes technologies et les outils utilisés. Cette démarche permettra de convaincre le lecteur de l'utilité de ceux-ci dans le développement d'une application Web, mais aussi de lui présenter sommairement les concepts qui entrent en jeu.

4.2.1 Technologies

4.2.1.1 Java

COSMICXpert ayant été programmé en Java (cf. Section 8.1 dans l'Annexe 8 pour plus de détails sur le langage), nous avons choisi ce langage également pour pouvoir favoriser la réutilisation du plus grand nombre de classes. De plus, les avantages exposés dans la Section 8.1 de l'Annexe 8 et le fait qu'il soit fortement répandu font de lui le meilleur choix. Nous avons donc utilisé le Java 2 Standard Edition Development Kit v1.5.0_03 pour le développement de SMXpert.

4.2.1.2 Java Servlets

Avant le milieu des années 90, les pages Web étaient statiques. Le besoin de générer des pages de manière dynamique est apparu, malgré tout, assez rapidement. Cela donna naissance au « Common Gateway Interface » standard (standard CGI) qui est un programme permettant au serveur Web (et non « Web Container ») d'interagir avec des applications externes. Un programme CGI permet par exemple d'effectuer une requête auprès d'une base de données et d'insérer les résultats dans une table d'une page Web.

Cependant, des limitations inhérentes aux applications CGI ne les rendaient pas très efficaces dans un environnement où plus d'un utilisateur à la fois effectuait des requêtes de pages dynamiques. En effet, ces applications ne peuvent satisfaire qu'une seule requête à la fois car elles sont externes au serveur Web et donc il deviendrait très compliqué de gérer la connexion, les autorisations d'accès, … alors que le précédent s'en charge. Pour pallier à cette situation, chaque concepteur de serveur Web avait décidé de développer des solutions propriétaires (mod_perl pour Apache, NSAPI pour Netscape et ISAPI pour Microsoft).

C'est dans ce contexte que les Java Servlets ont émergé. En effet, ils sont programmés en Java, ce qui leur donne l'avantage de s'exécuter sur des systèmes différents et de pouvoir profiter de l'ensemble des « Application Programming Interfaces » (APIs) qui sont accessible au langage (JDBC, JDOM, JUnit, …). De plus, la « Java Virtual Machine » (JVM) permet au servlet de créer un thread par requête d'un utilisateur, ce qui permet le multi-threading. En d'autres termes, avec chaque requête associée à un thread différent, chaque utilisateur peut être servi en même temps.

Un Java Servlet est associé à un ou plusieurs « Uniform Ressources Locators » (URLs). Lorsque le serveur reçoit une requête, il analyse l'URL demandé et la transmet au servlet adéquat. Cette transmission se fait par l'appel à la méthode « service » de ce dernier.

Le résultat du traitement de la requête par le Java Servlet correspond au code HTML (HyperText Markup Language) qui va être transmis à l'utilisateur pour qu'il puisse l'afficher. Ce code est littéralement généré. Cette manière de procéder demande un haut niveau d'abstraction car il faut pouvoir prévoir toutes les pages qui pourraient être affichées, ce qui est beaucoup moins pratique qu'un document dans lequel est enregistré le code HTML connu. Par exemple, à chaque fois que l'on désire modifier l'affichage final d'une requête, il faut effectuer les modifications directement dans le code source du servlet, ce qui entraînait également de devoir recompiler. Ce genre d'opération devient rapidement fastidieux lorsqu'il ne s'agit que de modifications en rapport à l'esthétique ou à l'internationalisation (support de plusieurs langues). De plus, cela ne permet pas de profiter de la spécialisation de certains métiers comme les graphistes qui n'ont pas nécessairement suivi de cours d'algorithmique et de programmation pure. Ce dernier désavantage est une manifestation d'un problème bien plus grave pour les ingénieurs logiciels ayant décidé d'architecturer l'application Web avec le

design pattern MVC. En effet, le servlet générant le langage de balises peut être comparé au contrôleur qui générerait la vue, ce qui les couple fortement contrairement à l'idée même du modèle.

Dans le but de résoudre les problèmes ci-dessus, les JavaServer Pages ont été créées. Cette technologie permet d'éviter de programmer la génération d'un langage de balises par le servlet, ce qui rend ce dernier plus clair. Maintenant, le Java Servlet peut se consacrer plus amplement au traitement et à la gestion de la requête. Grâce aux pages JSP et dans l'optique du *design pattern* MVC, la vue est bien scindée du contrôleur.

4.2.1.3 JavaServer Pages

Les JavaServer Pages sont des documents (semblables aux pages HTML) qui sont formés de code HTML (partie statique de la page) de zones spécifiques réservées à la programmation en Java (et donc, partie dynamique de la page). Ces zones sont identifiées par des symboles (« <% », « %> ») ressemblant à ceux qui forment les tags XML. Les pages JSP peuvent également contenir des « tags » permettant de réduire la longueur de leur code source (cf. Section 8.2 dans l'Annexe 8 pour plus de détails). râce à un traducteur, les pages JSP subissent un traitement préalable à leur utilisation qui consiste à les transformer en fichiers Java. Ensuite, ces derniers fichiers sont compilés et forment des fichiers « .class » prêts à être exécutés par le « Web Container ». On peut voir ceux-ci comme des servlets car ils ont pour but de traiter un type de requête bien précis.

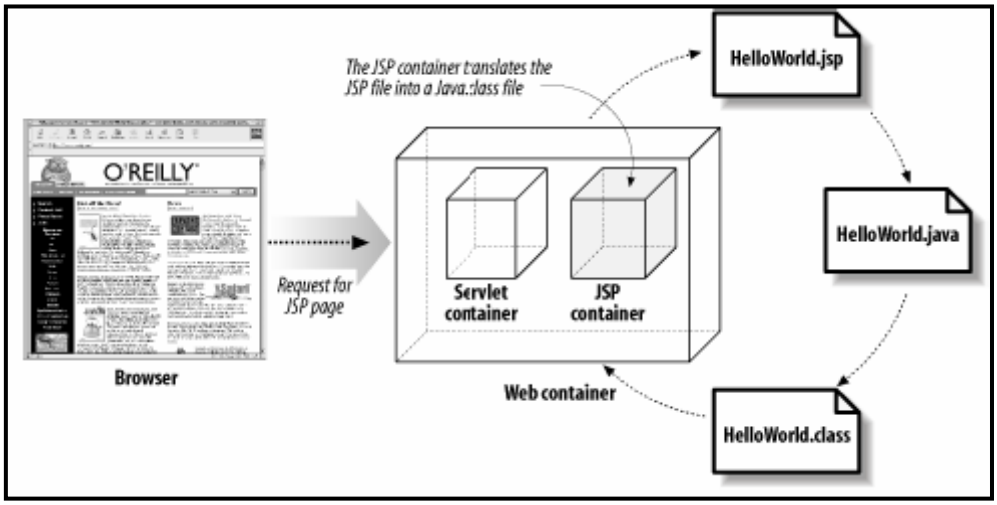


Figure 4.2 : Une page est traduite et compilée en un java servlet [52]

Les traitements sur les pages JSP, évoqués précédemment, sont effectués lors du démarrage du « Web Container » ou lorsqu'elles sont modifiées alors que le serveur est en cours d'exécution. Cela permet une vitesse d'exécution plus élevée du traitement des requêtes dans la pratique.

Tout comme les Java Servlets, le fait de pouvoir introduire du Java permet d'élargir les possibilitiés offertes par les pages JSP. En effet, l'accès à l'ensemble des APIs est un avantage non négligeable. De plus, le Java et les langages de balises ne sont pas dépendants d'une plateforme particulière, ce qui élargit considérablement le nombre d'utilisateurs pouvant en disposer.

4.2.2 Outil : Apache Tomcat

Cmme nous l'avons vu dans les schémas précédents (cf. Fig 4.1 et 4.2), une application Web a besoin d'un Web Container pour pouvoir offrir ses services. Apache Tomcat est un de ceux-ci, c'est-à-dire un programme qui gère la sécurité de l'accès, la concurrence, l'exécution et le cycle de vie des composants. Ce sont des services de base dont l'ingénieur logiciel n'aura pas à se préoccuper dans le développement de son application (cf. Section 8.3 dans l'Annexe 8 pour plus de détails).

4.3 Architecture physique de la base de connaissances

Comme nous l'avons vu dans le Chapitre 3, l'architecture logique de la base de connaissances n'a presque pas évolué. Il en est de même du point de vue physique. En effet, nous utilisons la même structuration des fichiers « *.xml ».

Cependant, si comme François Gruselin et Julien Viltz [2, p107-109], on représente les liens entre les différents types de fichiers. Nous pouvons observer qu'un nouveau lien a été créé dans le cadre de la révision du moteur d'inférence suite aux entrevues avec le Professeur April. Le moteur d'inférence en question ne fonctionne plus avec des pourcentages mais les questions (du concept) sont fortement liées aux recommandations (cf. Chapitre 3). Lorsque l'utilisateur répond « Non » à une question, on doit aller voir, dans la question, vers quelle

recommandation se diriger. Le lien se situe, donc, entre les fichiers représentant les concepts de maintenance et ceux représentant les cas problèmes. En fait, chaque question ou concept fait maintenant référence à une recommandation de chaque cas problème qui lui est également lié.

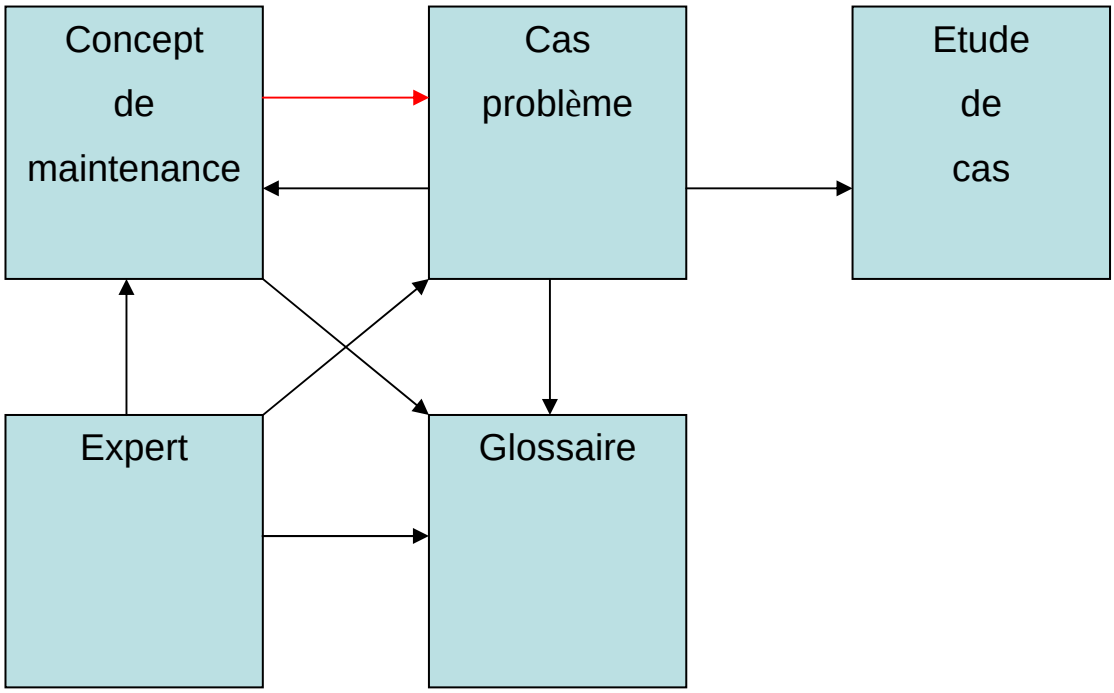


Figure 4.3 : Liens entre les différents types de fichiers de la base de connaissance

Pour des raisons pratiques (expliquées à la Chapitre 3), nous avons scindé le fichier « glossary.xml » en deux fichiers avec les mots-clés d'un côté (« glossary_KW.xml ») et les entrées d'index de l'autre (« glossary_INDEX.xml »). La disproportion entre le nombre d'entrées d'index (plus de 500) et celui de mots-clés (plus de 70) explique cette différenciation pour des raisons de clarté.

4.4 Architecture logique concrète et physique de l'application Web

A partir de l'architecture logique présentée dans le Chapitre 3, il est nécessaire d'effectuer une première itération de concrétisation qui va fournir l'architecture logique concrète. Cette itération a pour but d'exprimer les choix de stratégies globales sur la communication entre composants, d'interactions avec le domaine et plus particulièrement avec les utilisateurs.

Pour obtenir cette architecture logique concrète, nous allons structurer les composants abstraits de l'architecture logique selon le « Model – View – Controller » *design pattern*. En effet, il est ressorti de la revue de littérature que celui-ci était une bonne base pour la structuration de la plupart des applications. Les explications du « Model – View – Controller » *design pattern* sont présentées ci-après (cf. Section 4.4.1).

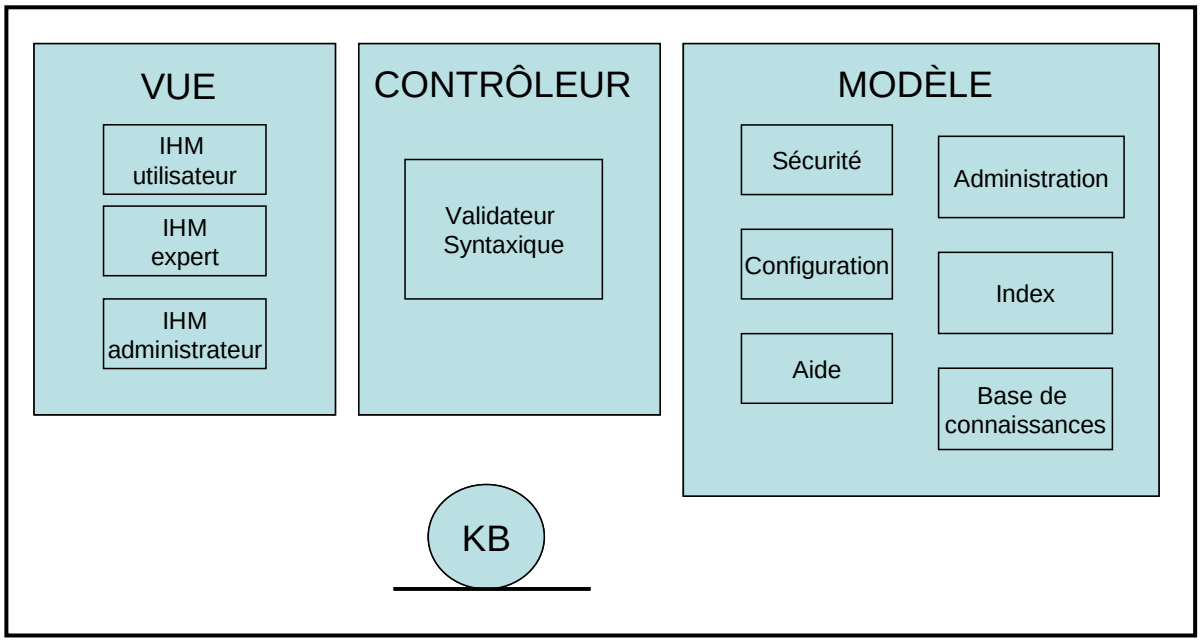


Figure 4.4 : Architecture logique concrète de SMXpert

Dans cette précédente figure, nous pouvons remarquer que le composant « Validateur Syntaxique » se situe au niveau de la couche contrôleur. Du fait de cette position stratégique, nous espérons obtenir un gain de performances. En effet, cela évite de faire transiter des données erronées entre le modèle et la vue (ce qui constituerait un gaspillage de ressources).

L'architecture logique concrète peut être considérée stable, quels que soient les évolutions techniques qui seront mises en œuvre lors du cycle de vie de développement du logiciel. Par contre, l'architecture physique est fortement dépendante des solutions technologiques. En effet, un changement dans celles-ci pourrait engendrer une révision de l'architecture physique. Par exemple, si la couche de persistance consiste simplement en un ensemble de fichiers « *.xml » stockés au même endroit que la logique d'affaires de l'application Web, alors le fait d'introduire un système de gestion de base de données va engendrer la création d'un troisième « tier ».

La figure ci-dessous représente l'architecture physique de SMXpert. C'est une architecture « multi-tiers » composée de deux tiers, c'est-à-dire d'une séparation physique des composants :

- Le « Client-tier » correspond au navigateur utilisé et à l'ensemble des pages qui constituent la vue. Il offre la possibilité à l'utilisateur d'interagir avec l'application Web. C'est également lui qui envoie des requêtes au « Server-tier », qui récupère les réponses et les communique à l'utilisateur.

- Le « Server-tier » contient l'application en tant que tel, ainsi que les mécanismes qui permettent une communication avec le « Client-tier ». Il traite les requêtes de ce dernier et lui renvoie les réponses. Il gère, également, le flux des pages qui vont être affichées, et les accès à la base de connaissances.

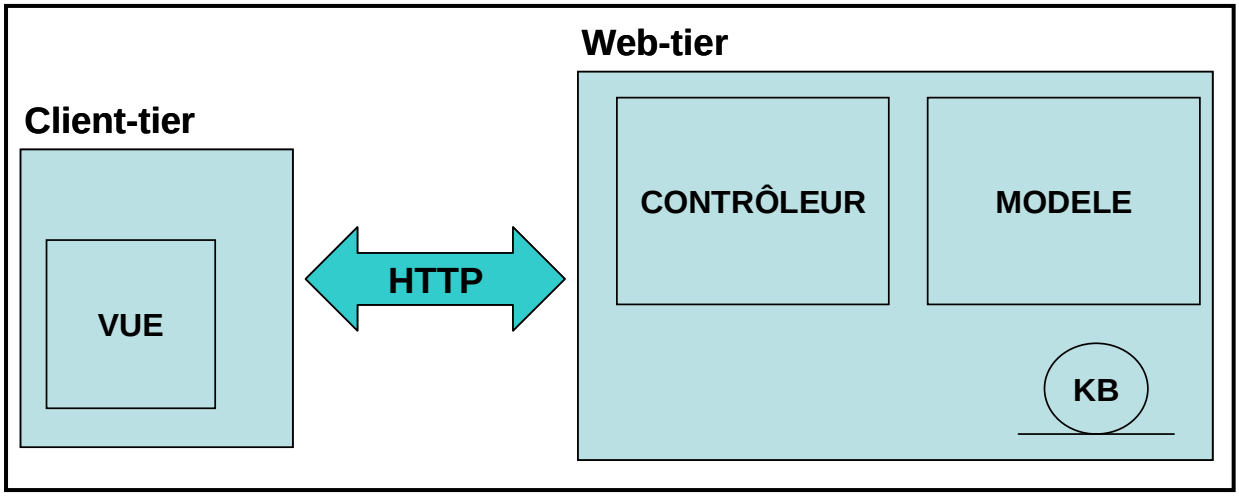


Figure 4.5 : Architecture physique de SMXpert

4.4.1 Le « Model – View – Controller » design pattern

Le *design pattern* MVC est l'ancêtre des *design patterns* orientés objet. Il a été créé dans les années 80 par Xerox PARC pour SmallTalk-80.

Une application architecturée sur base de ce *design pattern* est composée de trois couches :

- Modèle : les données (dont le stockage) et l'implémentation de la logique d'affaires

- Vue : la présentation des données
- Contrôleur : le contrôle du flux de données

Le modèle englobe les données et la gestion de la persistance de celles-ci (en général : connexion avec la base de données, ajout, suppression et modification de données), ainsi que la logique d'affaires qui correspond à tous les traitements que l'application permet sur les données précédentes.

La vue correspond à l'affichage des données du modèle (brutes ou après traitements). Elle prend également en charge les données introduites par l'utilisateur dans les différentes interfaces en les dirigeant vers le contrôleur. Les requêtes de l'utilisateur et les réponses sont donc du domaine de la vue.

Le contrôleur est constitué d'*handlers* qui vont récupérer les requêtes des utilisateurs, les analyser pour savoir quels sont les traitements adéquats du modèle à exécuter et, finalement, appeler ces derniers. Par après, il sélectionne la vue dans laquelle se fera l'affichage des résultats et la renvoie à l'utilisateur. Le contrôleur est donc le seul point d'entrée des requêtes et il encourage la réutilisation des éléments des autres couches du modèle, contrairement aux modèles dans lesquels la vue et le contrôleur sont fortement couplés (voire une seule et même entité.).

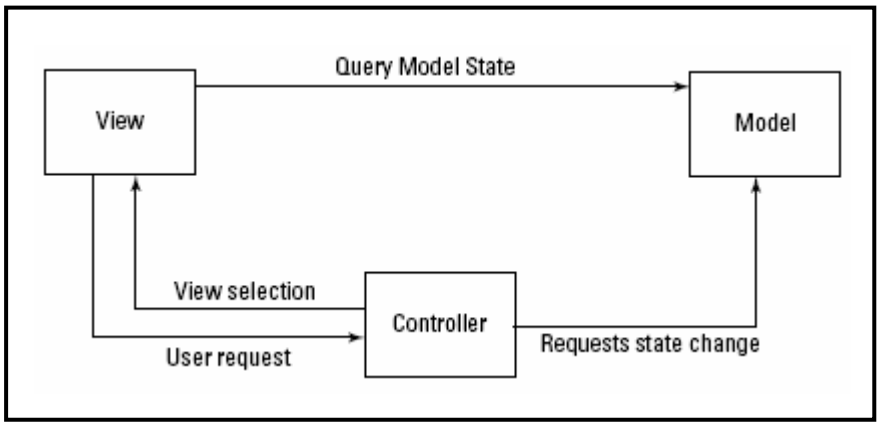


Figure 4.6 : Représentation des interactions entre les différentes couches du *design pattern* MVC [54]

Du point de vue des fonctionnalités, ces éléments forment trois niveaux différents qui s'enchaînent. Les interconnexions entre les couches se limitent au strict minimum pour éviter les dépendances. Ce dernier point assure qu'il est possible de modifier une couche sans devoir se préoccuper des répercussions sur les deux autres.

L'utilisation du *design pattern* MVC n'est pas obligatoire pour une application Web (mais est fortement recommandée). Dans ce cas de figure, nous pourrions imaginer que le programme soit un ensemble de pages JSP, chacune contenant une partie de la logique d'affaires. Cela peut sembler plus simple, mais ce n'est qu'une impression. En effet, une simple modification aurait des répercussions sur les traitements des autres pages JSP. Il est facile d'imaginer une métaphore représentant ces difficultés : une glace constituée de 3 boules de parfums différents permet de facilement remplacer un parfum par un autre, tandis qu'une glace dont les 3 parfums sont mélangés rend difficile voire impossible le changement de sa composition.

Selon Mike Robinson [54], les avantages du *design pattern* MVC sont :

- Une plus grande flexibilité au niveau des vues : Il est facile d'ajouter différents types de vues (HTML, JSP, …).
- Une plus grande flexibilité au niveau de la persistance : Il est facile de changer la façon dont on stocke les données car le modèle y accède généralement via une interface comme un bean. Les données peuvent être stockées dans une base de données ou, comme dans notre cas, dans des fichiers XML. Nous comprenons donc que l'évolution pourra se faire à moindre frais (en modifications) lorsque l'ampleur des données demandera l'utilisation d'une base de données.
- Meilleure utilisation de différentes compétences : Les designers peuvent se focaliser sur la vue et les programmeurs sur le modèle et/ou le contrôleur. Les interactions entre les différents métiers se font via des interfaces spécifiées dès le départ.
- Facilité de maintenance et d'extensibilité des fonctionnalités : Les structures et les flux des données sont clairement définis. Ceci est fortement renforcé par le fait que les différentes couches sont peu couplées. Au final, les différents composants de l'application Web ont des responsabilités claires et distinctes ce qui permet à l'ingénieur logiciel extérieur d'acquérir, plus facilement, une compréhension des fonctionnalités développées.

L'extensibilité des fonctionnalités peut être liée à la maintenance et donc, des avantages, dus à l'utilisation du *design pattern*, se font ressentir à ce niveau également. Par exemple, nous pourrions souhaiter sécuriser les accès à l'application. Dans le *design pattern* MVC, il suffirait de modifier le contrôleur (un seul composant, en général, dans lequel sont analysées toutes les requêtes des utilisateurs). Tandis que dans un modèle où la vue serait fortement couplée avec le contrôleur, il faudrait modifier chaque composant de la vue (chaque page).

Au vu de ces avantages, il était logique pour nous de se tourner vers ce *design pattern*. Ce qui nous a paru le plus important est la facilité de maintenance qu'apporte le modèle MVC associé à une bonne documentation (cf. Chapitre 3).

4.5 Mise en place de l'architecture physique

Dans cette partie, nous allons expliciter l'architecture physique de SMXpert et donner des explications détaillées sur son fonctionnement. Pour cela nous procèderons en deux étapes.

Premièrement, nous allons présenter les mécanismes de base qui permettent à l'application d'acquérir une certaine dynamique. Cela correspond surtout à la mise en place de la communication entre les différentes couches définies dans l'architecture logique concrète. Le cadre de référence Struts est l'outil principal de l'application correcte du « Model – View – Controller » *design pattern*. En effet, il permet de structurer l'application de telle manière que celui-ci soit mis en œuvre.

Deuxièmement, nous présenterons les fonctionnalités de SMXpert, plus précisément la logique d'affaires. Cela se traduit par une présentation des grands types de produits de l'implémentation (classes java, pages jsp, …), reposant sur le cadre de référence Struts, et leur structuration. Les mécanismes de base ont été abordés en détails dans la première partie, il n'est plus utile de les représenter dans cette seconde partie. En effet, la façon dont communiquent les différentes couches (cf. précédent paragraphe) est, à quelques détails près, le même quelle que soit l'application qui sera structurée sur base du « Model – View – Controller » *design pattern*.

4.5.1 Les mécanismes de base : Le cadre de référence Struts

4.5.1.1 Présentation

Struts est un cadre de référence « open-source » permettant la création d'applications Web sur base du langage de programmation Java et utilisant des technologies telles que les Java Servlets et les JavaServer Pages (JSP). L'historique du cadre de référence est présenté dans la Section 8.4 de l'Annexe 8.

L'utilisation d'un cadre de référence apporte un plus dans le développement d'un logiciel. En effet, il permet de développer plus rapidement une application et d'en diminuer le nombre de bogues. Cela s'explique par le fait qu'une telle plate-forme intègre des mécanismes de base qui sont déjà programmés et éprouvés. Dans sa plus simple condition, un cadre de référence est un ensemble de classes et d'interfaces qui coopèrent dans le but de résoudre un problème logiciel particulier.

Selon Chuck Cavaness [52] :

A good framework should provide generic behavior that many different types of applications can make use of.

A framework has the following characteristics:

• *A framework comprises multiple classes or components, each of which may provide an abstraction of some particular concept.*

• *The framework defines how these abstractions work together to solve a problem.*

• *Framework components are reusable.*

• *A framework organizes patterns at a higher level.*

Dans le cas de Struts, il est l'équivalent des fondations du programme (d'où son nom, cf. Section 8.4 dans l'Annexe 8). Des fondations solides (en termes informatiques : mécanismes de base efficaces, robustes et sans bugs) procurent de réels avantages pour ceux qui souhaiteraient créer une application Web de par l'économie de temps et d'argent qui en résulterait. En effet, l'ingénieur logiciel ne fait, dans un premier temps, que configurer le squelette et, donc, peut se focaliser sur la programmation des fonctionnalités propres au programme en devenir (logique d'affaires).

Struts structure et unifie les interactions des composants de l'application Web de manière bien définie. Il réussit grâce à l'implémentation du *design pattern* vu plus haut : le « Model-View-Controller » (MVC). Les composants entrant en jeu sont :

- Les servlets Java : Programmes écrits en Java et qui répondent aux requêtes des utilisateurs.

- Les pages JSP : Langage dans lequel sont exprimées les pages affichées et qui permet d'utiliser du Java pour présenter du contenu dynamique à l'utilisateur.
- Les Beans : Composants qui correspondent à un type ou une collection d'objets et qui comprennent les traitements applicables à ceux-ci.
- La Logique d'affaires : Le code qui implémente les fonctionnalités spécifiées par le client.

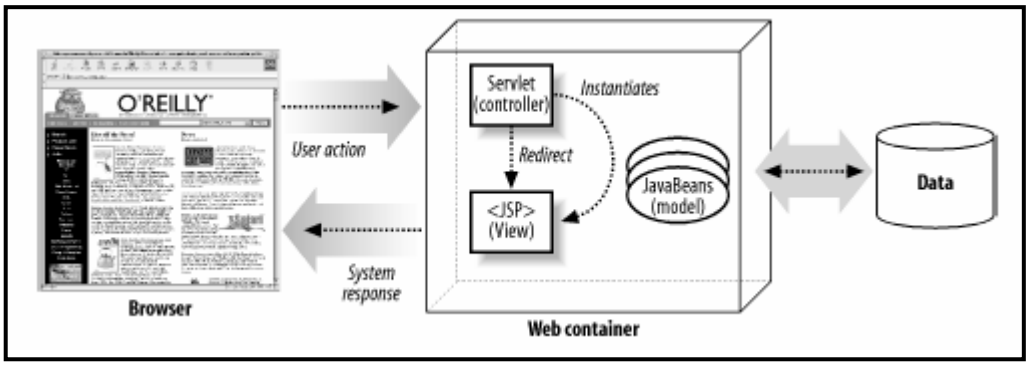


Figure 4.7 : Architecture du modèle JSP 2 [52]

Lorsque l'on déploie Struts et que l'on y ajoute les différents composants en respectant bien les fonctions qu'ils sont sensés avoir, alors nous sommes certains que le *design pattern* est respecté.

4.5.1.2 Intérêts de Struts dans notre démarche

Comme nous l'avons vu, le but premier du travail effectué est d'améliorer la qualité d'un système à base de connaissances (COSMICXpert) tout en modifiant le domaine de celui-ci (du domaine de la mesure au domaine des processus de maintenance logicielle). Nous avons plus particulièrement essayé d'apporter une amélioration en ce qui concerne la qualité de la maintenabilité.

Après avoir analysé tous les documents accompagnant COSMICXpert [17] [2] [3] ainsi que le programme en lui-même, nous avons déduit que l'architecture pouvait être fortement améliorée. En effet, à aucun moment, nous n'avons trouvé d'explications claires et précises sur la structure, tant logique que physique, du programme. Une vérification du code source nous a confirmé le haut niveau de couplage entre les différentes couches qui pourraient former une architecture de type MVC (cf. exemple Annexe 4).

De plus, après avoir effectué des tests sur COSMICXpert (cf. Annexe 5 contenant le plan de tests), nous avons évalué que l'application était fonctionnelle à 70% par rapport à ses cas d'utilisation. Toutes les fonctionnalités de COSMICXpert ayant été implémentées à partir de rien, la probabilité d'erreurs dans les mécanismes de base n'est pas à négliger.

Les avantages de Struts ayant été présentés précédemment (cf. Section 4.5.1.1), nous comprenons qu'il peut participer à la diminution des bogues dans les mécanismes de base et donc pour l'amélioration de la qualité. Deuxièmement, le *design pattern* MVC convient parfaitement pour les applications Web dès que l'on désire structurer un site offrant de multiples fonctionnalités. Une meilleure architecture rime également avec une meilleure maintenabilité.

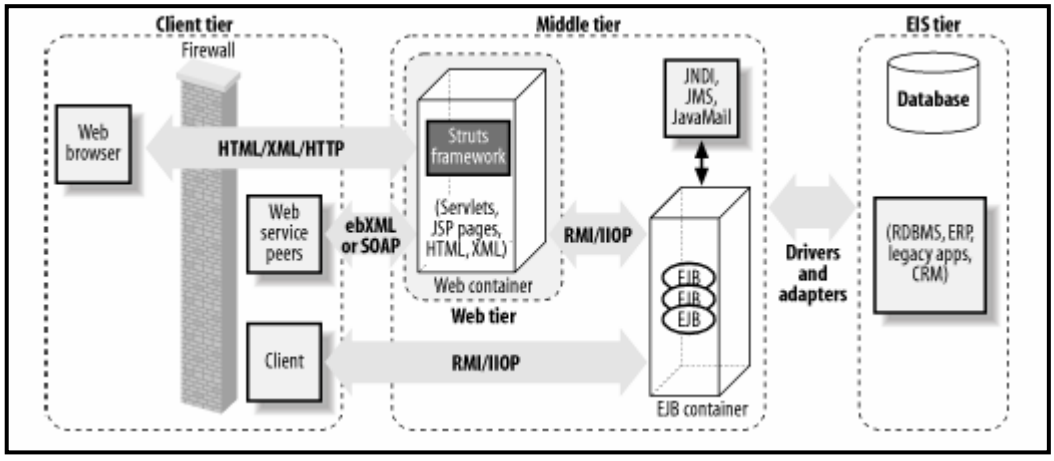


Figure 4.8 : Le cadre de référence Struts se situe au niveau du web-tier [52]

4.5.1.3 Struts en pratique

Struts fournit une implémentation concrète du contrôleur du *design pattern*, ainsi que des mécanismes qui interconnectent les différentes couches et leur permettent de communiquer entre-elles. Struts est composé de près de 300 classes Java, regroupées dans 8 *packages* différents (cf. Section 8.5 dans l'Annexe 8 pour plus de détails). Du fait de la rapidité avec laquelle a évolué le cadre de référence (cf. Section 8.4 dans l'Annexe 8), la façon dont sont organisés les packages n'est pas représentative du *design pattern* MVC. En effet, en plus de retrouver les classes du contrôleur dans le package « action », on y retrouve aussi des classes liées à la vue.

Pour ce qui est de la vue et du modèle, Struts ne fixe pas de contraintes en ce qui concerne la façon de structurer le modèle d'une application particulière ou le paradigme utilisé pour générer les vues.

4.5.1.4 Le contrôleur de Struts

Le contrôleur de Struts est donc le point névralgique permettant la mise sur pieds du *design pattern* MVC. En effet, il est principalement responsable de la récupération des requêtes des clients, de l'appel des classes adéquates du modèle et du choix de la vue qui sera renvoyée en réponse à l'utilisateur. Il est composé de différentes classes :

ActionServlet (org.apache.struts.action.ActionServlet) : C'est le Java Servlet qui dirige toutes les requêtes des utilisateurs vers les handlers appropriés pour qu'elles soient traitées (cf. Action).

L'ActionServlet étend la classe javax.servlet.http.HttpServlet, ce qui en fait un servlet utilisable par l'application Web. Il est important de savoir que le servlet dépend d'une autre classe de type RequestProcessor (cf. RequestProcessor ci-après) qui est appelée à chaque requête.

Choix de l'Action :

La décision de quelle Action mettre en œuvre se fait sur base du chemin demandé par la requête. En effet, le chemin demandé est différent lorsque qu'il s'agit de déclencher un traitement que de demander une page JSP. Cela se joue au niveau de l'extension du fichier demandé. Pour un page JSP, ce sera un fichier dont l'extension sera « *.jsp » tandis que pour une Action ce sera une extension définie dans le fichier de configuration « web.xml ».

```
<!-- Action Servlet Mapping -->
<servlet-mapping>
          <servlet-name>action</servlet-name>
          <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Comme on peut le voir dans l'exemple précédent (provenant du fichier « web.xml » de SMXpert), le contrôleur saura que lorsque la vue demande un fichier avec l'extension « *.do », il va devoir exécuter une Action.

Après avoir déduit qu'il fallait exécuter une Action, le contrôleur va récupérer dans le chemin de la requête le nom du fichier avec l'extension « *.do ». Supposons que la requête provenant de la vue contienne toutes les informations pour la suppression d'un mot-clé, alors elle va demander le fichier « delkw.do ». A partir de ceci, le contrôleur peut trouver précisément quelle classe de type Action il va devoir appeler. En effet, « delkw » correspond à l'identifiant (attribut « path » dans la balise action) d'un « action mapping » définissant l'Action à exécuter et présent dans le fichier de configuration de Struts « struts-config.xml » (cf. Annexe 6). L'exemple suivant illustre un « action mapping » :

```
<action     path="/delkw"
            type="application.KBase.DeleteKeywordAction"
            name="delkwForm"
      scope="request"
            input="/pages/expert/delete.jsp"
            validate="true">
            <forward    name="failure_deletekeyword_notexist"
                        path="/pages/xpert/error.jsp"/>
            <forward    name="failure_deletekeyword_last"
                        path="/pages/xpert/error.jsp"/>
            <forward    name="failure_deletekeyword_nokws"
                        path="/pages/xpert/error.jsp"/>
      <forward    name="ok_del"
                        path="/pages/xpert/success.jsp"/>
</action>
```

Dans le fichier de configuration (localisé dans le répertoire WEB-INF de l'application Web), un ensemble d'« action mappings » sont présents, c'est-à-dire des balises « action » avec un certain « path ». Il y en a généralement un pour chaque fonctionnalité offerte par l'application Web. Il contient des informations sur l'Action (attribut « type ») et l'ActionForm (attribut « name ») qui doivent être utilisés avec la requête et les données qu'elle contient.

L'« action mapping » fournit également des informations aux contrôleurs pour le moment où il devra prendre une décision sur la vue à renvoyer à l'utilisateur (attributs « forward »). Ce mécanisme sera explicité plus loin dans le document.

Tous les « action mappings » sont chargés dans la mémoire au démarrage de l'application et sont, donc, accessibles au cadre de référence à tout moment de l'exécution.

RequestProcessor (org.apache.struts.action.RequestProcessor) : Cette classe traite la requête pour l'ActionServlet. Ce traitement de la requête hors *servlet*, permet à l'ingénieur logiciel de créer une classe étendant le RequestProcessor et qui, de ce fait, l'autorise à modifier la façon dont sont traitées les requêtes. En effet, cela permet de garder les mécanismes qui font la spécificité du *servlet* comme la récupération de la requête et de la réponse à celle-ci en sélectionnant la vue appropriée.

Le RequestProcessor vérifie, éventuellement, les droits d'accès au lien demandé par l'utilisateur et/ou peut effectuer d'autres vérifications comme la validité de sa session (expirée ou pas, …). Ensuite, si toutes les vérifications se passent bien, elle va garnir les champs d'un bean qui correspond à la requête (cf. ActionForm ci-après dans la vue).

Action (org.apache.struts.action.Action) : Les classes de type Action permettent la communication entre le contrôleur et le modèle. L'Action a la responsabilité de décider et d'effectuer les enchaînements d'appels à la logique d'affaires (présente dans la couche modèle). En d'autres termes, l'Action constitue un *handler* de par les choix qu'il doit poser et de par le lien avec la couche modèle.

En pratique, lorsque le contrôleur reçoit une requête de la part d'un utilisateur et qu'il déduit quel *handler* (c-à-d Action) bien précis doit être utilisé, il vérifie qu'il en existe déjà une instance. Si ce n'est pas le cas, il le crée. Ensuite, le contrôleur appelle la méthode de la classe Action :

- public ActionForward execute(ActionMapping mapping,
  ActionForm form,
  HttpServletRequest request,
  HttpServletResponse response) throws Exception

Dans la classe Action, cette méthode n'est pas abstraite et son implémentation renvoie la valeur *null*. Le travail de l'ingénieur logiciel est donc de créer une classe qui étend l'Action avec une méthode qui écrase celle de la classe supérieure. C'est cette méthode qui est réellement le pont entre le contrôleur et le modèle car elle contient les appels aux différents composants de cette deuxième couche.

Chaque instance du type de classe Action consulte les données contenues dans l'« ActionForm » qui lui est lié. Ces données sont celles que convoyait la requête et c'est à partir de celles-ci qu'il va pouvoir effectuer les appels aux composants de la logique d'affaires. En fonction de ces mêmes données et/ou des résultats des traitements de la logique d'affaires, la méthode execute() va renvoyer un objet de type « ActionForward » (org.apache.struts.action.ActionForward).

Détermination de la vue à renvoyer :

La classe ActionForward va permettre de déterminer quelle vue va être retournée à l'utilisateur. En effet, elle représente une destination (une page JSP de la vue) à laquelle le contrôleur doit passer la main lorsque l'exécution de la méthode execute() de l'Action s'est terminée. L'ActionForward ne fait pas directement référence à un chemin vers une page JSP mais à une balise « forward » qui est présente dans l'« action mapping » de cette Action dans le fichier de configuration de Struts (« struts-config.xml »).

Ce mécanisme est intéressant pour avertir l'utilisateur d'éventuelles erreurs. En effet, on peut s'arranger pour que les classes du modèle informent l'Action (qui les a appelées) si les traitements se sont correctement effectués.

Dans SMXpert, nous avons fixé pour convention que les méthodes classes « *Bean.java » du modèle renvoient toujours des objets de type String ou de type boolean. Ces chaînes de caractères fournissent des informations à l'Action qui en fonction de cela peut renvoyer des ActionForward différents.

> Par exemple, si nous sommes sur la page expert donnant une vue sur les concepts de maintenance, les mots-clés et les cas problèmes qui n'a plus été mise à jour depuis longtemps, on souhaite supprimer le mot-clé « toto ». Cependant, celui-ci a été

supprimé par un autre administrateur entre-temps et la page qui nous est présentée est donc erronée.

Nous envoyons la requête pour la suppression à la classe qui étend la classe Action appelée DeleteKeywordAction, celui-ci appelle la méthode deleteKeyword de la classe source.application.KBase.KeywordBean. Avant de le supprimer, la classe vérifie que le mot-clé existe. S'il n'existe pas, la méthode renvoie le objet de type String "This keyword does not exist".

L'Action en déduit que le traitement ne s'est pas correctement effectué et, donc, renvoie l'ActionForward« mapping.findForward("failure_deletekeyword_notexist") ». Dans l'extrait du fichier « struts-config.xml » ci-dessus (cf. ActionServlet), nous pouvons voir qu'à cet ActionForward correspond la page « error.jsp ». Cette dernière fait partie de la vue et sera renvoyée à l'utilisateur en réponse à sa requête.

En plus de renvoyer un ActionForward, l'Action peut ajouter une ou plusieurs entrée(s), en fonction du nombre d'erreurs survenues, à un objet de type ActionErrors. Cet objet est accessible dans l'ensemble du cadre de référence le temps de la requête. Nous évoquerons plus en détails son utilité dans *la vue de Struts*.

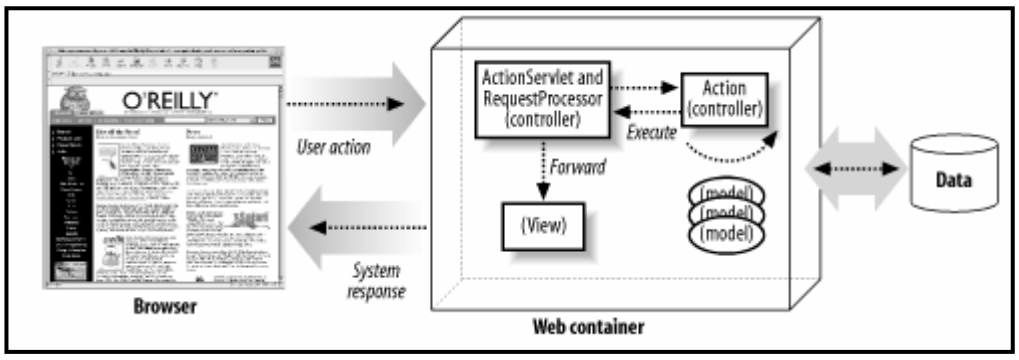


Figure 4.9 : La méthode execute ( ) est appelée par le contrôleur [52]

Pour avoir une vue plus globale de l'architecture physique de l'application Web, le schéma ci-dessous présente de manière simplifiée les liens et l'enchaînement des appels aux composants des différentes couches du *design pattern* :

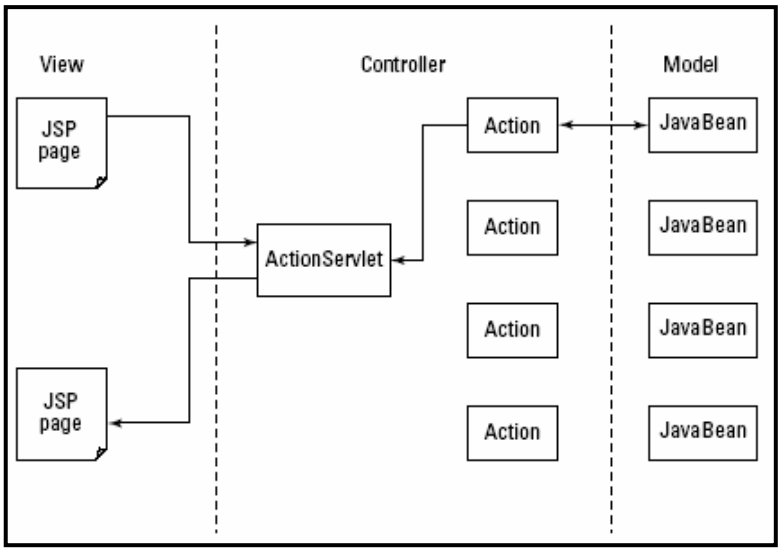


Figure 4.10 : Les liens et l'enchaînement des appels des composants des différentes couches du design pattern MVC [54]

Le schéma suivant présente la structure des relations entre les différentes classes de la couche contrôleur. On peut remarquer que les classes « ActionMapping » et « ActionForward » sont séparées car plutôt considérées comme des conteneurs. Ces derniers permettant de transférer un message comme des objets qui sont appelés explicitement par l'une des quatre autres classes.

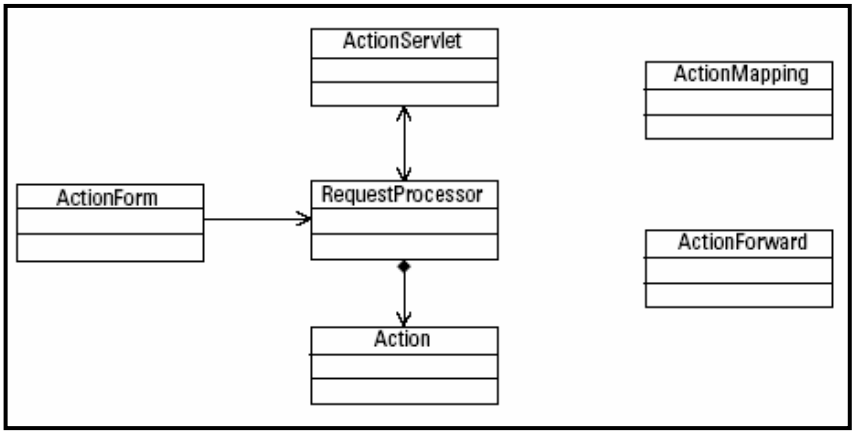


Figure 4.11 : Structure des relations entre les différentes classes de la couche contrôleur [54]

4.5.1.5 La vue de Struts :

La vue de Struts est constituée en grande partie de pages pouvant être présentées à l'utilisateur, ainsi que des classes de type ActionForm. Pour ce qui est des pages, elles ne dépendent pas d'un langage de balises particulier. En général, ces pages sont écrites en JSP.

Les classes de type ActionForm font partie de la vue, contrairement à ce que le nom du package dans lequel elles se trouvent le suggère (package Action). Elles constituent un moyen de faire parvenir les informations de l'utilisateur à la logique d'affaires et vice-versa.

ActionForm (org.apache.struts.action.ActionForm) : L'ActionForm est un Bean qui représente une ou plusieurs pages JSP, ayant trait, généralement, à une fonctionnalité spécifique de l'application Web. L'ensemble des attributs du Bean correspondent à ceux du/des Form(s) de la/les page(s) qui sont transmis au sein d'une requête en provenance de l'utilisateur. Il est à noter que ce Bean suit les règles imposées à la structure des JavaBeans, c'est-à-dire que l'on peut accéder aux précédents attributs par des méthodes de type set et get.

Lorsqu'une requête en rapport à un Action est envoyée, le cadre de référence va trouver dans le fichier de configuration de Struts (« struts-config.xml ») l'ActionForm qui correspond à cet Action. Cette information se trouve dans l'« action mapping » au niveau de l'attribut « name ». Par exemple, pour une requête demandant la suppression d'un mot-clé, le fichier demandé est « delkw.do ».

```
<action     path="/delkw"
            type="application.KBase.DeleteKeywordAction"
            name="delkwForm"
      scope="request"
            input="/pages/expert/delete.jsp"
            validate="true">
            <forward    name="failure_deletekeyword_notexist"
                        path="/pages/xpert/error.jsp"/>
            <forward    name="failure_deletekeyword_last"
                        path="/pages/xpert/error.jsp"/>
            <forward    name="failure_deletekeyword_nokws"
                        path="/pages/xpert/error.jsp"/>
      <forward    name="ok_del"
```

```
                              path="/pages/xpert/success.jsp"/>
    </action>
```

De l'« action mapping », le cadre de référence déduit que l'ActionForm qui entre en jeu dans ce cas-ci est « delkwForm ». Cependant, on s'attend à obtenir l'emplacement d'une classe, dans les packages de l'application, pour pouvoir l'instancier. Pour cela, il va falloir à nouveau consulter le fichier « struts-config.xml » dans lequel est, également, fourni un mapping entre l'identifiant obtenu précédemment (« delkwForm ») et la classe en question de type ActionForm. Ce système de mapping a l'avantage de permettre l'utilisation d'une telle classe sans devoir effectuer de changements dans tous les « action mappings » qui y sont liés. L'exemple ci-dessous provient du fichier « struts-config.xml » de SMXpert.

```
    <form-bean  name="delkwForm"
                    type="application.KBase.DeleteKeywordForm"/>
```

Lorsque le cadre de référence a identifié l'ActionForm dont il a besoin, il va vérifier si celui-ci a déjà été instancié ou pas (un seul ActionForm est instancié par Action qui lui correspond). S'il est déjà présent, alors il est réutilisé sinon le cadre de référence créée une nouvelle instance de celui-ci. Ensuite, la méthode reset() de l'ActionForm est appelée pour initialiser les champs du Bean avant de pouvoir le populer. Ceci étant fait, les attributs de l'objet vont pouvoir être remplis en fonction des données que transmet un Form au sein de la requête.

Gestion des erreurs (données introduites par l'utilisateur) :

Comme l'attribut « validate » de l'« action mapping » vaut *true*, la méthode validate() de l'ActionForm va finalement être appelée. Elle a pour but principal d'effectuer des vérifications sur les données provenant de la requête. Cela consiste généralement à vérifier que les données qui étaient obligatoires sont bien présentes. Nous présentons un exemple dans la Section 8.6 dans l'Annexe 8. La signature de la méthode est celle-ci :

```
    public ActionErrors validate( ActionMapping mapping,
                                  HttpServletRequest request)
```

Comme on peut le remarquer dans la signature de la méthode validate, celle-ci renvoie un objet de type ActionErrors. Lors des vérifications à chaque fois qu'une erreur est détectée,

une entrée dans l'instance de la classe ActionErrors va être créée. Cette entrée est constituée d'un identifiant de type String et d'un objet de type ActionError dans lequel se trouvent plus de détails sur l'erreur qui s'est produite. L'attribut « scope » de l'« action mapping » en rapport à l'ajout d'un fait étant initialisé à « request », les erreurs vont être retenues en mémoire le temps de la durée de vie de la requête. L'objet ActionErrors est visible dans l'ensemble du framerwork durant cette même période et ne sera supprimé qu'à la fin du traitement de la requête (qu'il se passe bien ou mal).

A la fin de l'exécution de la méthode validate de l'ActionForm, l'ensemble des erreurs qui se sont produites se retrouve dans l'ActionErrors. Cette méthode ayant été appelée par le contrôleur, c'est également lui qui va vérifier si l'objet renvoyé est vide ou rempli d'erreurs. Dans le premier cas, le contrôleur peut poursuivre et transmettre l'ActionForm à l'Action qui lui est lié. Dans le second cas, le contrôleur va renvoyer la page JSP qui est spécifiée dans l'« action mapping » comme page de retour, c'est-à-dire celle indiquée dans l'attribut « input ».

```
<action     path="/addfa"
            type="application.KBase.AddFactAction"
            name="addfaForm"
      scope="request"
            input="/pages/xpert/report.jsp"
            validate="true">
            <forward    name="failure_addfact_present"
                        path="/pages/xpert/success.jsp"/>
      <forward    name="ok_fa"
                        path="/pages/xpert/success.jsp"/>
</action>
```

Dans le cas de l'ajout d'un fait, on peut remarquer que le contrôleur va renvoyer vers la page « report.jsp ». Ci-dessous est illustré la partie de cette page relative à l'affichage des erreurs. Les deux dernières lignes vont permettre la consultation de l'objet ActionErrors, la récupération des erreurs à partir des identifiants de celles-ci et de l'affichage. Quant à la première ligne, elle correspond à l'affichage du titre du rapport d'erreur.

```
<h3><fmt:message key="common.reporting"/></h3>

    <p><html:errors property="missing"/></p>
    <p><html:errors property="wrong"/></p>
```

Comme nous l'avons vu dans la Section 4.5.1.4, l'ActionErrors peut également être utilisée dans l'Action. En effet, cela se prête particulièrement bien dans le cas de SMXpert car la logique d'affaires avertit de manière claire et précise des erreurs qu'elle a rencontrées. Avec les objets de type String en retour des méthodes appelées par l'Action, ce dernier peut ajouter des entrées dans l'ActionErrors juste avant de renvoyer son ActionForward. Généralement, si des objets de type ActionError ont été créés juste avant que la méthode execute de l'Action soit terminée, alors l'ActionForward va mener à une page d'erreur sur laquelle l'objet ActionErrors pourra être utilisé.

Au début, l'objet ActionForm a été présenté comme celui qui permettait de transmettre les données de la vue à la logique d'affaires. Cependant, il faut faire attention à ne pas passer l'objet ActionForm directement au modèle. En effet, cela résulterait en un couplage trop important des deux couches. A la place, il est conseillé d'utiliser un « Data Transfert Object » (DTO) qui est une structure de type Bean et représentant des entités logiques (par exemple : un concept, une question, un fait, …). Une autre façon de faire est d'appeler les méthodes de la logique d'affaires avec comme paramètres les données de l'ActionForm directement extraites dans l'Action. C'est une solution que nous utilisons aussi.

Officiellement, il est possible de court-circuiter le contrôleur. En effet, dans le *design pattern* MVC standard, il existe la description d'une interaction directe entre la vue et le modèle. Ce genre d'interactions est utile lorsque les données du modèle changent. En effet, ce dernier pourrait envoyer les changements directement à la vue pour que l'utilisateur puisse en être conscient. Cependant, dans le cas d'une application Web, il est plus difficile d'effectuer ce genre de mises à jour et donc, c'est à l'utilisateur de rafraîchir sa vue en effectuant de nouvelles requêtes.

4.5.1.6 Le modèle de Struts :

Le modèle de Struts ne doit pas répondre à des règles bien précises. Il doit comprendre un moyen d'accéder aux données et les traitements qui peuvent être effectués sur celles-ci. Il doit éviter autant que possible d'être lié à la vue. En effet, les interfaces ont tendance à évoluer esthétiquement au cours du temps alors que les fonctionnalités offertes restent les mêmes.

Dans ce type d'application et de *design pattern*, il est conseillé d'utiliser des objets qui encapsulent les données et les traitements que l'on peut faire dessus. Cependant, ces deux éléments peuvent se retrouver dans des classes séparées au sein du modèle. Il est à noter que l'ensemble des traitements représente la logique d'affaires de l'application.

En des termes plus pragmatiques que ce qui a été expliqué précédemment ci-dessus par rapport à la connexion entre le contrôleur et le modèle (cf. Action), le classe de type Action fait appel à l'instance de la classe « *Bean.java » qui correspond aux données sur lesquelles porte la requête et, éventuellement, à une classe du modèle qui contient les traitements adéquats (s'ils ne sont pas directement présents dans l'instance).

4.5.1.7 Scope des objets

Les requêtes des utilisateurs contiennent des informations qui vont mener à des traitements par l'application Web. Ces traitements peuvent engendrer des objets qui pourront être utilisés plus tard ou via le service d'une autre requête. De même, lors du lancement d'une application Web, des objets accessibles par tous les utilisateurs sont créés. Il existe donc une multitude d'objets qui peuvent être accédés par le même utilisateur et/ou différents utilisateurs (visibilité) et/ou à différents moments (durée de vie). La combinaison d'une certaine visibilité et d'une certaine durée de vie d'un objet est appelée « scope ». Il existe donc au sein de Struts différents scopes qui forment des zones partagées dans lesquelles sont stockées des objets. Ils sont au nombre de 4 : Request scope, Session scope, Application scope, Page scope (cf. Section 8.7 dans l'Annexe 8 pour plus de détails).

4.5.1.8 Configuration de Struts :

En plus de créer les classes du modèle, les pages de la vue et certaines classes du contrôleur, il faut définir les liens entre ces nouvelles classes (inconnues du cadre de référence pour le moment). Cette prise de conscience se fait par la rédaction du fichier de configuration de Struts (« struts-config.xml » dans le répertoire WEB-INF de l'application Web), cela permet principalement de définir les différentes connexions « vue-contrôleur ». Dans le cas de SMXpert, nous présentons le fichier dans l'Annexe 6.

Le fichier de configuration affecte la structuration des trois couches en permettant à l'ingénieur logiciel de préciser les interactions du contrôleur. Cela se fait par la définition des classes de type Action devant être utilisées dans certaines circonstances (provenance de la requête, …) et des classes de type ActionForm devant être liées aux précédentes. Il y a également la spécification de quelles vues vont être retournées à l'utilisateur suite au traitement de sa requête.

Le grand avantage de ce type de déploiement du cadre de référence est que l'on peut modifier les connexions entre les différentes parties de l'architecture MVC sans devoir effectuer des changements dans le code source des pages de la vue et des classes utilisées dans le modèle et/ou dans le contrôleur.

4.5.2 L'application Web : SMXpert

Dans cette partie, nous expliquons comment est structurée physiquement l'application.

Premièrement nous abordons les différentes classes qui ont été réalisées par les ingénieurs logiciels et qui reposent sur le cadre de référence Struts. Ce sont elles qui permettent à l'application d'offrir ses fonctionnalités.

Deuxièmement, nous présentons le mode opératoire des anciennes et des nouvelles fonctionnalités.

4.5.2.1 Les packages et les différents types de classes

Comme nous pouvons le voir ci-dessous, les classes sont identifiées par leur dénomination. Au sein de celle-ci se forme un ensemble de classes qui se ressemblent et qui offrent des services semblables dans leur façon de faire. Le symbole * correspond à la partie variable de la dénomination. Cela peut être un concept de la base de connaissances (Maintenance Concepts, Case problems, Facts, Index, Keywords, Questions, Recommandations, Users), une fonctionnalité, etc.

« *Form.java » : Ces classes étendent l'ActionForm de Struts. Ce type de classe a été décrit en détails ci-dessus. Il en existe une classe « *Form.java » pour chaque fonctionnalité du système et elle reprend toutes les informations qui ont rapport à la bonne exécution de celle-ci. Quelques exemples :

- Ajouter un mot-clé : « AddKeywordForm.java »
- Modifier un mot-clé : « ModifyKeywordForm.java »
- Supprimer un mot-clé : « DeleteKeywordForm.java »
- Ajouter un utilisateur : « AddUserForm.java »

« *Action.java » : Ces classes étendent l'Action de Struts. Ce type de classe a été décrit en détails ci-dessus. Comme dans le cas des classes précédentes, il existe une classe « *Action.java » pour chaque fonctionnalité du système. Dans un premier temps, elles récupèrent les informations contenues dans la classe « *Form.java » (qui correspond à la même fonctionnalité) et avec celles-ci elles peuvent appeler la méthode de la classe « *Bean.java » adéquate. Dans un deuxième temps, elles renvoient les erreurs ou le forward en fonction du résultat de la précédente méthode.

« *Bean.java » : Dans chaque composant logique de l'architecture, nous avons identifié des entités qui correspondaient à des regroupements de concepts ou, plus simplement, de données de la base de connaissances. Nous avons créer les classes « *Bean.java » pour représenter ces entités et les traitements (méthodes) qui pouvaient être appliquées sur elles. Nous présentons ci-dessous l'ensemble de ces classes :

- ListusersBean : reprend l'ensemble des utilisateurs et nous pouvons ajouter, effacer et vérifier si un utilisateur précis existe. Ce sont donc des opérations de gestion des utilisateurs.

- UserBean : reprend l'ensemble des utilisateurs et permet de faire des modifications sur un utilisateur en particulier. Ce sont donc des modifications de gestion d'un utilisateur.
- IndexBean : reprend l'ensemble des entrées de l'index et nous pouvons ajouter, modifier ou supprimer une de celles-ci.
- ConceptBean : reprend l'ensemble des concepts de maintenance de la base de connaissances et permet d'en ajouter de nouveaux.
- CpBean : reprend l'ensemble des cas problèmes de la base de connaissances et nous pouvons ajouter, modifier et supprimer un de ceux-ci.
- FactBean : reprend l'ensemble des faits d'une question de la base de connaissances et nous pouvons ajouter, modifier ou supprimer un de ceux-ci. Il est important de préciser qu'il n'est pas possible de modifier le nom et de supprimer les faits « Yes » et « No » car ils sont vitaux pour permettre au moteur d'inférence de se prononcer.
- KeywordBean : reprend l'ensemble des mots-clés de la base de connaissances et nous pouvons ajouter, modifier ou supprimer un de ceux-ci.
- QuestionBean : reprend l'ensemble des questions d'un concept de maintenance de la base de connaissances et nous pouvons ajouter, modifier ou supprimer une de celles-ci.
- RecommandationBean : reprend l'ensemble des recommandations d'un cas problème de la base de connaissances et nous pouvons ajouter, modifier ou supprimer une de celles-ci.
- Account : reprend l'ensemble des utilisateurs et permet d'effectuer les vérifications d'usage à la connexion d'un de ceux-ci. Il permet également la déconnexion d'un de ces utilisateurs. Ce sont donc des opérations de gestion des connexions des utilisateurs au système.

A chaque fois qu'une classe « *Bean.java » est instanciée, elle appelle la classe « *KB.java » adéquate qui lui permet de garnir sa variable globale privée qui représente la collection d'objets sur laquelle il va travailler.

« *DTO.java » : Ces classes sont utilisées comme structures de données. Elle possède des variables privées qui ne sont accessibles que via des méthodes publiques de type « get ( ) » et ne peuvent être fixées que via des méthodes publiques de type « set ( ) ».

« KB*.java » : Ces classes sont celles qui permettent de faire le lien entre la logique d'affaires et la couche de persistance, c'est-à-dire les fichiers « *.xml ». Il existe une classe de ce type qui traite les fichiers « *.xml » en rapport à chaque regroupement de concepts (Maintenance Concepts, Case problems, Facts, Index, Keywords, Questions, Recommandations, Users). Les classes qui y font appel sont les classes de type « *Bean.java ».

Pour chacune de ces classes, il existe :

- Une variable globale privée destinée à recueillir l'ensemble des instances d'un type de concept présent dans la base de connaissances. Détaillons la contenance de cette variable en fonction de chaque concept :
    - Concepts : l'ensemble des concepts.
    - Case problems : l'ensemble des cas problèmes.
    - Facts : les faits d'une question particulière.
    - Index : l'ensemble des entrées d'index.
    - Keywords : l'ensemble des mots-clés du système.
    - Questions : les questions d'un concept particulier.
    - Recommandations : les recommandations d'un cas problème particulier.
    - Users : l'ensemble des utilisateurs du système.
- Un constructeur.
- Une méthode « public boolean setKB* (…) » qui lit les fichiers « *.xml » adéquats, en fonction du concept, pour garnir la variable globale. Le contenu correspond à un vecteur d'instances de la classe « *DTO.java » qui sont du même type que le concept manipulé par cette classe « KB*.java ».
- Une méthode « public Vector getKB* ( ) » qui fournit le contenu de la variable globale.
- Une méthode « public void writeKB* (Vector list) » qui écrit les fichiers « *.xml » adéquats, en fonction du concept, à partir du vecteur passé en argument. Le contenu de ce vecteur correspond à des instances de la classe « *DTO.java » qui sont du même type que le concept manipulé par cette classe « KB*.java ». Ces instances ont été modifiées par les classes de type « *Bean.java » et ne sont donc plus en accord avec les instances du concepts des fichiers « *.xml » concernés, d'où la réécriture.

- Un ensemble de sous méthodes qui effectuent des traitements pour les trois méthodes précédentes.

Au sein de l'application Web SMXpert, les précédentes classes sont organisées en packages. La dénomination des packages correspond exactement à la décomposition en composants de l'architecture logique. En effet, chaque package porte le nom d'un composant. En ce qui concerne le composant logique « Aide », nous n'avons pas vu l'utilité de le matérialiser par un package.

source.application.administration : Ce package contient toutes les classes en rapport à la gestion des utilisateurs.

source.application.config : Ce package ne contient que deux fichiers. Ceux-ci correspondent à la fixation des valeurs du filtre. Ce sont une classe « SetFilterForm.java » et une autre « SetFilterAction.java ». Il n'y a pas de Bean qui interagit avec cette classe Action car les valeurs récupérées par l'application doivent être stockées dans la session pour qu'elles puissent être enregistrées avec les questions ajoutées ou modifiées.

source.application.index : Ce package contient toutes les classes en rapport à la gestion de l'index.

source.application.KBase : Ce package contient toutes les fonctionnalités ayant rapport aux traitements de la base de connaissances

source.application.security : Ce package contient les classes en rapport à la connexion, à la déconnexion des utilisateurs.

Les cinq packages précédents ne sont constitués que de classes de types « *Bean.java », « *Action.java » et « *Form.java ».

source.application.tools : Ce package, quant à lui, contient :

- Des classes héritées de COSMICXpert
- Toutes les classes « *DTO.java » et « KB*.java »
- Le moteur d'inférence « Engine.java »

- Une classe gérant le passage du langage de balises HTML au XML (et vice-versa) « TextHTMLConverter.java »

source.framework : Ce package ne contient que la classe « CustomRequestProcessor.java » qui étend la classe « RequestProcessor » officielle du cadre de référence Struts (cf. L'internationalisation ci-après).

Pour ce qui est de l'interface de l'application, elle est constituée de pages « *.jsp ». Comme le prévoit le *design pattern* MVC, une interaction entre la vue et la couche de persistance est possible. Ce sont les fichiers « *.xsl » qui permettent cela. Ils proviennent tous de COSMICXpert mais ont été adaptés. Les pages sont réparties en trois sous-dossiers correspondant à chaque partie de l'application : user, admin et xpert.

La base de connaissances quant à elle est structurée de la même façon que celle de COSMICXpert. Elle est également constituée de fichiers de type « *.xml ». Pour plus de détails, le lecteur pourra consulter le mémoire de F. Gruselin et J. Vilz [2], ainsi que celui de C. Duterme et N. Fabry [3].

Suite à la présentation des différents fichiers impliqués dans l'application, nous pouvons dessiner un schéma qui présente la structuration des fichiers abordés ci-dessus :

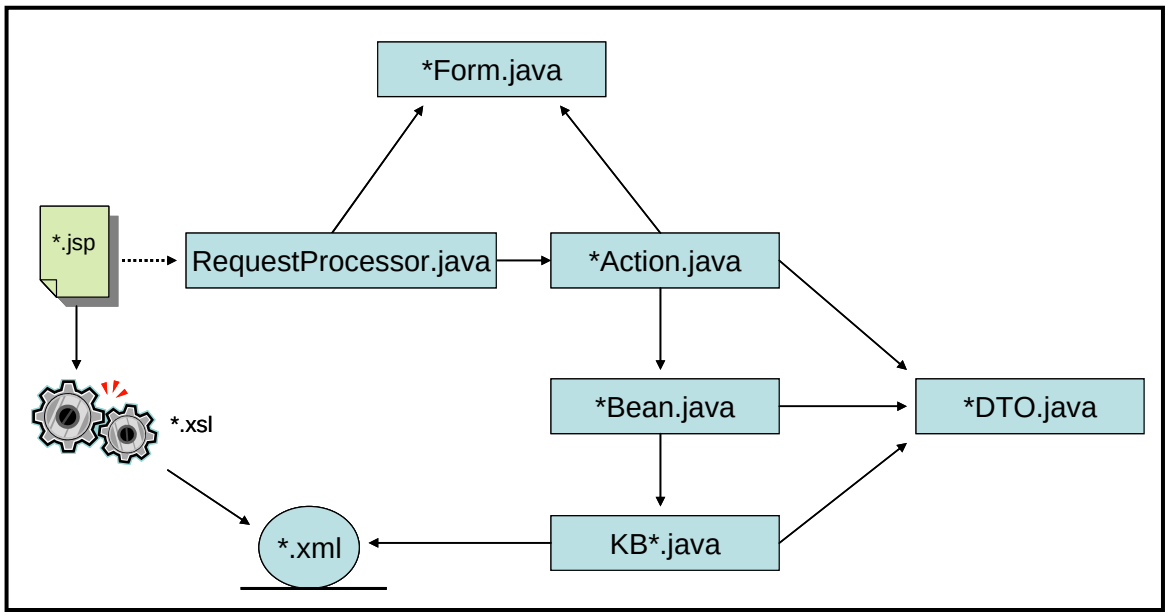


Figure 4.12 : Interdépendances des types de fichiers

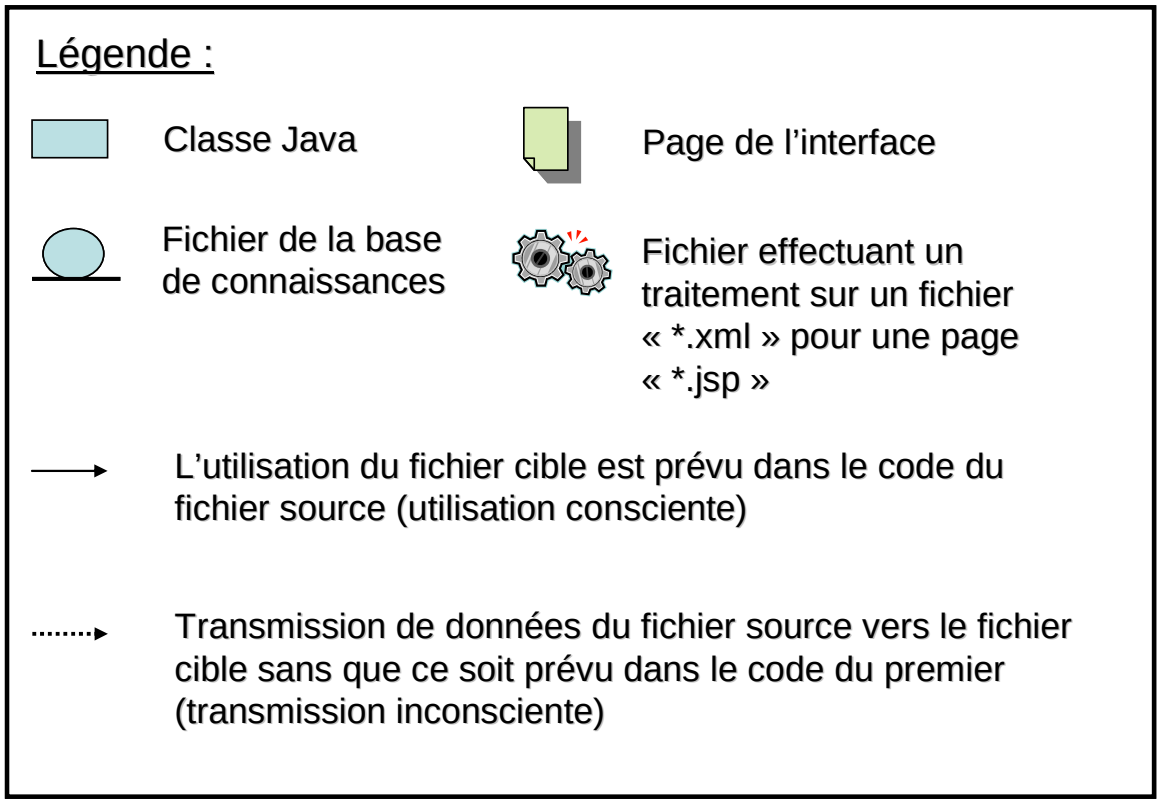


Figure 4.13 : Légende de la Figure précédente

4.5.2.2 L'internationalisation

L'utilisation du cadre de référence Struts apporte des facilités en ce qui concerne le support de différentes langues et régions pour l'application Web. Lorsque l'on parle d'internationalisation, on parle surtout de la possibilité pour l'ingénieur logiciel de remplacer facilement, et sans devoir modifier le code source des interfaces, l'ensemble des termes d'une langue utilisés par les traductions dans une autre langue. En effet, il est impensable d'imaginer une restructuration de l'application à chaque fois que l'on souhaite ajouter le support d'une langue.

On peut déduire que les règles de base pour l'internationalisation d'une application sont :

- Stocker les textes et les images apparaissant dans les interfaces hors du code source pour rendre l'ajout de nouvelles langues plus facile.
- Le format des dates, la représentation des nombres et de la monnaie doivent correspondre à ceux utilisé par l'utilisateur dans sa région de provenance.
- L'utilisateur peut facilement changer la langue et la région de l'application.

Le support de l'internationalisation par le cadre de référence Struts se base grandement sur l'utilisation des librairies Java ayant rapport au sujet. Deux classes sont importantes : celle qui permet de définir la langue et la région de l'utilisateur (java.util.Locale) et celle qui permet d'accéder aux fichiers regroupant les textes des différentes langues à afficher (org.apache.struts.util.MessageResources et son unique sous-classe concrète org.apache.struts.util.PropertyMessageResources).

java.util.Locale : Une instance de la classe Locale représente une langue et une région spécifique. Il n'est pas prévu d'appeler des méthodes de cette classe dans le but d'effectuer des traitements pour adapter son environnement. En effet, elle ne sert que de structure pour un stockage standardisé des informations précédentes. L'instance de la classe Locale est généralement passée en paramètre aux méthodes des autres classes qui, dès lors, s'adaptent en conséquence. Voici des exemples utilisant les constructeurs qui servent à la créer :

Locale beLocale = new Locale("fr", "BE") : Belge parlant français
Locale caLocale = new Locale("fr", "CA") : Canadien parlant français

En ce qui concerne Struts, le Web Container va créer une instance de la classe Locale sur base des requêtes envoyées par l'utilisateur. En effet, lors de chaque requête, le navigateur envoie sa configuration (langue et région) dans le header « Accept-Language » de celle-ci. Cela constituera la Locale par défaut. Le servlet récupère ces valeurs grâce à l'utilisation des méthodes suivantes de l'objet implémentant l'interface HttpServletRequest qui est un objet manipulable représentant la requête :

public java.util.Locale getLocale( ) : La Locale préférée
public java.util.Enumeration getLocales( ) : Les Locales préférées par ordre décroissant

Bien que toutes les requêtes contiennent les informations nécessaires pour constituer une instance de la classe Locale à chaque fois, il se peut que Struts soit configuré pour ne pas se préoccuper de l'internationalisation. Dans ce cas, il ne doit se référer qu'à une seule liste de termes apparaissant dans les interfaces (cf. Resource Bundle ci-après). Le fichier dans lequel on définit la non utilisation est le fichier de configuration de Struts « struts-config.xml » (exemple ci-dessous).

```
<controller
    debug="3"
    locale="true"
    nocache="true"
          processorClass=
                "source.application.framework.CustomRequestProcessor"/>
```

Dès que l'attribut « locale » équivaut à *true*, Struts ne créée d'instance de la classe Locale qu'une fois par session. Le cadre de référence la stocke ensuite dans l'objet Session, accessible par toutes les classes qui le constituent. Pour récupérer l'objet Locale, il suffit d'appeler la méthode getAttribute avec comme clé « Action.LOCALE_KEY ».

Cependant, si l'application offre à l'utilisateur la possibilité de modifier la langue de l'interface, il faut que le contrôleur vérifie à chaque requête si l'instance de la classe Locale stockée actuellement dans la Session correspond aux désirs de l'utilisateur. Le meilleur endroit pour effectuer cette vérification est le RequestProcessor car il est le point de passage obligé pour l'ensemble des requêtes. Cette classe faisant partie, dans son implémentation de base, du package « Action » de Struts, il n'est pas conseillé de le modifier directement. Il est préférable de créer une classe qui l'étend.

Dans le cas de SMXpert, nous avons créé une classe CustomRequestProcessor (source.framework.CustomRequestProcessor) qui étend la classe RequestProcessor. Notre classe écrase la méthode standard de Struts car il est obligatoire de la réécrire à partir du moment où l'utilisateur peut changer la configuration de sa Locale comme bon lui semble. Elle est présentée dans la Section 8.8 de l'Annexe 8.

Nous pouvons cependant préciser qu'elle va récupérer l'objet Locale (non nécessairement présent) qui est stocké dans la Session et celui contenu dans la requête de l'utilisateur. Enfin, l'objet Locale de la requête est enregistré dans la Session au cas où il ne serait pas déjà présent ou serait différent du premier.

org.apache.struts.util.MessageResources et org.apache.struts.util.PropertyMessageResources : La classe PropertyMessageResources est une implémentation concrète de la classe abstraite MessageResources. Elle permet de gérer un regroupement de ressources ayant trait à une Locale spécifique. Les ressources

correspondent principalement à du texte destiné à prendre place dans l'interface (boutons, titres de pages, messages d'erreur, …), mais peut aussi correspondre à des fichiers (images, …).

Chacun de ces regroupements est enregistré dans un fichier « *.properties » différent au sein du dossier « WEB-INF/classes/ » de l'application Web. Ce sont des fichiers textes qui doivent respecter des standards définis dans la classe java.util.Properties. Le plus important est de respecter un certain format de message. Ce format est le suivant :

clé=valeur

Un exemple provenant du fichier « ApplicationResources.properties » de SMXpert met bien en valeur cette structure :

```
loggedin.title=SMXpert Project
loggedin.msg=Welcome, {0}. You are now logged in.
```

En effet, chaque ressource peut être accédée via une clé que l'ingénieur logiciel défini. Ces clés seront les mêmes pour tous les fichiers « *.properties » de l'application Web, seules les ressources qui leurs sont associées vont être modifiées pour la langue du regroupement. La Figure 4.14 montre une capture d'écran de NetBeans représentant deux fichiers, l'un en anglais et l'autre en français.

Les fichiers « *.properties » sont définis dans le fichier de configuration de Struts « struts-config.xml » par des balises « message-resources » (exemple ci-dessous). Les regroupements étant chargés sous forme d'objets dans le ServletContext, l'attribut « key » permet de récupérer celui qui correspond à la Locale dont le cadre de référence a besoin. Dans le cas de SMXpert, nous disposons de deux fichiers « *.properties » qui correspondent à une version anglaise et à une version française de l'application :

```
<message-resources    key="MESSAGES_KEY"
                      null="false"
                            parameter="ApplicationResources"/>
    <message-resources      key="MESSAGES_FR_CA_KEY"
                      null="false"
```

```
parameter="ApplicationResources_fr_CA"/>
```

Il est à noter l'importance de fixer un nom de base commun à tous les fichiers « *.properties » et de respecter une certaine mise en forme. En effet, Struts va déduire, à partir des noms de fichiers, celui qu'il devra utiliser en fonction de la valeur de la Locale. Le nom final du fichier doit respecter la mise en forme : <nom de base>+ « _ »+ <abréviation ISO langue>+ « _ »+<abréviation ISO région>+ « .properties »

Le fichier par défaut ne contient pas d'information en rapport à une Locale et se contente du nom de base commun à tous. Struts le choisit lorsque aucun des noms de fichiers ne contient la même langue ou la même région que la Locale présente dans l'objet Session.

En conclusion, Struts permet une avancée considérable en ce qui concerne l'internationalisation. Précédemment, pour adapter une application conçue pour une seule langue, il aurait fallu modifier directement le code source des pages JSP en risquant d'introduire des erreurs et d'oublier de modifier certains endroits. Désormais, il suffit de modifier un fichier et de permettre à l'utilisateur de définir une variable. De plus, certains EDI comme NetBeans permettent une édition du fichier « application.ressources » qui facilite la modification de ce dernier. En effet, il est possible d'ajouter une langue en la sélectionnant parmi une liste et de compléter ensuite la nouvelle colonne, reprenant une case pour chaque propriété, par les termes de cette nouvelle langue (cf. représentation ci-dessous).

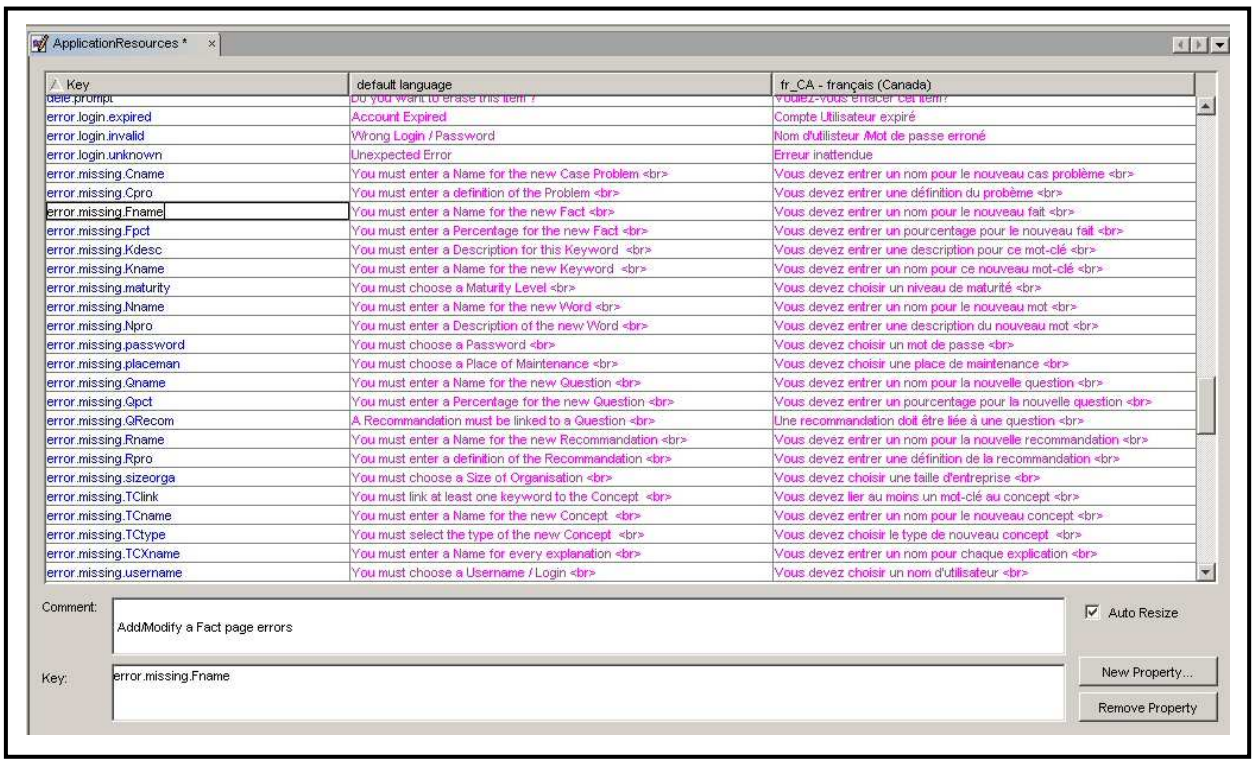


Figure 4.14 : Fichier ApplicationResources.properties édité avec l'IDE Eclipse d'IBM Corp.

4.5.2.3 Sécurité

Une démarche de sécurisation des plus simples est la configuration du Web Container pour qu'il interdise l'accès direct à certaines ressources directement. En effet, si rien n'est prévu, alors on peut imaginer que certains dossiers sensibles de l'application soient accessibles par n'importe quel utilisateur malveillant. Dans la plupart des cas, les utilisateurs n'ont pas idée de la structure des dossiers d'une application Web et de l'éventuelle possibilité d'y accéder directement. Cependant, il est de la responsabilité des ingénieurs logiciels de ne pas prendre le moindre risque.

Dans le cas de SMXpert, les fichiers « *.xml » de la base de connaissances sont particulièrement sensibles. Avant que le Web Container ne soit configuré, il était possible d'introduire l'URL correspondant à un dossier bien particulier et d'en visualiser le contenu (exemple : http:// 142.137.16.246/SMXpert/KB/).

Pour éviter ce type de déconvenue, il faut que le Web Container soit au courant de ce qu'il peut afficher et de ce pourquoi il doit demander un nom d'utilisateur et un mot de passe (propre au Web Container et non pas seulement à l'application Web !). Cette prise de conscience se fait au travers du fichier de configuration de base d'une application Web, c'est-à-dire « web.xml ». Comme on peut en voir un exemple ci-dessous, des balises « security-constraint » y ont été introduites.

```
<security-constraint>
    <web-resource-collection>
          <web-resource-name>Secured Ressource</web-resource-name>
          <url-pattern>/KB/</url-pattern>
          <http-method>GET</http-method>
          <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
          <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>
```

Dans cet exemple, nous pouvons remarquer que la balise « url-pattern » correspond au dossier de l'application Web qui va demander l'introduction d'un nom d'utilisateur et d'un mot de passe (dans ce cas-ci, c'est le dossier qui contient la base de connaissances de SMXpert) et ce pour toute requête http de type GET ou POST. Grâce à la balise « auth-constraint », on comprend que seuls les utilisateurs ayant le rôle « admin » auprès du Web Container Tomcat peuvent accéder aux ressources. Les noms d'utilisateur et les mots de passe qui correspondent au rôle « admin » sont ceux présents dans le fichier de configuration des utilisateurs de Tomcat « tomcat-users.xml » (localisé dans le répertoire Tomcat 5.0/conf/ et représenté ci-dessous). En voyant les comptes utilisateur ci-dessous, on comprend que les utilisateurs autorisés sont « root » et « admin ».

```
<tomcat-users>
      <role rolename="tomcat"/>
      <role rolename="role1"/>
      <role rolename="manager"/>
      <role rolename="admin"/>
      <user username="tomcat" password="tomcat" roles="tomcat"/>
```

```
        <user username="both" password="tomcat" roles="tomcat,role1"/>
        <user username="role1" password="tomcat" roles="role1"/>
        <user username="root" password="bubblesoft" roles="admin,manager"/>
        <user username="admin" password="fundp" roles="admin,manager"/>
</tomcat-users>
```

Une autre manière de sécuriser l'application Web est d'agir directement sur les requêtes et les contrôler une par une en fonction des ressources auxquelles elles veulent accéder. Cela permet de vérifier que les utilisateurs connectés à l'application accèdent bien au contenu qui leur est réservé. En effet, nous pourrions imaginer que l'existence de liens entre différentes parties de l'interface donne accès à des informations sensibles pour un utilisateur non accrédité (exemple: un lien de la partie utilisateur vers la partie expert). C'est pour cela qu'il est intéressant d'offrir un contrôle dynamique des requêtes grâce au contrôleur.

Précédemment, nous avons vu que le RequestProcessor est un point de passage pour toutes les requêtes. Cela en fait donc un acteur privilégié pour des traitements impliquant l'application dans toute sa globalité. La sécurité fait partie de ces traitements.

En conclusion, les deux approches sont complémentaires et il ne faut en négliger aucune. La première permet de protéger l'application des utilisateurs externes malveillants et la deuxième permet de vérifier les droits d'accès des utilisateurs réels aux différentes parties de l'application en fonction de leur statut.

4.6 Les problèmes rencontrés et les solutions apportées

4.6.1 Internet Explorer versus FireFox :

Suite à des tests avec des navigateurs différents (en l'occurrence Microsoft Internet Explorer et Mozilla FireFox), nous avons remarqué que SMXpert avait tendance à avoir des comportements différents. En effet, lors de tests avec Microsoft Internet Explorer, l'application Web subissait des crashs inexpliqués. Après avoir effectué les mêmes jeux de tests avec le navigateur Mozilla FireFox tout se passait à merveille.

Nous avons donc analysé les données qui transitaient effectivement entre le navigateur et le serveur. Le résultat est que l'entièreté de la requête ne pouvait pas passer avec Internet Explorer du fait de la limitation intentionnelle du cache par Microsoft. Par contre, Mozilla FireFox semble disposer d'une taille de cache plus importante qui permet une utilisation correcte de SMXpert.

4.6.2 Autorisation des cookies

Il est obligatoire d'autoriser les cookies de session car Tomcat y stocke des informations qui permettent de reconnaître un utilisateur lors de la session. En effet, lors de la première requête, le serveur fournit un cookie au navigateur de l'utilisateur. Ce dernier le renverra à chaque envoi d'une requête au serveur.

Si le cookie en provenance de Tomcat n'est pas accepté, alors l'application ne pourra pas s'exécuter et donnera l'impression d'avoir subi un crash.

Cette technique est pratique pour savoir à quel utilisateur appartient quel objet Session, surtout dans le cas où l'utilisateur changerait subitement d'adresse IP en pleine utilisation de SMXpert. En effet, l'utilisation de l'adresse IP est peu pratique dans ces cas et son stockage suppose des questions d'ordre déontologique au sujet de la vie privée.

4.6.3 Encodage des fichiers « *.xml »

Dans l'ensemble des fichiers « *.xml » se trouve une en-tête qui définit l'encodage qui est utilisé.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Le standard UTF-8 est particulièrement utilisé sur le continent américain. En effet, il convient à l'anglais et ne tient donc pas compte des lettres accentuées. Par contre, le standard ISO-8859-1 englobe plus de caractères, ce qui le rend plus pratique en vue d'une internationalisation de l'application.

Dans SMXpert, nous avons du transformer l’ensemble des en-têtes des fichiers « *.xml » pour qu’ils soient tous encodés en respectant la norme ISO-8859-1. L’application étant prévue pour supporter le français, ladite modification était une obligation.

Chapitre 5 : Démarche et outils pour l'analyse de la qualité de l'existant[9]

Nous allons aborder dans ce chapitre la question de la phase de tests lors du cycle de vie de développement d'un logiciel. En effet, cette phase est très importante car son objectif premier est de mettre en évidence les erreurs pour pouvoir les corriger et assurer, de ce fait, le respect des spécifications de départ (du cahier des charges).

L'IDEE- Standard Glossary of Software Engineering Terminology IEEE-STD729 (1983) propose une définition de ce qu'est un test:

> « Le test est l'exécution ou l'évaluation d'un système ou d'un composant par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus. »

De plus, il est important de préciser que corriger une erreur coûte entre 5 et 100 fois plus si le logiciel est déjà commercialisé et utilisé [48].

Le professeur J.-M. Desharnais présente dans le cours « Contrôle de la qualité et métriques » (LOG510) les conséquences des bogues comme nous pouvons le voir ci-dessus.

Cependant, les tests ne peuvent pas supprimer totalement la présence d'erreurs car tous les comportements possibles de l'application ne peuvent pas être balayés. Intuitivement, plus l'application prend de l'ampleur, plus la combinaison des entrées qu'elle accepte devient importante. Le temps alloué à la phase de tests étant limité, il faut privilégier les jeux de tests qui couvrent un maximum de comportements. Ces limitations sont généralement prises en compte par le cahier des charges.

---

[9] Les principales références pour ce chapitre sont les cours du Professeur N. Habra comme le « Laboratoire de Méthodologie de développement de logiciels » (2003-2004) ainsi que « Laboratoire de Méthodologie de développement de logiciels, matière approfondie » (2003-2004) d'où nous utilisons [67]. Les sites Internet contenus dans la section « Liens Chapitre 5 » des références ont également été d'une grande utilité.

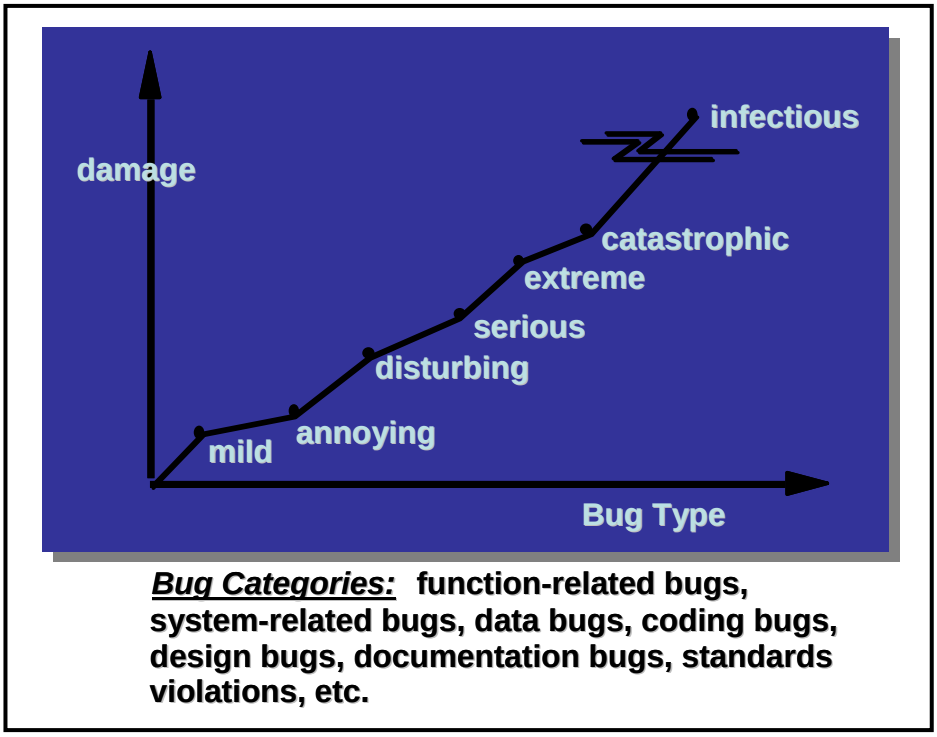


Figure 5.1 : Conséquences de bogues comme le montre cette figure provenant du cours LOG 510 [51]

Ce dernier prévoit une disponibilité de l'application pendant un certain pourcentage du temps total d'utilisation de celle-ci. Cette contrainte tolère, donc, que le système puisse être hors service pendant une certaine période. Implicitement, nous pouvons supposer que cette tolérance s'exprime vis-à-vis des erreurs non détectées lors de la phase de tests.

Dans ce chapitre, la première étape consistera à bien identifier tous les types de tests qui peuvent être déployés au cours de cette phase, de se rendre compte de leur importance sur la qualité d'une application et d'identifier la représentation qu'ils doivent adopter.

Ensuite, nous allons aborder deux manières de réaliser des tests : la manière traditionnelle consiste à réaliser les tests manuellement, tandis que la seconde consiste en leur réalisation automatique.

5.1 Différents types de tests

Il existe différents types de tests qui entrent en jeu à différents moments du cycle de vie du développement logiciel, c'est-à-dire à différents niveaux d'abstraction. Une caractéristique commune à tous ceux-ci est qu'ils doivent se baser sur des documents pour pouvoir être créés. Ce sont des documents qui sont produits au cours des différentes étapes du cycle de vie. Bien qu'il existe beaucoup de cycles de vie différents (le modèle en cascade, le modèle en spirale, le modèle par incréments, …), nous pouvons prendre l'exemple du cycle de vie en V ci-dessous qui est le plus connu et certainement le plus utilisé. Nous allons nous baser sur celui-ci pour expliquer les différents types de tests et les documents sur lesquels ils s'appuient (méthode MODAL).

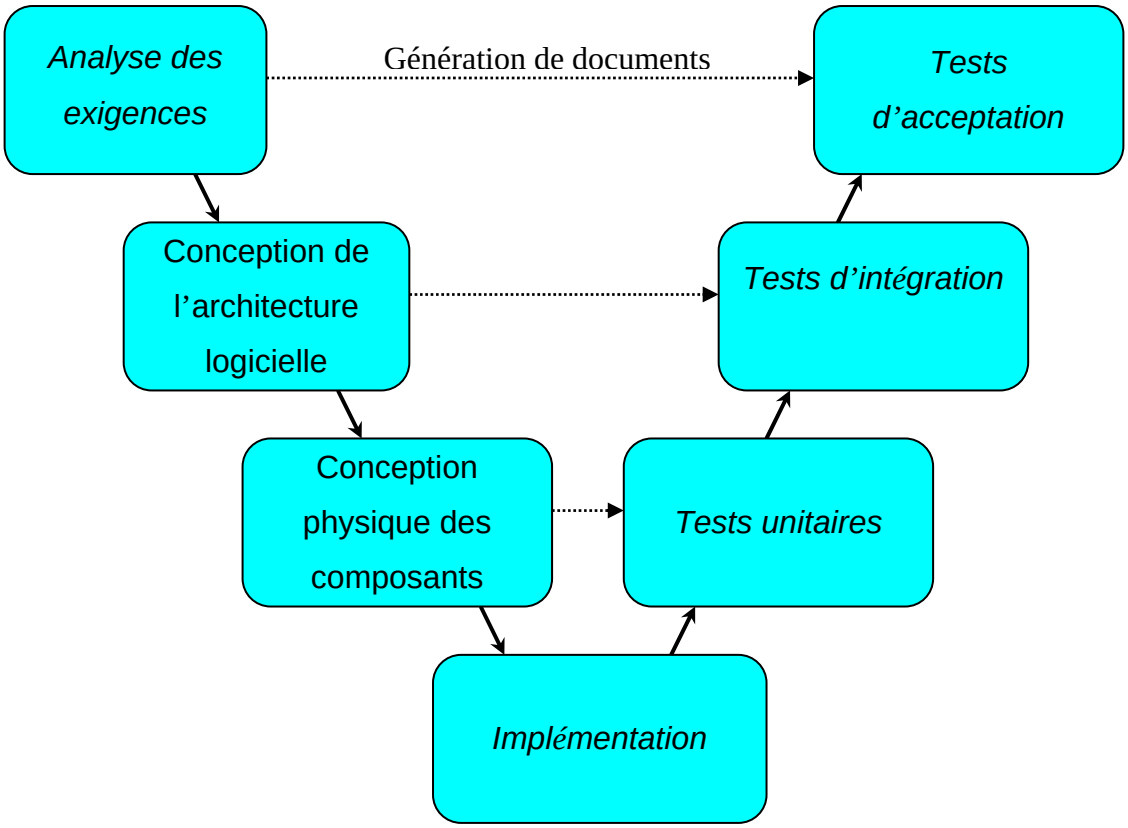


Figure 5. 2 : Cycle de vie en V du développement de logiciel

Les tests d'acceptation ont pour objectif de valider le logiciel dans sa globalité et à convaincre le client que l'application en question respecte bien le cahier des charges, avant la mise en œuvre opérationnelle de celui-ci. Pour cela, l'équipe de tests va vérifier chacune des

fonctionnalités sur base des jeux de tests définis lors de l'analyse des exigences. Cette analyse a produit des cas d'utilisation sur lesquels l'équipe de test peut se baser pour en déduire les jeux de tests à appliquer dans le futur.

La source des erreurs, identifiées lors de cette phase de tests, est liée aux définitions des besoins produites lors de l'analyse des exigences. Cela correspond généralement à des erreurs dans la définition ou des omissions de besoins. De ce fait, il est possible que l'équipe de développement se base sur un document qui ne reflète pas les vraies fonctionnalités requises par l'utilisateur. C'est le genre de situation que les tests d'acceptation mettent en évidence.

Pour pouvoir corriger les erreurs identifiées à ce niveau, le cycle de vie va devoir être recommencé du départ pour pouvoir modifier le besoin qui pose problème et la fonctionnalité qui lui est liée. En effet, l'équipe de développement s'étant basée sur des documents erronés, il faut recommencer toutes les étapes suivant l'analyse des exigences car des choix ont pu être posés en fonction de ce qui avait été fait auparavant. Cela oblige donc à revoir la spécification, la découpe en composants logiques et physiques, ainsi que l'implémentation sous-tendant les fonctionnalités. L'équipe de développement a un rôle important à jouer dans cette phase en cas de détection de problèmes.

Pour que l'on puisse mettre en œuvre les tests d'acceptation, il faut que les tests d'intégration soient terminés pour les composants logiques entrant en jeu dans les fonctionnalités à tester. Lorsque cette phase de tests se conclut, l'équipe aura l'accord ou le désaccord du client pour pouvoir déployer le logiciel.

Les avantages relatifs aux tests d'acceptation se trouvent principalement dans le fait que l'application est testée dans des conditions réelles, définies par le client (via les cas d'utilisation que ce dernier a validé lors de l'analyse des exigences) et non par l'équipe de développement. Les résultats des tests sont bien quantifiés ce qui permet de facilement juger si chaque test est réussi ou raté. Cela permet donc une bonne validation du produit et une assurance contre les erreurs avant le déploiement.

Les tests d'acceptation ne sont pas exempts de désavantages. Ils permettent effectivement de savoir qu'une fonctionnalité ne répond pas aux critères définis par le client, mais, à aucun moment, il n'est possible d'identifier les causes exactes de ce dysfonctionnement.

Les tests d’intégration ont pour objectif de valider chaque composant logique par rapport à ses spécifications, c’est-à-dire, d’une part, valider le fait que tous les composants physiques développés indépendamment fonctionnent bien ensemble, d’autre part, s’assurer que les interfaces des composants sont cohérentes entre elles et, finalement, que leur intégration permette de réaliser les fonctionnalités prévues. Ainsi, cette validation se passe en grande partie au niveau des interactions entre les composants qui sont définis lors de la conception de l’architecture logicielle. Les jeux de tests permettant de tester ces précédentes interactions regroupent tous les documents produits lors de la phase de conception de l’architecture logicielle comme les diagrammes de séquence.

Pour que les tests d’intégration puissent être effectués sur un composant logique, il faut que chacun des composants physiques (classes) le constituant ait réussi à passer les tests unitaires qui avaient été prévus pour lui.

Les tests unitaires ont pour objectif de valider chaque composant physique (classe) par rapport à ses spécifications. Cette validation a lieu au niveau du composant et non du produit complet. Pour cela, l’équipe de tests a besoin que l’implémentation des composant physiques soit terminée et que les spécifications des services offerts par ces composants aient été réalisées. Ces dernières sont rédigées lors de la phase de conception physique des composants du cycle de vie du développement logiciel. Lors de cette même phase et après que les composants physiques aient été clairement définis, les jeux de tests sont élaborés à partir des spécifications de chaque composant. En effet, celles-ci mettent bien en évidence tous les cas qui pourraient survenir, et plus particulièrement les cas limites.

Comme les autres types de tests, les tests unitaires font partie d’une démarche de validation de l’application et sont de type « black box ». Cette dernière expression signifie qu’ils se basent sur les spécifications, sans tenir compte du produit en lui-même (code source, …) (contrairement aux tests « white box » dans le cadre d’une démarche de vérification). Par exemple, pour les tests unitaires, l’implémentation des détails des composants n’est pas connue, voire ignorée volontairement, et seuls le comportement et le fonctionnement observables de l'extérieur sont testés. Il est important de préciser que les ingénieurs logiciels testent les composants un par un, comme si c’était des entités isolées et sorties du contexte des autres composants. Ils font cela de manière à s'assurer qu'ils fonctionnent correctement.

La source des erreurs identifiées lors de cette phase de tests, est liée à des erreurs dans le code source des services offerts par le composant physique. Pour pouvoir corriger les erreurs identifiées à ce niveau, seule la phase d'implémentation du cycle de vie devra être recommencée pour pouvoir effectuer les modifications adéquates. Ce travail sera à charge de l'équipe de développement ou de l'équipe de maintenance, si ces tests ont été effectués après le déploiement de l'application.

La mise en œuvre des jeux de tests et la phase d'implémentation sont fortement liées. En effet, lors de la conception des tests, l'équipe de développement et l'équipe de tests planifient une stratégie d'implémentation dans laquelle l'enchaînement des étapes d'implémentation et de tests unitaires est défini. Cela a pour but de planifier le meilleur séquencement de l'implémentation des classes afin de pouvoir les tester au mieux.

5.2 Importance de la phase de tests

Lors de chaque phase de conception du cycle de vie, la rédaction des jeux de tests pour les composants de cette phase peut mener à une réflexion sur ces derniers, et ce avant que les tests aient été effectués. En effet, s'ils sont écrits par l'équipe de développement, le fait de tenir en compte les cas limites lui permet de se remettre en question. Des doutes peuvent apparaître, remobiliser l'ensemble de l'équipe de tests et réengendrer une nouvelle itération au sein de cette phase. Cela consiste en une forme de vérification du travail qui a été accomplie au niveau de la conception.

Il va sans dire que la rédaction et l'application de tests permettent de détecter des erreurs et, donc, de s'assurer que le logiciel correspond bien au cahier des charges. En effet, l'équipe de développement et de tests permet de certifier que les fonctionnalités prévues s'exécutent sans problème et fournissent les résultats demandés. Cette pratique permet de minimiser la maintenance du produit car il est désormais accepté par le milieu informatique que l'effort de maintenance est plus grand que l'effort de correction du logiciel lors de son développement. La principale raison est que l'équipe de maintenance n'est pas nécessairement celle qui a développé l'application. La précédente va donc devoir consacrer un certain temps à la compréhension des formalismes de représentation de la conception de l'application à

différents niveaux d'abstraction, ainsi qu'à ce qui a été produit. En guise de référence, nous ne pouvons à nouveau que faire référence à Boehm B. et Basili V.R., comme dans le début du chapitre [48].

La planification des tests au sein de chaque phase de développement du cycle de vie permet de conserver la cohérence entre ce que l'on veut tester et ce que l'on est en train de concevoir. Les différents types de tests permettent donc de se limiter à un niveau d'abstraction différent et, donc, de détecter des erreurs spécifiques à celui-ci.

Les phases de tests forment les seules étapes qui permettent de valider l'application lors du cycle de vie de développement logiciel. Sans celles-ci, il se pourrait que des erreurs concernant d'importants composants passent entre les mailles du filet. L'équipe de maintenance serait alors chargée de corriger des éléments cruciaux de l'application bien trop tard.

Au vu des arguments précédents, nous avons pris conscience que les tests sont utiles. Cependant, le cycle de vie de développement de logiciel choisi peut jouer un grand rôle dans l'efficacité de ceux-ci. En effet, dans le cas du cycle en V, la préparation des tests peut être fastidieuse dans la mesure où il s'agit de prévoir bien à l'avance et de formaliser des tests qui ne seront pas passés avant bien longtemps. Cela implique un droit à l'erreur limité en terme d'oublis de cas à tester ou de définitions de jeux de tests erronées, voir inutiles. Il peut également en résulter une approche de tests moins formalisée et plus dynamique de la part de l'équipe de développement (remplaçant l'équipe de tests), surtout lors de l'implémentation. Chaque programmeur risque de tester seul son « morceau » de code source, sans avoir préalablement pensé aux différents cas. Bien que certaines personnes soient très douées pour ce genre de choses, elles peuvent perdre de vue les spécifications.

5.3 Importance de la phase de tests dans notre cas

Dans le cadre de la restructuration, la conception et l'exécution de tests sont des étapes très importantes car elles permettent de se familiariser avec l'ensemble de l'application. Ces deux étapes rassemblées forment ce que nous appelons la phase de tests, qui est un des premiers pas dans le processus de restructuration que nous avons initié.

La conception des tests nécessite d'effectuer des recherches au sein de la documentation du logiciel. Bien évidemment, cette documentation ne consiste pas en le seul « guide de l'utilisateur » mais en l'ensemble des documents générés au cours des différentes phases du cycle de vie de développement de logiciel. Ceux-ci sont très importants pour comprendre comment est architecturée l'application tant au niveau logique que physique et le cycle de vie de développement logiciel ayant été choisi par l'équipe de développement originelle. L'acquisition est primordiale pour l'équipe de restructuration.

En fonction du cycle de vie, des plans de tests peuvent être présents car ayant été générés au cours de chaque phase. Dans ce cas, il est légitime de se poser la question de l'utilité de concevoir de nouveaux plans de tests. La réponse est qu'il est utile d'effectuer à nouveau cette démarche car les plans de tests peuvent être erronés ou incomplets. En effet, si la restructuration d'une application est décidée, c'est qu'il existe un malaise vis-à-vis de celle-ci de la part des utilisateurs (et/ou des commanditaires). A moins que l'équipe de tests ait été peu scrupuleuse en passant sous silence les erreurs découvertes, il est plus que probable que les erreurs n'aient pas été détectées par les plans de tests créés au moment du développement de cette application.

En ce qui concerne les types de tests à rédiger, la conception de tests d'intégration et unitaires nécessitent, selon nous, beaucoup de ressources (temps, facteur humain, …). Cette affirmation est une réalité dans la mesure où nous souhaiterions tester tous les composants de l'application. Dans le cas d'une restructuration, nous conseillons d'effectuer ces tests seulement pour les composants liés aux fonctionnalités qui semblent contenir des erreurs (révélées par les tests d'acceptation).

En ce qui concerne **l'exécution des tests**, elle nous a permis de nous familiariser avec l'application en tant qu'utilisateur. En effet, les interfaces peuvent apporter beaucoup d'informations tant sur l'ontologie utilisée que sur l'enchaînement des fonctionnalités. Par exemple, dans le cas de COSMICXpert, il est plus agréable de se rendre compte de quels types de données sont composés les concepts de l'ontologie (concept topologique, cas problème, …) que d'aller directement consulter les fichiers de définition des structures dans la base de connaissances.

Lors de l'exécution des tests d'acceptation, nous sommes confrontés à l'interface graphique qui est le seul moyen d'interaction avec l'utilisateur. C'est pour cela que certaines équipes de développement utilisent celles-ci comme formalisme de spécification dans un but de communication avec le commanditaire. Nous comprenons, donc, que la manipulation des interfaces graphiques peut accélérer la compréhension des fonctionnalités.

Suite à l'exécution des jeux de tests du plan, nous avons pu identifier les fonctionnalités qui posaient problème. A partir de ce moment, nous allons pouvoir nous concentrer sur la correction de celles-ci. Dès que cette étape est franchie, nous disposons d'un logiciel théoriquement correct.

5.4 Le plan de tests

Lors de chaque phase de conception du cycle de vie, des documents de conception sont rédigés (analyse des exigences : use cases, conception de l'architecture logicielle par des diagrammes de séquencement et autres, …). A partir de ceux-ci et en fonction de la philosophie du cycle de vie, il arrive qu'un plan de tests soit conçu (cf. exemple du cycle de vie de développement logiciel en V ci-dessus).

Il regroupe l'ensemble des jeux de tests à effectuer sur les composants d'une phase de conception du cycle de vie de développement de logiciel. Ces jeux de tests sont représentés par des cas de tests. Ci-dessous, nous pouvons en voir un exemple provenant du plan regroupant les tests d'acceptation de COSMICXpert dans le cadre de la restructuration.

ID :	COSMICEXP-CT-2e
Description :	Ajouter un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert.
Entrée(s) :	L'utilisateur clique sur le bouton **Edit.** Ensuite sur **Add a keyword**. Il ne remplit aucun champ du formulaire.
Résultat(s) attendu(s) :	On attend du système qu'il refuse d'ajouter le mot-clef à la base de connaissance actuelle.

Figure 5.3 : Cas de test provenant du plan de tests de l'application COSMICXpert

Dans cet exemple de cas de test, nous pouvons identifier cinq parties bien distinctes :

- ID : représente l'identifiant du cas de test au sein du plan de tests.
- Description : décrit la fonctionnalité testée par ce cas de test.
- Pré-condition : décrit le contexte dans lequel l'expérimentateur et l'application doivent se trouver avant de pouvoir jouer le cas de test.
- Entrée(s) : décrit ce que l'expérimentateur doit faire dans le cadre de ce cas de test.
- Résultat(s) attendu(s) : décrit la réaction du système suite aux actions de l'expérimentateur.

En plus des cas de tests, le plan de tests peut contenir d'autres informations concernant le contexte dans lequel il va être exécuté. Par exemple, si l'équipe de tests n'est pas celle qui va les effectuer, alors il peut y avoir la procédure de tests à l'attention de l'expérimentateur ou les procédures d'exceptions lorsqu'il se trouve face à un problème inattendu. Pourraient également figurer des consignes à suivre (regroupées sous la forme de ce que l'on appelle un script d'orientation), une liste de tâches ou des feuilles d'observations pour noter ce qui s'est passé durant les tests.
Le plan de tests sert de guide pour l'équipe de tests lors de la validation du logiciel. Cette dernière trouve à l'intérieur de celui-ci toutes les informations utiles à l'exécution des tests qui ont été prévus au cours des différentes phases.

La plus grande utilité de ce document est que la période pendant laquelle une réflexion a lieu au sujet d'un certain type de tests diffère de la période à laquelle les jeux de tests seront exécutés. Il est donc vital d'en garder une trace pour éviter des oublis de cas limites. En effet, si aucun support ne regroupe les jeux de tests, alors l'équipe de tests risque de créer des tests à la volée, sans nécessairement disposer du recul nécessaire pour pouvoir couvrir tous les cas.

5.5 Réalisation manuelle des tests

5.5.1 Notre manière de procéder

Dans le cadre de la restructuration de COSMICXpert, nous avons commencé par **rédiger** un plan de tests après avoir consulté les cas d'utilisation créés par le professeur J.-M. Desharnais et présents dans sa thèse [15] (tests d'acceptation).

Nous avons commencé par utiliser les interfaces de COSMICXpert. Celles-ci faisant office de spécifications, elles nous a aussi permis de mieux comprendre le domaine étudié.

Après avoir mieux appréhendé le fonctionnement de COSMICXpert, nous sommes arrivé à la conclusion qu'il était possible de considérer le système dans son ensemble comme l'union de trois systèmes distincts, visant chacun un type d'utilisateur bien particulier. Par conséquent, l'idée de diviser le plan de test en trois parties indépendantes nous est naturellement venue à l'esprit.

Dans un premier temps, nous avons isolé les différentes catégories d'utilisateurs. Ces catégories sont :

- Administrateur
- Expert
- Mesureur

De cette distinction et de la structuration des cas d'utilisation, il nous a paru logique de répartir les cas de test en trois grandes catégories correspondant aux fonctionnalités spécifiques à chaque type d'utilisateur et en y ajoutant une quatrième reprenant les fonctionnalités communes à tous. Par « spécifique », nous entendons, par exemple, que même si un administrateur a la possibilité d'accéder à l'interface du mesureur et de l'expert, les cas de tests de la catégorie administrateur ne reprendront pas les tests sur ces deux dernières interfaces, mais seulement la partie propre à l'administrateur (la gestion des utilisateurs).

Dans le plan de tests, les cas de tests sont identifiés par un code dont le format générique est :

COSMICXXX-CT-Yz

Le XXX de cette représentation correspond à un code de trois lettres renseignant sur la catégorie à laquelle se rattache le cas de test visé. La sémantique de ce code est donnée par la table suivante :

Code	Catégorie
ADM	Fonctionnalité propre à l'administrateur

EXP Fonctionnalité propre à l'expert

MES Fonctionnalité propre au mesureur

COM Fonctionnalité commune à tous

Le Y de cette représentation correspond à un nombre identifiant le cas de test de manière unique. Pour sa part, le z fait référence à une lettre minuscule qui servira à identifier les éventuelles variantes d'un même test.

Maintenant que nous disposons d'un formalisme pour les identifiants des cas de tests, nous dressons la liste de ceux qui constituent les jalons de notre plan de tests. Le tableau ci-dessous ne liste que les cas de test et non chacune de leur variante.

ID	*Description*
COSMICADM-CT-01	Ajout d'un utilisateur à la liste des utilisateurs du système.
COSMICADM-CT-02	Suppression d'un utilisateur.
COSMICADM-CT-03	Changer le mot de passe d'un utilisateur.
COSMICADM-CT-04	Changer la date d'expiration d'un compte utilisateur.
COSMICADM-CT-05	Changer le statut d'un utilisateur.
COSMICADM-CT-06	Restaurer la base de connaissance.
COSMICADM-CT-07	Interaction avec l'interface Mesureur.
COSMICADM-CT-08	Interaction avec l'interface Expert.
COSMICEXP-CT-01	Consulter un arbre.
COSMICEXP-CT-02	Ajouter un mot-clef.
COSMICEXP-CT-03	Ajouter un concept topologique.
COSMICEXP-CT-04	Ajouter un cas problème.
COSMICEXP-CT-05	Ajouter un thème.
COSMICEXP-CT-06	Ajouter un fait.
COSMICEXP-CT-07	Ajouter une recommandation.
COSMICEXP-CT-08	Modifier un mot-clef.
COSMICEXP-CT-09	Modifier un concept topologique.
COSMICEXP-CT-10	Modifier un cas problème.
COSMICEXP-CT-11	Modifier un thème.
COSMICEXP-CT-12	Modifier un fait.
COSMICEXP-CT-13	Modifier une recommandation.
COSMICEXP-CT-14	Effacer un mot-clef.
COSMICEXP-CT-15	Modifier un concept topologique.
COSMICEXP-CT-16	Effacer un cas problème.
COSMICEXP-CT-17	Effacer un thème.
COSMICEXP-CT-18	Effacer un fait.
COSMICEXP-CT-19	Effacer une recommandation.
COSMICMES-CT-01	Consulter la définition d'un mot-clé.
COSMICMES-CT-02	Recherche des concepts topologiques en rapport à un mot-clef.
COSMICMES-CT-03	Recherche par index.
COSMICMES-CT-04	Consulter la définition des concepts topologiques/cas problèmes/thèmes/recommandations.
COSMICMES-CT-05	Obtenir une recommandation.
COSMICCOM-CT-1	Se connecter.
COSMICCOM-CT-2	Se déconnecter.

Figure 5.4 : Liste des cas de test du plan de tests de l'application COSMICXpert

La liste ci-dessus constitue l'ossature du plan de tests. Les détails des différents cas de test peuvent être consulté dans le document « Plan et rapport de tests – COSMICXpert » se trouvant en annexe (cf. Annexe 5).

Après la rédaction du plan de tests, nous abordons **l'exécution** de celui-ci. Nous suivons scrupuleusement les indications des cas de test en ce qui concerne les pré-conditions et les entrées. A chaque test, nous nous plaçons dans les conditions spécifiées et effectuons les manipulations prévues.

Les comportements et la présentation des données dans les interfaces sont comparés avec la spécification des résultats attendus du cas de test joué. Lorsque que l'expérimentateur arrive à la conclusion que ceux-ci présentent certaines anomalies (à partir des informations visuelles et autres), il note ses observations dans un rapport de test détaillé contenant une entrée pour chaque cas de test.

Une partie du rapport de test de COSMICXpert est illustrée ci-dessous (cf. Figure 5). Comme nous pouvons le voir, il reprend quelques cas de test de la partie « administrateur ». En plus des données de spécification d'un cas de test, chaque entrée dispose de trois parties supplémentaires :

- Résultats obtenus : les résultats attendus étant exprimés sous forme de points (conditions) à atteindre (à l'opposé d'une forme plus verbeuse dans le plan de tests), nous faisons de même pour les résultats obtenus. En fonction du comportement de l'application, nous réinscrivons ces précédents points dans une autre couleur (cf. Légende) exprimant le degré avec lequel le point a été atteint.
- Testé : elle prend la valeur « O » dans le cas où le test à bien été joué et « N » dans le cas contraire.
- Commentaires : dans le cas où le test n'a pas été joué ou que des points des résultats attendus n'ont pas été pleinement atteints, l'expérimentateur donne des explications sur les raisons le poussant à ne pas le jouer ou sur les problèmes rencontrés.

ID	Description	Entrées	Résultats attendus	Résultats obtenus	Testé	Commentaires
COSMICADM-CT-1a	Ajouter un utilisateur à la liste des utilisateurs du système.	Login : vincent Password : coucou Status :expert Date X : 2004-09-23	✓ utilisateur enregistré. ✓ accès au système. ✓ caractéristiques conformes : 1. date d'expiration. 2. type d'utilisateur.	✓ utilisateur enregistré. ✓ accès au système. ✓ caractéristiques conformes : 1. date d'expiration. 2. type d'utilisateur.	OUI	La fenêtre ne se rafraîchit pas après l'introduction du nouvel utilisateur. L'administrateur doit se reconnecter au système pour voir les changement.
COSMICADM-CT-1b	Ajouter un utilisateur à la liste des utilisateurs du système.	Login : idrissa Password : idrissa Status : measurer Date X : Ø	✓ refus de l'enregistrement.	✓ refus de l'enregistrement.	OUI	
COSMICADM-CT-1c	Ajouter un utilisateur à la liste des utilisateurs du système.	Login : idrissa Password : idrissa Status :Expert Date X : Ø	✓ refus de l'enregistrement.	✓ refus de l'enregistrement.	OUI	
COSMICADM-CT-1d	Ajouter un utilisateur à la liste des utilisateurs du système.	Login : jacques Password : coucou Status : admin Date X : 2004-08-12	✓ utilisateur enregistré. ✓ pas d'accès au système.	✓ utilisateur enregistré. ✓ pas d'accès au système.	OUI	
COSMICADM-CT-1e	Ajouter un utilisateur à la liste des utilisateurs du système.	Login : barth Password : coucou Status : expert Date X : 2004-09-22	✓ utilisateur enregistré. ✓ pas d'accès au système.	✓ utilisateur enregistré. ✓ pas d'accès au système.	OUI	
COSMICADM-CT-1f	Ajouter un utilisateur à la liste des utilisateurs du système.	Login : renson Password : coucou Status : measurer Date X : 2004_12_23	✓ refus de l'enregistrement.	✓ refus de l'enregistrement.	OUI	Le système accepte d'enregistrer un utilisateur dont la date d'expiration est syntaxiquement incorrecte

Légende

Fonctionnement normal — Fonctionnement OK mais anomalie mineure

Fonctionnement KO — Insuffisance de l'Interface Homme-Machine

Figure 5.5 : Une partie du rapport de tests détaillé de COSMICXpert

La version exhaustive du rapport de tests peut être consultée dans le document « Plan et rapport de tests – COSMICXpert » se trouvant en annexe (cf. Annexe 5).

Dans un but de lisibilité et d'évaluation du degré de fonctionnalité de l'application, nous avons créé une liste reprenant les entrées du rapport de tests dans une version simplifiée. En effet, ces dernières contiennent moins de détails et une nouvelle partie « Résultat » qui permet de fixer un ratio à un cas de test.

Le ratio de la partie « Résultat » correspond au nombre de conditions remplies (en fonction de la couleurs des points de la partie « résultats obtenus ») sur le nombre de conditions attendues (le nombre total de points de la partie « résultats attendus »). Nous attribuons une valeur « 1 » à un point qui a été atteint, une valeur « 0,5 » à un point n'ayant pas été atteint mais ne posant pas de problème pour la fonctionnalité (faute non-critique) et une valeur « 0 » à l'ensemble du cas à partir du moment où nous n'avons pas réussi à le terminer. Cette dernière valeur met en évidence le fait qu'une faute critique est survenue. Nous entendons par ceci des fautes qui empêchent le système de donner un résultat correct tandis que par faute non-critique, nous signifions des fautes qui mènent à un fonctionnement correct mais accompagné d'un effet non prévu.

Il est important d'attirer l'attention sur le fait que ces valeurs et les points à atteindre sont arbitraires. Ce système d'évaluation nous permet de nous rendre compte des fonctionnalités qui posent problème et ne délivrent pas exactement le service attendu.

Comme nous l'avons vu plus haut (cf. version détaillée du rapport de tests), certains cas de test n'ont pas été effectués car les fonctionnalités se ressemblaient fortement. Le test aurait alors introduit une redondance faussant les résultats. Par exemple, « Ajouter un thème » et « Modifier un thème » sont fort semblables. Après avoir testé l'ensemble des cas liés à l'ajout ainsi que le cas de base de la modification, il n'est plus utile de jouer les cas de test alternatifs de cette dernière fonctionnalité. En effet, si une erreur est présente lors de l'ajout et que celle-ci apparaît dans la modification, alors le fait de jouer ces cas de test générerait un double comptage d'une seule et même erreur à corriger. Dans de tels cas, l'expérimentateur aura rempli la partie « Commentaires » du cas de test concerné.

ID	*Description*	*Testé*	*Résultat*
COSMICADM-CT-1a	**Ajout d'un utilisateur à la liste des utilisateurs du système.**	**O**	**2.5 / 3**
COSMICADM-CT-1b	Ajout d'un utilisateur à la liste des utilisateurs du système.	**O**	1 / 1
COSMICADM-CT-1c	Ajout d'un utilisateur à la liste des utilisateurs du système.	**O**	1 / 1
COSMICADM-CT-1d	Ajout d'un utilisateur à la liste des utilisateurs du système.	**O**	2 / 2
COSMICADM-CT-1e	Ajout d'un utilisateur à la liste des utilisateurs du système.	**O**	2 / 2
COSMICADM-CT-1f	Ajout d'un utilisateur à la liste des utilisateurs du système.	**O**	0 / 1
COSMICADM-CT-2	**Suppression d'un utilisateur.**	**O**	**2 / 2**
COSMICADM-CT-3a	**Changer le mot de passe d'un utilisateur.**	**O**	**1 / 1**
COSMICADM-CT-3b	Changer le mot de passe d'un utilisateur.	**O**	1 / 1
COSMICADM-CT-3c	Changer le mot de passe d'un utilisateur.	**O**	1 / 1
COSMICADM-CT-4a	Changer la date d'expiration d'un compte utilisateur.	**O**	1 / 1
COSMICADM-CT-4b	Changer la date d'expiration d'un compte utilisateur.	**O**	0.5 / 1
COSMICADM-CT-4c	Changer la date d'expiration d'un compte utilisateur.	**O**	0.5 / 1
COSMICADM-CT-4d	Changer la date d'expiration d'un compte utilisateur.	**O**	0 / 1
COSMICADM-CT-4e	Changer la date d'expiration d'un compte utilisateur.	**O**	0 / 1
COSMICADM-CT-05	Changer le statut d'un utilisateur.	**O**	2 / 2
COSMICADM-CT-06	Restaurer la base de connaissance.	**O**	0 / 1
COSMICADM-CT-7a	Interaction avec l'interface Mesureur.	**O**	1 / 1
COSMICADM-CT-7b	Interaction avec l'interface Mesureur.	**O**	0 / 1
COSMICADM-CT-7c	Interaction avec l'interface Mesureur.	**O**	0 / 1
COSMICADM-CT-08	Interaction avec l'interface Expert.	**O**	1 / 1

Figure 5.6 : Une partie du rapport de tests non-détaillé de COSMICXpert (cas de test partie administrateur)

5.5.2 Résultats avant la correction de COSMICXpert

Nous allons présenter les résultats des cas de test sous forme d'un tableau récapitulatif. Les colonnes permettent de visualiser les résultats pour les différentes parties (administrateur, expert, mesureur, commune).

Au sein de ce tableau, nous indiquons, entre autres, la quantité de fautes critiques ou non-critiques présentes. Résumant les résultats des cas de test, la dernière ligne du tableau contient un calcul du pourcentage de la complétude de chaque partie. Celui-ci correspond à une fraction :

Total des valeurs attribuées par l'expérimentateur aux points (conditions) des cas de test de cette partie

---

Nombre total de points (conditions) à atteindre

Figure 5.7 : Méthode de calcul du pourcentage de la complétude de chaque partie

Comme nous basons nos calculs sur des points à atteindre, nous ne pouvons tirer des informations que sur ces points. Ces informations sont du type « 70% des points sont atteints et 30% sont toujours à atteindre pour que le système puisse offrir toutes les fonctionnalités fournissant des services corrects en accord avec le cahier des charges ».

Bien que nous ayons tenté de ne pas jouer les cas de test qui entraîneraient une certaine redondance dans les résultats, la façon dont s'effectue le comptage permet d'effectuer une correction même si nous les exécutions. En effet, si un point à atteindre se retrouvait dans plusieurs « résultats attendus » de différents cas de test, alors il serait compté deux fois dans le dénominateur (comme s'il était distinct). Bien que la valeur du point à atteindre puisse être différente selon le cas de test, elle est également comptée deux fois dans le numérateur. Nous supposons qu'il y a de grandes chances qu'un même point possède la même valeur dans les différentes fonctionnalités dans lesquelles il apparaît.

Le pourcentage final ne serait biaisé que si la valeur d'un même point à atteindre était comptée deux fois sans qu'il le soit dans le nombre total de points.

	ADM	***EXP***	***MES***	***COM***
Nombre total de cas de test	**21**	**94**	**9**	**4**
Nombre total de cas de test utilisés	**21**	**62**	**9**	**4**
Nombre total de				

cas de test non utilisés	**0**	**32**	**0**	**0**
Nombre total de cas de test d'un poids « 1 »	**16**	**61**	**9**	**4**
Nombre total de cas de test d'un poids « 2 »	**4**	**1**	**0**	**0**
Nombre total de cas de test d'un poids « 3 »	**1**	**0**	**0**	**0**
Poids total	**16*1 + 4*2 + 1*3 = 27**	**61*1 + 1*2 = 63**	**9*1 = 9**	**4*1 = 4**
Nombre d'erreurs non critiques	**3**	**13**	**0**	**1**
Nombre d'erreurs critiques	**6**	**11**	**1**	**0**
Poids des cas de test réussis	**19,5**	**44,5**	**8**	**3,5**
Nous évaluons cette partie du système correct à	**19,5 / 27 = 72 %**	**44,5 / 63 = 70 %**	**8 / 9 = 88 %**	**3,5 / 4 = 87 %**

Figure 5.8 : Résultats de l'application COSMICXpert au plan de tests avant correction

5.5.3 Résultats après la correction de COSMICXpert

Après avoir identifié les fonctionnalités posant problème, nous avons apporté des modifications à l'application. Nous avons ensuite joué à nouveau les cas de test pour pouvoir comparer l'évolution de la qualité des services de l'application. Il est à noter que nous n'avons pas reproduit les six premières lignes du tableau précédent vu qu'aucun cas de test n'a été ajouté ou supprimé au nombre total.

	ADM	***EXP***	***MES***	***COM***
Poids total	**16*1 + 4*2 + 1*3 = 27**	**61*1 + 1*2 = 63**	**9*1 = 9**	**4*1 = 4**
Nombre d'erreurs non critiques	**0**	**1**	**0**	**0**
Nombre d'erreurs critiques	**1**	**2**	**0**	**0**
Poids des cas de test réussis	**26**	**60,5**	**9**	**4**
Nous évaluons cette partie du système correct à	**26 / 27 = 96 %**	**60,5 / 63 = 96 %**	**9 / 9 = 100 %**	**4 / 4 = 100 %**

Figure 5.9 : Résultats de l'application COSMICXpert au plan de tests après correction

5.5.4 Evaluation de la qualité de COSMICXpert

Suite à la première phase de tests, nous avons pu remarquer que les erreurs, qui se répétaient, étaient d'un même type. Nous listons ci-dessous les grands types d'erreurs rencontrés durant cette précédente phase :

- Des erreurs de design liées à l'interface homme-machine.
- Des erreurs liées à des accès à des fichiers qui n'existent pas.
- Des erreurs liées à l'utilisation des verrous.
- Des erreurs liées à l'utilisation des pop-up menus, bloqués par certains systèmes d'exploitation et/ou pares-feu.
- Un problème d'affichage de données pour certaines images.

Lorsque nous comparons les résultats avant et après la correction de COSMICXpert, nous pouvons conclure que nous avons expérimenté une augmentation qualité.

Bien qu'il y ait eu une amélioration au niveau des fonctionnalités de COSMICXpert du fait d'un bon balayage des exigences fonctionnelles du logiciel par les cas de test. Il peut toujours rester des problèmes au niveau non-fonctionnel (lenteur, facilité à maintenir, …). Cette constatation est la principale faiblesse de cette phase de tests. En effet, nous n'avons fait que corriger les erreurs de programmation. Les habitudes de programmation des différentes équipes se faisant déjà sentir, nous n'avons, par exemple, pas amélioré la facilité à maintenir l'application.

En conclusion, ces phases de tests et cette phase de corrections nous ont permis de nous rendre compte à quel point l'architecture de COSMICXpert nécessitait d'être remodelée. De plus, l'opportunité de développer un système à base de connaissances pour le domaine des processus de maintenance nous a incité à le restructurer.

5.6 Automatisation des tests

Lors de notre stage à l'Ecole de Technologie Supérieure de Montréal (ETS), nous avons eu l'occasion de présenter la partie « Les tests avec un outil intégré (Rational) » du cours « Contrôle de la qualité et métriques » (LOG510) enseigné par le professeur J.-M. Desharnais, notre maître de stage.

Au cours de cet exposé, nous avons présenté, de manière critique, l'utilité que procurait l'automatisation des jeux de tests, ainsi que la manière d'utiliser certains outils de la suite Rational d'IBM comme Robot, Test Manager et SiteCheck. Cette tendance tendant s'accentuant, nous allons donner de plus amples explications sur celle-ci.

5.6.1 Quels outils pour quels types de tests ?

Il existe des programmes permettant d'automatiser la plupart des types de tests. L'ouvrage, portant le titre « Software testing tools », de Pentti Pohjolainen [65] prend en compte sept types différents et en donne une explication, comme nous pouvons le voir ci-dessous :

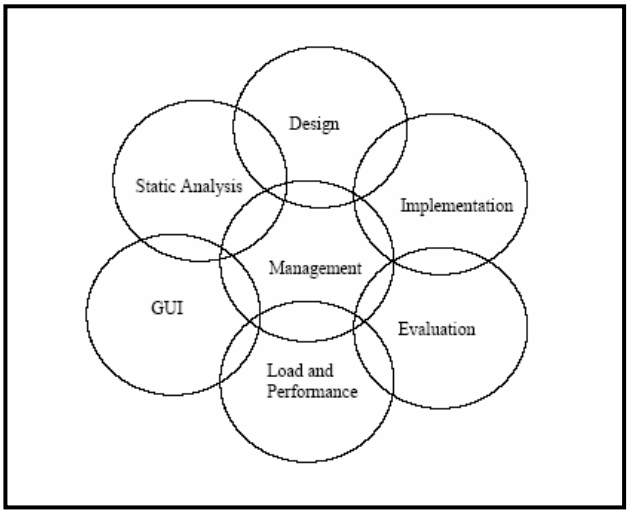


Figure 5.10 : Classification des outils de tests automatisés [65]

Test Design Tools

Tools that help you decide what tests need to be executed. Test data and test case generators.

GUI Test Drivers

Tools that automate execution of tests for products with graphical user interfaces. Client/server test automation tools, including load testers, also go here.

Load and Performance Tools

Tools that specialize in putting a heavy load on systems (especially client-server systems). These tools are often also GUI test drivers.

Test Management Tools

Tools that automate execution of tests for products without graphical user interfaces.

Also tools that help you work with large test suites.

Test Implementation Tools

Miscellaneous tools that help you implement tests. For example, tools that automatically generate stub routines go here, as do tools that attempt to make failures more obvious (assertion generators, etc.)

Test Evaluation Tools

Tools that help you evaluate the quality of your tests. Code coverage tools go here.

Static Analysis Tools

Tools that analyse programs without running them. Metrics tools fall in this category.

Dans un souci de documentation, nous laissons le soin au lecteur de consulter le document de Pentti Pohjolainen [65] qui recense pas moins de 198 logiciels de tests automatiques, toutes catégories confondues.

5.6.2 Tests automatiques : avantages et inconvénients

La phase de tests est une opération longue et coûteuse en ressources (temps, facteur humain, …). La rédaction des cas de test peut prendre beaucoup de temps pour chaque phase de tests planifiée et s'avérer, dans certains cas, répétitive. De la recherche de réduction des coûts est née l'idée de l'automatisation de cette phase. En effet, les tests automatiques, une fois créés, sont répétables à volonté. C'est un grand avantage dans la mesure où l'on veut s'assurer qu'une application est testée dans les mêmes conditions quelle que soit sa version ou la plateforme. De plus, cela facilitera la comparaison des résultats des tests aux différents moments de la vie du logiciel. En effet, les cas de test sont joués exactement de la même manière et les erreurs humaines au cours de ceux-ci sont supprimées.

Cette répétitivité est réellement la pierre angulaire des tests automatiques. Elle permet de tester plus et plus souvent, surtout dans les cas où le plan de tests est tellement imposant qu'il en devient impossible de les rejouer manuellement. Dans la mesure où la phase de tests est effectuée rapidement, les erreurs peuvent être détectées plus tôt que lors de tests manuels.

A partir du moment où les erreurs sont détectées plus tôt, l'application peut être livrée rapidement ce qui assure une plus grande confiance de la part du commanditaire dans l'équipe

de développement. En effet, il peut s'imaginer que si le logiciel lui est livré si rapidement, c'est que l'équipe a jugé qu'il était fonctionnel.

Les tests automatisés forment également un moyen de formaliser les cas de test. Il arrive que des plans de tests soient très mal documentés, tandis que le passage à l'automatique engendre le respect de certaines contraintes (découpe des informations d'un cas de test, cf. Test Manager). Dans le cas de Rational ROBOT, les scripts qu'il utilise pour jouer ces cas de tests apportent la rigueur car nous pouvons savoir exactement ce qui va se passer lors du test.

Les tests automatisés ouvrent de nouveaux horizons aux phases de tests. En effet, il est maintenant possible de réaliser des tests qui étaient jusque là impossibles à réaliser manuellement. Par exemple, nous pourrions simuler la présence de 2.000 utilisateurs.

Sous certaines conditions, les tests automatisés peuvent apporter des avantages dans le cadre d'une restructuration. En effet, les cas de test rédigés, pour l'application de départ, peuvent être réutilisés à des fins de tests de la nouvelle application. Il n'est pas exclu que ces derniers doivent être modifiés pour qu'ils puissent effectivement être correctement utilisés.

Cependant les tests automatisés ne se sont pas imposés comme la solution miracle qui permettrait d'éviter toutes les erreurs. Cela est principalement du au fait que dans la plupart des outils l'humain est toujours à la base de la création des tests. De plus, lors de l'exécution manuelle des tests, la faculté d'adaptation de l'humain peut lui permettre de faire face à un comportement non prévu de l'application et donc de pouvoir continuer le cas de test.

Dans le cas où les tests automatisés prennent la forme de scripts exprimés dans un certain langage, l'ingénieur logiciel se doit d'apprendre celui-ci. Cela rend ce type de tests difficile à créer s'il n'y a pas d'assistance et encore plus dans le cas de la maintenance de cas de test automatisés. S'il existe un outil pour assister la création des cas de test, il va falloir apprendre. Lors de la première utilisation, cette démarche fait augmenter la charge de travail, mais la récente maîtrise de l'outil peut être rentabilisée lors des projets suivants.

Un problème assez handicapant concerne les difficultés d'interopérabilité du logiciel de test avec le logiciel à tester. Les outils se basant sur les interfaces peuvent être fortement liés à une plateforme. Par exemple, Rational ROBOT a été optimisé pour s'exécuter sous Microsoft

Windows et plus particulièrement avec les sites web via Microsoft Internet Explorer. Les applications n'ayant pas une interface web compatible avec le précédent navigateur ne sont pas impossible à tester mais demandent beaucoup plus de travail. Dans certains cas, cela peut rendre le test manuel plus économique que l'automatique.

Une tendance qui est assez marquée est que les tests effectués manuellement permettent de trouver plus d'erreurs lors de la première exécution par rapport aux tests automatisés. Il ne faut pas non plus croire que le fait d'effectuer une phase de tests permet de détecter l'ensemble des problèmes d'une application. En effet, ce n'est pas parce qu'aucun problème n'a été trouvé qu'il n'y en a pas.

Les logiciels offrant de l'assistance pour créer les tests automatisés et/ou permettant de tester d'autres logiciels directement, peuvent également contenir des erreurs. Dans le cas d'assistance lors de la création de tests automatisés, ils pourraient induire les utilisateurs en erreur (écriture de scripts,…). Ironiquement, il est certain qu'ils ont eux-mêmes du être testés, alors que nous sommes certains que les tests de type « black box » sont rarement capables de découvrir toutes les erreurs d'une application.

Ci-dessous, nous pouvons observer le diagramme de Kiviat réalisé par Patrice Bellot [66] pour comparer les tests manuels aux tests automatiques sur base de quatre critères que sont l'efficacité de détection, l'évolutivité, la couverture et les coûts. En reliant les points sur les axes représentant la quantification de ces derniers, nous pouvons obtenir une surface.

La surface de test manuel et une de celles des tests automatisés peuvent être comparées. Les cas de test étant créés par des humains, il est normal que les caractéristiques de couverture et d'efficacité de détection soient équivalentes dans les deux types de tests. Par contre, les tests automatisés sont moins évolutifs que les tests manuels. En effet, une fois créés, ils sont difficiles à modifier (cf. les scripts de Rational ROBOT plus loin), contrairement à l'intelligence humaine qui peut s'adapter à la situation.

Grâce au diagramme, nous pouvons également étudier l'évolution des surfaces des tests automatisés au cours du temps. Vu leur répétitivité, l'équipe de tests à tendance à effectuer plus de phases de tests qui coûtent, malgré tout, en temps et en facteur humain (de manière beaucoup moins importante que si nous effectuions autant de tests manuels).

Il est important de remarquer que le diagramme de Kiviat ne peut être utile que si les axes sont correctement normés selon l'importance donnée à chaque critère.

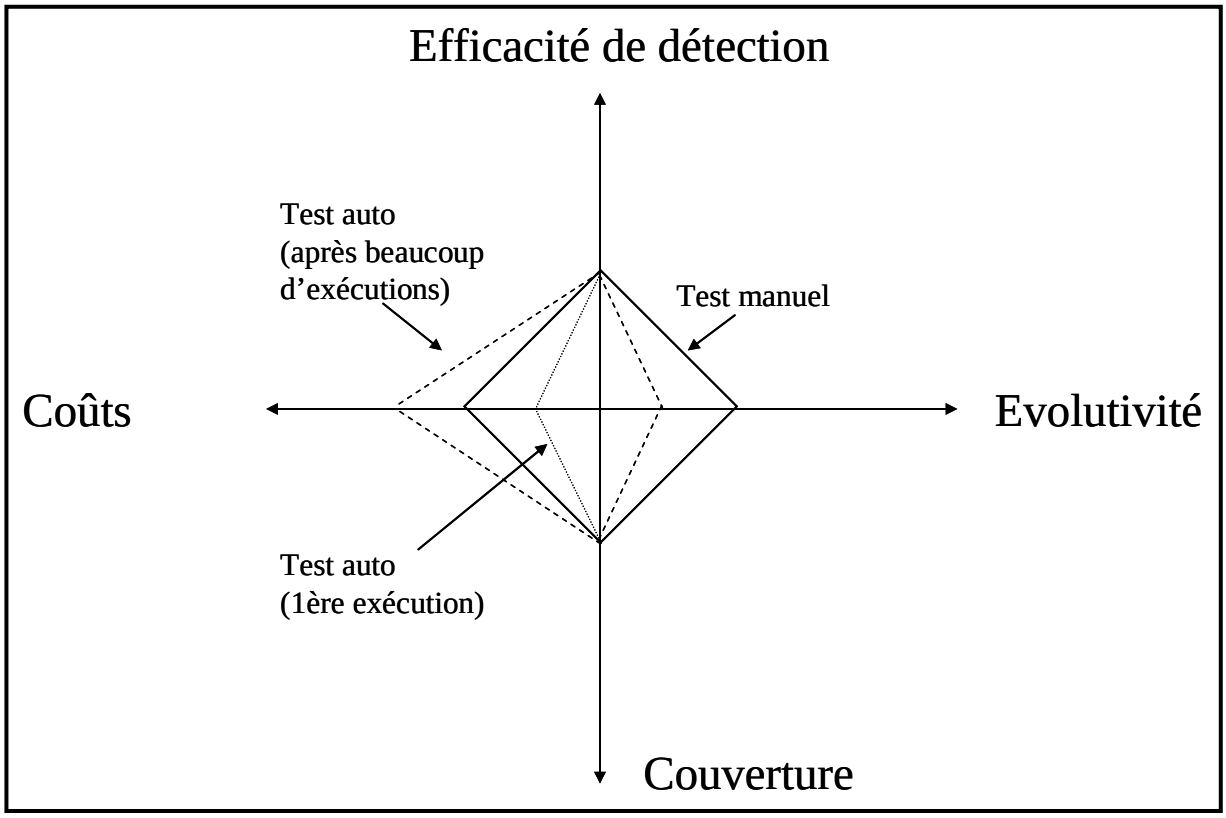


Figure 5.11 : Diagramme de Kiviat [66]

En conclusion, l'automatisation ne peut être complète et les tests automatisés ne remplaceront jamais les tests manuels. En effet, les programmes de tests automatiques ne possèdent pas de faculté d'adaptation pour réagir de manière efficiente (mener l'enquête) et obtenir des informations sur les causes de l'erreur. Dans certains cas, ils forment une aide qui permet d'exécuter les cas de tests plus rapidement, plus souvent et de manière plus rigoureuse.

5.6.3 Les programmes de tests automatisés

Tous les programmes de tests automatiques ne sont pas utilisables avec un logiciel. Nous pouvons prendre l'exemple de COSMICXpert et de SMXpert. L'utilisation du Logiscope sur ces derniers est vaine. En effet, la particularité de ce programme est d'analyser le code source

et d'avertir l'ingénieur logiciel des erreurs possibles. Cependant, les applications citées précédemment sont un mélange de Java et de JSP ce qui limite l'analyse des fonctionnalités.

Dans notre cas, il était donc préférable de se limiter aux outils qui interagissaient avec l'application via son interface. Cela convenait particulièrement bien pour éviter de devoir résoudre le problème épineux de l'entremêlement des technologies utilisées, mais aussi pour le fait que nous nous trouvions dans une restructuration avec une phase de tests d'acceptation. Ces tests étant basés sur les cas d'utilisation, ils représentent réellement les interactions entre l'utilisateur et l'application via les interfaces.

Rational ROBOT et Rational SiteCheck, dans une moindre mesure, font partie de ces outils de tests automatiques qui se basent sur les interfaces. Notre choix s'est donc porté sur eux.

5.6.4 Suite Rational (IBM)

Pentti Pohjolainen fournit un mot d'explication sur Rational ROBOT dans son document « Software testing tools » [65] :

Automated functional testing tool. Allows user to create, modify, and run automated functional, regression, and smoke tests for e-applications built using a wide variety of independent development environments. Rational TestManager and Rational SiteCheck are included, enhancing ability to manage all test assets and have them available to all team members; includes capabilities for Web site link management, site monitoring, and more. Platforms: Windows

L'ensemble de la suite vient en support Rational Unified Process (RUP), proposé par Philippe Kruchten et Ivar Jacobsen. RUP est une méthodologie supportant UML et qui propose une approche itérative pour le développement orienté objet de systèmes. Cette approche se focalise sur ses propres méthodes et laisse peu de place à l'intégration d'autres méthodologies.

5.6.4.1 Test Manager

Rational TestManager est le logiciel de la Suite Rational dédié à la planification de tests qui engendre, entres autres, la rédaction du plan de tests. Cette planification est une étape très importante car elle permet à l'équipe de développeurs de mesurer et gérer l'effort de test au cours du projet. En effet, durant celle-ci, l'équipe identifie les types de tests à effectuer, les stratégies de déploiement et les ressources nécessaires.

Tout artefact du projet (spécifications, code source,…) peut être utilisé par Rational TestManager pour planifier, ébaucher et déployer les cas de tests. Et ainsi tester le système sous tous ses aspects.

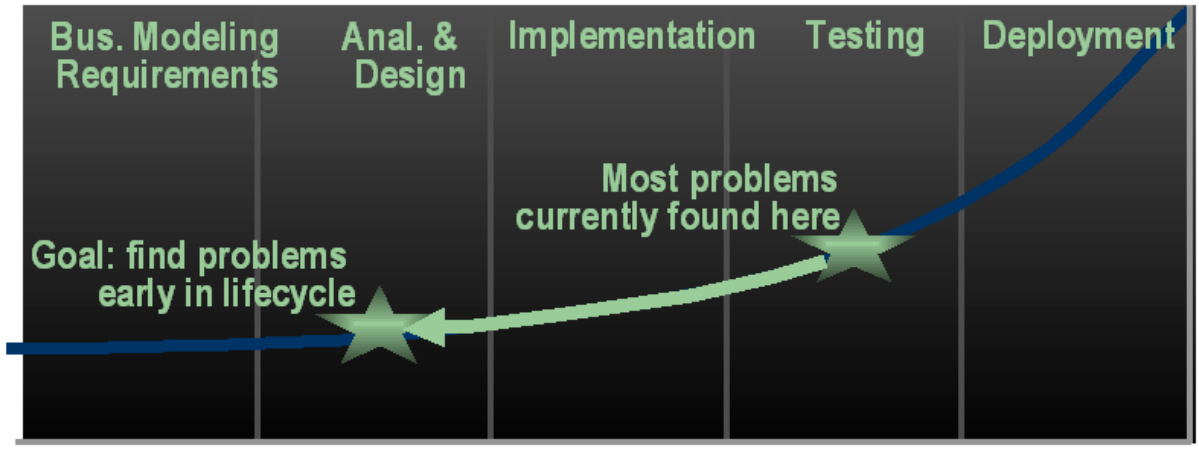


Figure 5.12 : Schéma représentant la quantité d'erreurs détectées en fonction des étapes du cycle de vie

Comme représenté sur la figure précédente, la planification doit commencer tôt dans le processus de développement afin de minimiser les impacts négatifs d'erreurs présentes dés le départ. Rational TestManager permet, grâce à une forte intégration avec les autres outils de la suite, de commencer cette planification à temps.

Un des grands avantages de Rational TestManager est qu'il fournit l'accès à toutes les informations liées aux tests. En effet, il aide à garder une trace du nombre de cas de test qui ont été planifiés, écrits et effectués. De plus, lorsqu'un test a été joué, le logiciel affiche si le cas de test a réussi ou échoué et il permet d'obtenir des statistiques sur bases de ces résultats. Comme tous les membres de l'équipe peuvent accéder à Rational TestManager, nous

comprenons que le logiciel facilite le partage, au sein de l'équipe, des informations à propos des progrès de la phase de test.

Rational TestManager combiné à Rational Robot offre également la possibilité de jouer des cas de test automatiquement (scripts). Il permet aussi d'appliquer des points de vérifications afin de vérifier si les tests sont passés avec succès ou non (cf. Annexe 7).

En résumé, Rational TestManager fournit un support intégré pour implémenter et mener les tests fonctionnels créés dans Robot.

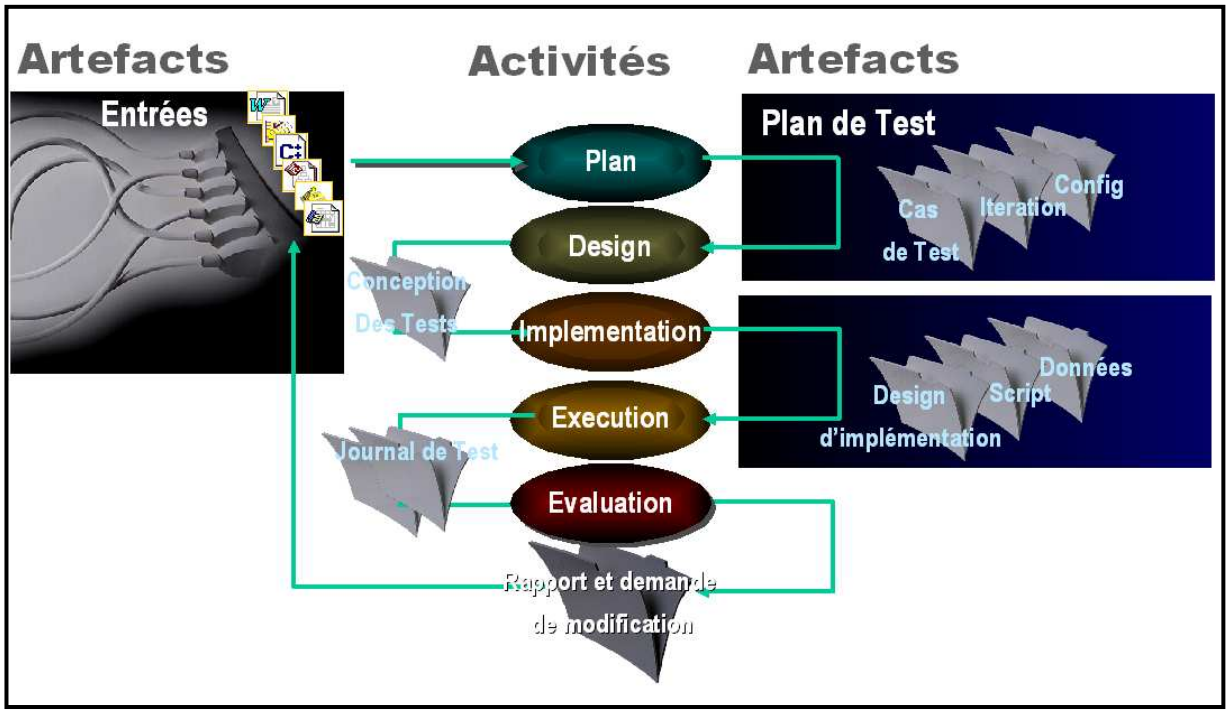


Figure 5.13 : Le cycle de vie de développement et d'exécution de tests

5.6.4.2 Rational Robot

Rational ROBOT entre dans la catégorie des outils de tests automatisés qui effectuent leurs tests automatiques sur base de l'interface graphique et des propriétés des composants (même cachées) qui forment celle-ci. Il permet d'exécuter les tests mais aussi d'assister l'utilisateur pour leur création.

Pour ce qui est des tests, il permet de réaliser automatiquement des tests de type fonctionnels (tests d'acceptation, …) ou des tests de performance. Dans la première éventualité, les cas de test prennent la forme de scripts, exprimés en SQABasic compatible avec Visual Basic. Ces scripts peuvent être modifiés pour pouvoir les adapter à d'autres applications à tester ou pour simplement les améliorer. Dans le second type de tests, c'est Test Manager qui contrôle Rational Robot pour vérifier que le système offre toujours des performances correctes dans des conditions de charge variables.

Grâce à Rational ROBOT, nous pouvons également organiser des tests de régression. Cela consiste en la comparaison de la dernière version de l'application avec les précédentes grâce à l'action de rejouer les tests rédigés auparavant. En exécutant plusieurs fois les scripts pendant le cycle de développement, l'équipe de tests peut se rendre compte de différences apparues ou du fait que l'équipe de développement n'a pas introduit de bogues. Pour ce qui est des différences, elles peuvent être délibérées ou non voulues. Dans ce dernier cas, une correction sera nécessaire.

ROBOT et TestManager sont fortement liés car le second permet de lancer les scripts (ou d'exécuter l'entièreté d'un plan de tests) et de récupérer les résultats pour les afficher et permettre à l'utilisateur de les analyser. En général, un script est lié à un cas de test du plan.

Pour utiliser Rational ROBOT, il est vivement conseillé que le plan de test, et donc les cas de test, ait été réalisé avec TestManager. Dans le cas contraire, nous nous retrouverions avec des scripts ne reposant pas sur une base théorique. En effet, ce serait comme si la phase de rédaction d'un plan de tests, à partir des documents provenant de la phase d'analyse des exigences, avait été abandonnée. Alors, les scripts auraient été créés de manière ad-hoc.

Pour ce qui est du **fonctionnement** de ROBOT, il est principalement basé sur la création de scripts intimement liés aux scénarios de tests. Il offre une assistance à l'utilisateur pour cette création, cela consiste à l'enregistrement des interactions d'un utilisateur avec une Interface graphique Homme-Machine. En effet, il n'existe rien de plus naturel pour tester fonctionnellement (tests d'acceptation) et valider le comportement d'une application.

En plus de créer des scripts, il faut pouvoir juger le déroulement de leur exécution. En effet il faut pouvoir déterminer si le cas de test qu'il représente correspond à une réussite ou à un

échec. Si on se limitait à l'enregistrement d'interactions, alors ROBOT ne posséderait pas les indices nécessaires pour se prononcer sur le bon déroulement d'un script. Pour cela, nous allons introduire des points de vérification lors de l'enregistrement.

Les points de vérification correspondent à des vérifications qui sont faites sur les propriétés, à un moment donné du test et non nécessairement visibles, des objets constituant l'interface (emplacement d'un bouton, texte dans un champ, …). Cela correspond généralement à vérifier qu'une propriété égale ou n'égale pas une certaine valeur. Nous pouvons dire qu'au final un cas de test réussi correspond à retrouver les zones dans lesquelles ont été entrées les données et se retrouver dans l'état final décrit dans son script.

L'équipe de tests peut définir autant de points de vérification qu'elle le souhaite. Ils se trouvent dans le code du script et peuvent être modifiés *a posteriori*. Dans ce dernier cas, il faut réussir à bien pouvoir se représenter dans quel état va se retrouver l'interface au moment où on définit le point de vérification. Sinon, le cas de test échouera à chaque fois du fait de ce point de vérification qui ne trouve pas l'objet qu'il veut vérifier et en conclu, donc, un problème.

Après l'exécution des tests, les résultats des points de vérifications sont affichés dans TestManager et peuvent être consultés en détails. Pour chaque cas de test, le résultat peut prendre une des trois valeurs :

- Pass : Les points de vérification ont donné les bons résultats et le script est bien arrivé à terme.
- Warning : Le script n'est pas arrivé à terme. Par exemple, ROBOT a terminé l'exécution du script après ne pas avoir trouvé un objet permettant la continuation.
- Fail : Un ou plusieurs point de vérification n'ont pas donné les résultats souhaités.

Les contraintes de Rational ROBOT sont multiples. Par exemple, à chaque exécution d'un test, le système doit se trouver dans les exactes conditions pour lesquelles le script a été prévu (lors de son enregistrement). Si plusieurs instances du même navigateur (Microsoft Internet Explorer dans ce cas), alors ROBOT peut ne pas exécuter le test car il ne saura pas se décider sur quelle fenêtre choisir. Cela est principalement du au langage de script fortement lié au système d'exploitation Microsoft Windows.

De plus, nous avons expliqué précédemment les problèmes liés à l'interopérabilité de la suite Rational avec les différentes plateformes. Sa limitation à l'environnement lié au système d'exploitation Microsoft Windows réduit sa puissance.

5.6.4.3 Site Check

Rational SiteCheck permet de tester des applications avec une interface Web. A partir d'une URL, il va vérifier automatiquement tous les hyperliens (en se concentrant sur la détection des liens cassés et les pages orphelines) et analyser les temps de réponse.

Pour ce qui concerne COSMICXpert et SMXpert, les pages ne se succèdent pas en cliquant sur un simple lien. En général, l'utilisateur de l'application doit introduire des informations avant de pouvoir accéder à d'autres pages de l'interface. Ce contenu interactif peut aussi être vérifié en utilisant une fonctionnalité de Rational SiteCheck appelée « ActiveScan ». Celle-ci simule automatiquement les entrées d'un utilisateur, mais il y a une nécessité d'accompagnement par l'équipe de tests, ce qui rend la procédure assez paradoxale pour de l'automatisation.

Il existe également un lien fort entre SiteCheck et TestManager. En effet, ce dernier peut accéder aux résultats obtenus, ce qui permet de les analyser et de suivre la progression des corrections.

5.7 Tests pour SMXpert

A la fin de la restructuration, nous disposons du logiciel à base de connaissances SMXpert. Comme nous l'avons vu dans le Chapitre 3, la plupart des fonctionnalités entre COSMICXpert et le précédent sont équivalentes. Seules quelques adaptations et ajouts de fonctionnalités ont eu lieu. Cette caractéristique nous a permis de réutiliser le plan de tests que nous avions rédigé précédemment pour les fonctionnalités communes entre les deux logiciels. Cependant, par manque de temps, nous n'avons pas pu consigner de manière écrite les adaptations du plan de tests et les cas de test effectués. Les tests de l'application SMXpert ont

été joués de manière *ad hoc*, c'est-à-dire au gré des fonctionnalités nouvellement implémentées.

Au départ, nous avons envisagé de réutiliser les tests automatisés créés avec Rational ROBOT dans le cadre des tests de COSMICXpert. Cependant, plusieurs constatations indiquent qu'il faudrait adapter les scripts :

- Ils sont très sensibles à la configuration de l'environnement dans lequel ils sont exécutés.
- Ils sont très sensibles à l'enchaînement des différentes pages de l'interface.
- Ils contiennent des points de vérification qui se basent sur des caractéristiques d'objets de l'interface.
- En général, les caractéristiques des objets de l'interface correspondent à du texte. En effet, celui-ci varie en fonction du langage (cf. l'internationalisation, section 4.5.2.2) ainsi que d'un logiciel à base de connaissances à un autre. Par exemple, le terme « Topological Concept » est devenu « Maintenance Concept »

L'adaptation manuelle des scripts aurait demandé de démontrer une connaissance en SQABasic dont nous ne disposons pas ou d'effectuer des réenregistrements des interactions avec l'interface. Comme nous l'avons vu précédemment dans le chapitre, la réalisation de cette dernière possibilité reviendrait à exécuter les tests manuellement. En effet, les tests automatiques ne sont profitables que si l'équipe de tests prévoit de les réutiliser un grand nombre de fois et non une seule fois.

Chapitre 6 : Amélioration de la qualité et ébauche d'un cadre de référence

Après avoir expliqué les bases théoriques de notre démarche dans les chapitres précédents, nous allons maintenant traiter de l'amélioration de la qualité du logiciel engendrée par cette démarche.

La Section 6.1 donne un bref aperçu de divers travaux évoqués dans la littérature et consacrés à la restructuration de logiciel. Ceux-ci pourraient nous guider dans la généralisation de notre démarche expérimentale. La Section 6.2 examine dans quelle mesure les concepts théoriques évoqués dans les chapitres précédents ont pu s'appliquer dans la pratique. Nous y évoquerons également les obstacles rencontrés au cours de la restructuration de COSMICXpert. Enfin, dans la Section 6.3, nous présenterons, sur la base des observations précédentes, une ébauche de *cadre de référence* destiné à la restructuration[10] d'un logiciel à base de connaissances. La principale caractéristique de ce dernier sera de proposer une approche amenant une amélioration de la qualité du logiciel, avant et après sa restructuration. L'objectif de ce *cadre de référence* sera d'être le plus générique possible, afin de permettre la restructuration de logiciels à base de connaissances portant sur différents domaines.

6.1 Pistes de réflexions pour la généralisation de la démarche

La question soulevée dans cette section est de savoir si notre démarche expérimentale englobe tout ce dont nous avons besoin pour la généraliser. Nous posons la question de savoir s'il existe des étapes supplémentaires lorsque la démarche prend place dans un autre contexte. De plus, nous tentons de trouver dans la littérature, des exemples capables de nous guider dans notre ébauche d'un *cadre de référence*.

Nous trouvons dans [62] un *cadre de référence* de réingénierie guidée par les exigences. Cette approche semble concorder en plusieurs points avec la notre. Tout d'abord, notre

[10] Par restructuration d'un logiciel à base de connaissances, nous faisons référence à la démarche particulière illustrée au cours des chapitres précédents et visant à modifier le domaine de connaissances du logiciel.

restructuration est fortement liée aux exigences. Ensuite, ces travaux mettent l'accent sur l'amélioration de la qualité. Cette référence ne s'applique cependant pas dans son entièreté. En effet, la source citée concerne la réingénierie de systèmes *legacy*. Or, comme expliqué dans les chapitres précédents, les transformations appliquées au cours de notre démarche ne sont pas si profondes. Elles sont plutôt de l'ordre de la restructuration. Cependant, nous pouvons mettre en exergue des points communs avec nos objectifs et retirer des informations précieuses.

Premièrement, nous pouvons remarquer que le *cadre de référence* concerné met l'accent sur la définition de buts. En second lieu, nous remarquons que la première étape de ce *cadre de référence* concerne justement l'établissement de règles régissant la suite des opérations

D'autre part, nous trouvons en [63] un indice supplémentaire concernant la légitimité de l'amélioration de qualité que peut apporter la restructuration. La citation suivante évoque cet indice :

> Software restructuring is the modification of software to make the software easier to understand and to change, or less susceptible to error when future changes are made. "Software" includes external and internal documentation concerning source code, as well as the source code itself.

Cette assertion étant vérifiée pour la restructuration au sens général du terme, elle se vérifiera très probablement pour le cas particulier d'une restructuration de logiciel à base de connaissances. En ce sens, elle apporte un fondement théorique plus fort à notre esquisse de *cadre de référence*. De plus, la dernière phrase de la citation nous permet de mettre en évidence l'importance de la documentation autant que celle du code. L'acquisition de cette documentation devra donc jouer un rôle dans le *cadre de référence*. Enfin, [63] nous fournit une liste très complète d'outils de restructuration permettant d'étoffer notre *cadre de référence*.

Concrètement, les sources examinées semblent nous orienter vers l'ajout d'une étape précédant toutes celles de notre approche expérimentale. Cette dernière permettrait de calibrer le *cadre de référence* en fonction du contexte dans lequel se déroule la restructuration. Ainsi, notre *cadre de référence* devrait-il être paramétrable afin de s'adapter à l'environnement et aux exigences des commanditaires de la restructuration. En effet, certains logiciels pourraient

ne trouver aucun intérêt à améliorer un point spécifique de leur qualité (sécurité, temps de réponse, etc.) et il serait alors inutile de produire des efforts concernant cet aspect particulier.

Enfin, la différence entre restructuration et réingénierie (cf. Section 3.1) aura un impact direct sur les composantes du *cadre de référence* que nous tentons d'élaborer. Passer à une phase de réingénierie pure et ignorer tout le code existant pour recommencer de zéro n'est pas ce que nous cherchons, puisqu'une telle approche mènerait à une dépense inutile d'efforts, de temps et, dans de nombreux contextes, d'argent. Le *cadre de référence* devra donc, comme notre démarche expérimentale, mettre l'accent sur la réutilisation des composantes existantes, particulièrement au niveau du code. De plus, le *cadre de référence* devra se limiter à conserver la « forme » de l'application. Notre démarche n'a, par exemple, pas pour but de transformer une application *web-based* en système embarqué.

6.2 La restructuration de COSMICXpert dans la pratique

Phase Initiale	**Etape 1 :** Elaboration d'un plan de tests.
	Etape 2 : Réalisation de la phase de tests.
	Etape 3 : Correction des erreurs et des bogues.
Phase de Restructuration	**Etape 4 :** Analyse architecturale et du domaine cible.
	Etape 5 : *Design* architectural.
Phase de Réalisation	**Etape 6 :** *Design* technique et implémentation.
Phase Finale	**Etape 7 :** Phase de tests et corrections.
	Etape 8 : Migration de la base de connaissances source.

Tableau 6.1 : Liste des étapes de la démarche de restructuration.

6.2.1 Phase Initiale

Notre travail de restructuration de COSMICXpert a débuté par une phase de tests d'acceptation du logiciel existant (cf. Chapitre 5).

L'**Étape 1** de notre démarche consiste à élaborer d'un plan de tests (cf. Annexe 5). Cette étape nécessite un travail de prise en main du système indispensable pour la bonne compréhension de celui-ci (cf. Chapitre 5), et la bonne marche des opérations futures. La rédaction du plan de tests a été réalisée sur base de la documentation du système existant ainsi que de l'expérience acquise lors de la manipulation du système sur des exemples *ad hoc*. Par exemple, nous pouvons citer les tentatives d'ajout d'un concept topologique à la base de connaissances. L'étape 1 nous permet donc de détecter les écarts entre les faits annoncés dans la documentation et les faits observés dans la pratique. En effet, il existe certaines différences entre la manipulation annoncée par les cas d'utilisation ou le manuel de l'utilisateur, et la manipulation concrète implémentée par les développeurs. La rédaction du plan terminée, nous sommes en mesure de prendre ces divergences en compte dans la réalisation des étapes suivantes.

L'**Étape 2** de notre travail a consisté en l'exécution des tests pour en arriver à un rapport le plus complet possible (cf. Annexe 5). Le but premier de cette tâche est d'apporter de l'information sur la qualité du logiciel existant. Nous sommes alors en mesure de cerner les principales défaillances du système (au niveau fonctionnel) ainsi que le degré de respect du produit fini par rapport à ses spécifications. De plus, la familiarité acquise avec le logiciel au cours des tests nous permet de découvrir un certain nombre de problèmes d'ordre non fonctionnel, tels que des questions d'ergonomie, de temps de réponse ou de sécurité.

L'**Étape 3** consiste en la correction des erreurs et des bogues découverts à l'**Étape 2**. Elle doit nous apporter une meilleure connaissance du logiciel existant, puisqu'elle nécessite d'entrer plus avant dans le système, au niveau du code et de la technique. La correction effectuée, nous sommes en mesure de comprendre les principaux rouages du logiciel ainsi que de son implémentation. De plus, nous pouvons mettre en évidence certaines faiblesses de programmation s'expliquant par le fait que plusieurs générations successives d'ingénieurs

logiciel ont travaillé sur le projet. Enfin, nous bénéficions de la certitude de commencer la restructuration (*sensu stricto*) du logiciel sur la base d'un produit fini et connu.

A la fin de la **Phase Initiale**, nous sommes en mesure de cerner correctement le système existant, tant au niveau architectural que technique. Nous connaissons le système, d'un point de vue *utilisateur* ainsi que d'un point de vue *développeur*. Nous pouvons donc raisonner sur ce logiciel, être plus critique et commencer la restructuration proprement dite. De plus, nous pouvons affirmer que la qualité du logiciel existant a déjà bénéficié d'une amélioration de qualité. Cette amélioration se marque principalement au niveau fonctionnel, c'est-à-dire au niveau de la *correction* des services offerts et non pas des performances offertes. Nous avons en effet amélioré l'efficacité du code mais des aspects tels que la lisibilité de celui-ci ou la maintenabilité du logiciel n'ont pas encore été pris en compte. Cette affirmation est justifiée puisque la finalité même d'une phase de tests est d'améliorer la qualité du système testé.

6.2.2 Phase de Restructuration

L'**Étape 4** de notre démarche a eu un double objectif. D'une part, nous restructurons la documentation de l'architecture existante. D'autre part, nous vérifions que la structure de l'ontologie de COSMICXpert permet de modéliser les connaissances de SMXpert. Nous déterminons également dans quelle mesure elle nécessite une adaptation. Toutefois, même si l'on adapte la structure de l'ontologie, la manipulation peut être différente. Le Chapitre 1 illustre d'ailleurs que l'architecture de la base de connaissances et le logiciel qui la manipule peuvent évoluer de manière indépendante. Nous devons donc vérifier la possibilité de modéliser le domaine cible avec le formalisme utilisé par la base de connaissances existante. Mais nous devons de surcroît nous assurer que les raisonnements appliqués aux connaissances du logiciel existant sont pertinents et efficaces pour traiter les connaissances du domaine cible. Concrètement, nous devons nous appuyer sur les conseils et avis d'une équipe d'experts[11] dans l'un et l'autre domaine. A la fin de cette étape, nous connaissons les besoins

[11] Avec l'aide des Professeurs Desharnais et April, ainsi que de Mlle Fadila Oudjehane, nous avons pu déterminer le degré de compatibilité entre l'ontologie de COSMICXpert et les exigences de SMXpert. Nous avons alors pu déduire les adaptations nécessaires sur les structures de données de COSMICXpert. Ensuite, nous avons défini les types de raisonnements applicables à ces structures modifiées. Cette étape nous a donc livré les

de l'utilisateur et de l'expert du domaine cible. Nous avons une idée claire du travail de restructuration à faire et savons que le passage du domaine source au domaine cible est réalisable.

L'**Étape 5** consiste alors en une phase de *design* architectural, nécessaire à la réalisation du projet. Celle-ci constitue la pierre angulaire de notre démarche. En effet, elle nous permet d'améliorer le plus significativement la qualité du logiciel. Durant la phase de *design* architectural, nous avons pris la décision de modifier l'architecture du système existant. Ce sont les résultats des réflexions menées alors qui sont exposés dans les Chapitres 3 et 4. Les options retenues durant le *design* architectural de la nouvelle solution sont étroitement liées aux observations effectuées durant la **Phase Initiale**. En effet, la connaissance du logiciel existant et de ses faiblesses (surtout au niveau non fonctionnel) permet de corriger ces dernières et d'améliorer les aspects tels que le temps de réponse ou la facilité de maintenance.

A la fin de la **Phase de Restructuration**, nous avons un *cahier de charges* pour l'implémentation du nouveau système et une solution plus générique pour améliorer la structure du logiciel. Cette phase complétée, nous sommes en mesure de commencer l'implémentation du logiciel cible. De plus, nous pouvons penser que ce dernier gagnera en qualité puisque les choix posés sont orientés sur cette volonté d'amélioration.

6.2.3 Phase de Réalisation

L'**Étape 6** est la seule étape de la **Phase de Réalisation**. Elle consiste à mettre en place le *design* technique et à l'implémenter. Il s'agit de l'étape la plus importante en terme d'efforts et aussi la plus complexe en terme de réalisation. Elle reste la plus délicate. En effet, les obstacles inhérents à la réalisation du logiciel reposent sur l'écart présent entre les choix technologiques posés ici (cf. Chapitre 4) et les technologies présentes dans le système d'origine. Afin de garantir que notre démarche est bien une restructuration et non un cycle de développement pur, nous devons conserver un maximum de compatibilité, et favoriser la réutilisation des composants déjà développés. La modularité de notre architecture nous a

---

modifications clés à appliquer au moteur d'inférence et à la base de connaissances de COSMICXpert afin de le restructurer en SMXpert.

permis de réutiliser des composants existants sans devoir les recréer. Elle a également permis de migrer certains traitements à des endroits plus appropriés. De plus, la mise en œuvre de technologies plus adaptées permet de simplifier des traitements jusqu'alors fastidieux. A ce stade, maîtriser les technologies mises en oeuvre est primordial. En effet, le principal frein et la principale source de retard sont la mise en œuvre des nouvelles technologies. Plus encore, il est indispensable de maîtriser les interactions entre celles-ci et les technologies présentes dans le logiciel originel.

A la fin de la **Phase de Réalisation**, nous avons généré le code et la documentation pour notre produit final. Nous sommes en possession du logiciel cible et celui-ci est fonctionnel. Une version préliminaire du logiciel a été placée sur un serveur afin de procéder à la **Phase Finale.**

6.2.4 Phase Finale

L'**Étape 7** de notre travail consiste en de nouveaux tests et corrections, mais cette fois-ci sur le logiciel restructuré. Nous nous assurons de la qualité par des tests. Ce travail est facilité par l'existence d'un plan de tests issu de l'**Étape 1** et lié au logiciel source. En effet, puisque les deux applications présentent les mêmes fonctionnalités (cf. Chapitre 3), le plan de tests de COSMICXpert est largement réutilisable pour SMXpert. Les nouvelles fonctionnalités pourraient faire l'objet d'un plan de tests annexe structuré mais, par manque de temps, ces fonctionnalités ont été testées de façon *ad hoc* (cf. Chapitre 5). Des tests plus structurés, tels que ceux réalisés dans la **Phase Initiale** pour COSMICXpert, doivent encore être réalisés avec SMXpert. Cette étape complétée, nous possédons un logiciel restructuré dont le domaine est bien le domaine cible. De plus, nous possédons des indices raisonnables (plan de tests réutilisés et tests *ad hoc*) que le logiciel restructuré est *de qualité* sous ses aspects fonctionnels. Nous attendons de cette nouvelle version des performances améliorées, sans pourtant pouvoir l'affirmer de façon certaine puisque les tests systématiques réalisés à la phase initiale n'ont pu être reproduits.

L'**Étape 8** de notre démarche expérimentale consiste à transposer la base de connaissances de COSMICXpert directement dans la structure de SMXpert. Il devrait être facile de transposer les connaissances pour les raisons suivantes.

- Premièrement, l'architecture d'un logiciel à base de connaissances peut évoluer indépendamment de l'architecture de sa base de connaissances elle-même (cf. Chapitre 1). De ce fait, l'évolution de la structure du logiciel ne le rend pas forcément incompatible avec son ancienne base de connaissances.
- Deuxièmement, notre manière de faire évoluer les structures de données postule de ne jamais abandonner d'éléments ou d'attributs, même s'ils n'ont pas d'utilité pour les connaissances du domaine cible (cf. Chapitre 3). Aucune connaissance du domaine source n'est donc impossible à modéliser dans la nouvelle base de connaissances.
- Troisièmement, les choix architecturaux posés à l'**Étape 5** mettent l'accent sur la modularité. Ils permettent donc une plus grande flexibilité du moteur d'inférence. L'encapsulation des traitements propres à ce dernier rend notamment possible son adaptation à moindre effort. Elle le rend également plus générique.

L'**Étape 8** fera l'objet d'une recherche qui débutera à l'automne 2005.

Cependant, cette étape reste optionnelle de par la nature imprévisible des améliorations de qualité. Le fait de réimplanter la base de connaissances originelle dans le logiciel restructuré n'a de sens que si les améliorations de la nouvelle structure apportent un gain réel pour le logiciel source. Notons que même sans passer par l'**Étape 8**, le logiciel originel aura gagné en qualité (pour les exigences fonctionnelles). En effet, dès l'**Étape 3** complétée, il bénéficie déjà d'avancées considérables dues à la correction des bogues et erreurs.

A la fin de la **Phase Finale,** nous devons disposer d'un logiciel restructuré *de qualité*. De plus, le logiciel restructuré devrait avoir aussi gagné en qualités non fonctionnelles. Et le logiciel source devrait pouvoir bénéficier des mêmes améliorations non foncionnelles.

6.3 Elaboration d'un cadre de référence de restructuration

Sur base de notre propre expérience et des observations théoriques de la Section 6.1, nous définissons une ébauche de *cadre de référence* permettant : la restructuration d'un logiciel à base de connaissances, le changement du domaine de connaissances traité, et l'amélioration de la qualité de ce logiciel.

6.3.1 Objectifs et limites du modèle

Les objectifs du *cadre de référence* liés à la restructuration et au changement de domaine sont clairement précisés dans les premiers chapitres. Il nous reste à traiter de l'amélioration de qualité que l'on peut obtenir par l'utilisation du *cadre de référence*. Il nous faut aussi aborder les limitations de ces améliorations, ainsi que leur champ d'application. L'objectif d'amélioration de la qualité mérite quant à lui que nous nous y attardions un peu plus. La question que nous soulevons ici se rapporte au degré d'amélioration que nous pourrons retirer de l'utilisation du *cadre de référence*, ainsi qu'à ses limitations et à son champ d'application. Les limitations à retenir sont les suivantes :

- Notre démarche de restructuration vise un logiciel à base de connaissances. De ce fait, nous ne pouvons garantir que le *cadre de référence* ébauché peut s'appliquer à toute autre forme de logiciel.
- Il s'agit d'un outil qui vise uniquement les applications basées sur Internet.
- De plus, notre démarche s'appuie principalement sur les possibilités offertes par le paradigme Orienté Objet, tant par l'architecture (modularité) que dans l'implémentation. Nous ne pouvons donc pas assurer des gains pour des applications sortant de ce canevas précis.

Toutefois, même avec ces conditions réunies, il nous reste à déterminer dans quelle mesure nous attendons une amélioration, et sur quels points précis la qualité pourrait augmenter. Pour ce faire, nous nous référons à la liste de critères de réussite d'une application web élaborée en [64]. Nous y distinguons trois critères principaux (*Reliability, Usability, Security*) suivi de quatre critères additionnels (*Availability*, *Scalability*, *Maintainability, Time-to-market*). Nous pouvons espérer une amélioration liée à ces sept critères dans des mesures raisonnables. Mais la démarche expérimentale à la base de cette ébauche montre que le gain de qualité résiderait plus particulièrement dans le niveau de maintenabilité, de disponibilité et de sécurité. Nous pouvons donc espérer que le *cadre de référence*, généralisant celle-ci, agira sur les mêmes critères.

Il est à noter également que le gain de qualité du système après sa restructuration à l'aide du *cadre de référence* devrait être incrémental. En effet, chaque restructuration par ce procédé devrait amener un gain de maturité et de performances. L'expérience devra montrer dans

quelle mesure l'amélioration de la qualité est vérifiée et, jusqu'à quel point celle-ci peut augmenter par des restructurations successives.

6.3.2 Présentation du *cadre de référence*

6.3.2.1 Phase Préliminaire : Spécifications et Acquisition

Les pistes de réflexion suivies à la Section 6.1 suggèrent la nécessité d'un travail préliminaire à la démarche expérimentale. Celui-ci engloberait un paramétrage des objectifs poursuivis par la démarche (par exemple, choisir de se concentrer seulement sur la maintenabilité de l'application) et une familiarisation avec le logiciel à restructurer. Bien que la démarche présentée à la Section 6.2 ne le mentionne pas, nous avons bien effectué un tel travail. Cependant, nous l'avons fait de manière implicite, au travers de la préparation de notre stage. Afin de rendre le cadre de référence plus généralisable, nous devons systématiser ce travail préliminaire.

Notre nouveau *cadre de référence* prévoit donc l'ajout d'une **Phase Préliminaire.** Celle-ci permet de prendre en compte les objectifs poursuivis par l'équipe de restructuration et le contexte de cette dernière.

Phase Préliminaire	**Etape 0a :** Définitions des exigences du système restructuré.
	Etape 0b : Elaboration des *lignes directrices*.
	Etape 0c : Acquisition de connaissances.
	Etape 0d : Acquisition de la documentation.

Tableau 6.2 : Liste des étapes de la phase préliminaire

La **Phase Préliminaire** comprend quatre étapes :

L'**Étape 0a** permet de déterminer les objectifs de la restructuration en cours. Toutefois, au sein de ce *cadre de référence*, la *définition des exigences* recouvre deux sens distincts. Il s'agit certes de déterminer les exigences fonctionnelles et non fonctionnelles du système restructuré (cf. Chapitre 3). Mais, de plus, une liste rigoureuse des améliorations de qualité

attendues devrait être dressée. Celle-ci comportera des informations sur les types de critères visés mais surtout sur la mesure dans laquelle une amélioration est attendue. Cette définition faite, les choix posés durant les étapes ultérieures pourront s'en inspirer. Cette liste permettra donc de mieux cibler les efforts d'amélioration liés à la restructuration.

L'**Étape 0b** visera à définir les *lignes directrices* de la restructuration. Concrètement, l'équipe devra définir les méthodes et règles à suivre lors de la restructuration[12]. Cette étape permettra d'améliorer la précision des exigences puisque le choix des *lignes directrices* aura tendance à prioriser ces dernières. Ainsi, si l'étape précédente décrivait le *quoi* du problème, la présente décrirait plutôt le *comment*. Les méthodes et règles élaborées pourront être définies depuis zéro ou puisées dans la littérature, présentant de nombreux outils de restructuration, comme évoqué dans [64], pour peu qu'elles restent en accord avec le contexte particulier des logiciels à base de connaissances.

L'**Étape 0c** permet le déroulement d'une tâche indispensable : l'*acquisition de connaissances.* Elle prévoit que l'équipe de restructuration se familiarise avec le domaine de connaissances du logiciel source. Cette familiarisation n'implique pas que l'équipe devienne elle-même experte dans le domaine. Mais une maîtrise de base des concepts clés du domaine source est indispensable aux étapes ultérieures.

L'**Étape 0d** offre la possibilité d'adapter l'usage du cadre de référence à divers contextes. En effet, Il faut noter que le contexte particulier dans lequel se plaçait le travail mené à l'ETS nous a permis de passer une étape qui, en d'autres circonstances, aurait pu s'avérer indispensable : la phase de rétro-ingénierie. Nous avons bénéficié d'une documentation importante et fonctionnelle de part la nature même du projet. Toutefois, la toute première phase de notre *cadre de référence* prévoit une étape d'acquisition de documentation et/ou de rétro-ingénierie. Cette étape 0d permettra de retrouver une documentation de base dans un contexte où celle-ci serait inexistante. Ceci assure à notre outil une certaine flexibilité quant aux projets auxquels il s'applique.

[12] Notons que cette étape correspond aux réflexions illustrées à la Section 3.2.1.2

6.3.2.2 Cadre de référence et observations

Tel que mentionné, nous avons ajouté une **Phase Préliminaire** de « paramétrage ».

Notons enfin que :

- L'**Étape 4** peut constituer un point de sortie prématuré du *cadre de référence*. Si le domaine cible s'avère totalement incompatible avec les traitements de la connaissance existants, la suite de la restructuration n'aura plus lieu d'être. On lui préférera peut-être une procédure de réingénieire ou de développement depuis zéro.[13] Toutefois, les quatre étapes de la **Phase Préliminaire** devraient donner des indices suffisants de cette compatibilité à l'équipe. Elles limiteraient donc les cas d'échecs liés au *cadre de référence*.
- L'**Étape 5** (*design* architectural) aura un impact direct sur l'amélioration de la qualité globale du logiciel mais aussi sur les efforts à produire dans la suite. On peut raisonnablement avancer que plus l'effort de *design* architectural sera important, plus la qualité du logiciel restructuré sera augmentée. Mais, *a contrario*, plus l'effort de restructuration sera marqué, plus l'effort de réalisation à la phase suivante sera important. De plus, une attention particulière devra être accordée à la compatibilité arrière des structures de données et de contrôles.

[13] Il est à noter que, même dans le cas ou la procédure prend fin avant terme, le cadre de référence aura quand même engendré une amélioration de qualité, suite à la **Phase Initiale**.

- L'**Étape 6** est plus qu'une simple étape d'écriture de code. L'accent doit être mis sur la réutilisation des composants existants pour assurer des économies de temps, d'efforts et d'argent (si nous généralisons l'utilisation de ce *cadre de référence* au monde de l'entreprise et non plus au seul monde académique). Notre *cadre de référence* propose deux outils principaux pour assurer la récupération du code existant :
  - **L'encapsulation des codes épars dans des classes** : il s'agit de retrouver les traitements épars sur les pages du côté client et de les transformer en librairies internes. Ensuite, le code dans les pages sera remplacé par un appel à ces libraires.
  - **La transformation des classes existantes en librairies externes** : cette manœuvre permet de garantir une haute réutilisabilité. En effet, quelles que soient les classes développées pour les besoins de la restructuration, les anciennes classes doivent être conservées plutôt que détruites afin de pouvoir aisément y faire référence, si besoin.
- L'**Étape 7** devrait être réalisée de façon plus complète et systématique (réalisation d'un plan de tests complet sur bases du plan de la Phase Initiale). Ainsi, le cadre de référence apporterait plus de certitudes quand à la qualité du produit fini. La seule raison justifiant l'absence d'un tel plan dans notre approche pratique est un manque de temps.
- L'**Étape 8** ne prendra son sens que si l'amélioration de qualité sur le domaine cible est pertinente pour le domaine source. Supposons par exemple un domaine source de taille réduite, un domaine cible très vaste et une restructuration axée sur l'amélioration du temps de réponse. Ce cas de figure ne permet vraisemblablement pas de modifier la qualité du logiciel source par le biais de la nouvelle architecture. En effet, les efforts seront probablement axés autour d'un meilleur agencement des fichiers de la base de connaissances afin de traiter l'augmentation de taille. Or, il est très peu probable qu'une telle approche engendre un gain pour un domaine limité en taille, puisque la gestion des fichiers ne constitue pas un problème majeur pour celui-ci.

Phase Préliminaire	**Etape 0a :** Définitions des exigences du système restructuré.
	Etape 0b : Elaboration des *lignes directrices*.
	Etape 0c : Acquisition de connaissances.
	Etape 0d : Acquisition de la documentation.
Phase Initiale	**Etape 1 :** Elaboration d'un plan de tests.
	Etape 2 : Réalisation de la phase de tests.
	Etape 3 : Correction des erreurs et bogues.
Phase de Restructuration	**Etape 4 :** Analyse du domaine cible.
	Etape 5 : Design architectural.
Phase de Réalisation	**Etape 6 :** Design technique et implémentation.
Phase Finale	**Etape 7 :** Phase de tests et corrections.
	Etape 8 : Migration de la base de connaissances source.

Tableau 6.3 : Liste des étapes décrites par le cadre de référence

Le Tableau 6.3 illustre l'ébauche finale de notre cadre de référence. Elle est essentiellement le même que notre démarche expérimentale.

Chapitre 7 : Conclusions et perspectives futures

Il existe de nombreux exemples de logiciels à base de connaissances (KBS) basés sur les technologies WEB. La généralisation de ce type d'applications se justifie par les nombreux domaines de connaissances dont elles peuvent traiter. Les exemples donnés impliquent souvent un développement depuis zéro des applications (base de connaissances *et* logiciel). Pourtant, la littérature nous montre que la base de connaissances et le logiciel qui la manipule peuvent faire l'objet de traitements indépendants. Des méthodes ont donc été développées en vue de mettre en œuvre des mécanismes de réutilisation sur un KBS, évitant des efforts de développement inutiles. Nous pouvons notamment citer les travaux de Schreiber (CommonKADS [70]) et de van Heyst [39]. Ils apportent une réponse théorique la question suivante : une démarche de restructuration d'un logiciel à base de connaissance permet-elle de l'adapter à un nouveau domaine de connaissances ?

Notre démarche a été, quant à elle, essentiellement pratique. Sur base d'un cas soumis (adaptation de **COSMICXpert Web Edition**), nous avons tenté de mener une telle restructuration. Nous avons également entrevu une autre opportunité. En effet, la restructuration pourrait permettre de mener des étapes visant l'amélioration de la qualité, tant du logiciel source et du logiciel cible. Afin de valider ce fait, des méthodes de tests et de mesure sont applicables au cours de notre démarche. Elles ont été appliquées aux étapes 1 à 3 de notre démarche mais pas pour les étapes 7 et 8 (cf. Chapitre 5 et 6).

Pour tenter d'apporter des pistes de réponse à ces questions, nous avons appliqué une démarche de restructuration à **COSMICXpert Web Edition** visant à l'adapter au domaine de la maintenance. Celle-ci nous à permis de produire un logiciel (**SMXpert Web Edition**) dont le domaine de connaissances est différent mais qui repose sur les mécanismes du premier.

Les premières étapes de cette démarche ont permis de gérer la qualité de **COSMICXpert Web Edition** au niveau fonctionnel. Nous avons montré (Chapitre 5) que la démarche suivie avait apporté une amélioration de la *correction* du logiciel. Par contre, l'amélioration des aspects non fonctionnels, ne peut pas être affirmée avec certitude. Une telle affirmation nécessiterait des mesures rigoureuses que nous n'avons pas pu effectuer par faute de temps. Notre démarche est orientée amélioration (cf. Chapitre 3) et nous espérons donc que les

mécanismes mis en oeuvre apportent effectivement l'amélioration prévue. Notons toutefois que des aspects non fonctionnels de la qualité se vérifient plus aisément (par exemple, le fait que de code dans les pages JSP entraîne une sécurité accrue) que d'autres (l'amélioration du temps de réponse nécessiterait une mesure rigoureuse).

Enfin, nous avons vu qu'il était possible de généraliser quelque peu la démarche expérimentale suivie afin qu'elle s'inscrive dans un *cadre de référence* d*e restructuration de logiciel à base de connaissances*. Ce cadre de référence reprend les phases nécessaires à l'appropriation du logiciel source et de son domaine, à l'amélioration de sa correction, au changement de son domaine de connaissances et, enfin, à l'amélioration de la qualité des logiciels source et cible.

Les perspectives futures en rapport à notre travail s'articulent autour de trois axes :

Pour COSMICXpert Web Edition : Comme le prévoit l'étape 8 de notre *cadre de référence* (cf. Chapitre 6), il devrait être possible de transposer aisément la base de connaissances du logiciel avant restructuration, dans le logiciel restructuré. De ce fait, COSMICXpert bénéficierait des éventuelles améliorations non-foncionnelles dues à la restructuration. Un travail pertinent serait d'effectuer cette transposition et d'appliquer diverses méthodes de mesure sur le logiciel obtenu. Ensuite, il serait intéressant de mesurer les efforts réels déployés, afin de juger de l'intérêt et valider le bien fondé de l'étape 8.

Pour SMXpert Web Edition : Le premier travail à venir sera d'étoffer la base de connaissances du logiciel. Il sera alors possible de mener les mesures de qualité de manière rigoureuse, sur une base de connaissances de taille pertinente. Le logiciel devra également être amélioré au niveau des interfaces (surtout la partie expert). Ce travail visera principalement les qualités (non abordées dans ce mémoire) liées à l'*ergonomie* et l'*utilisabilité* de l'application. Il sera facilité par la nouvelle implémentation mettant en oeuvre le *design pattern* « Model-View-Controller » (cf. Chapitre 4).

Pour le *cadre de référence* ébauché : Un travail nécessaire sera de l'appliquer à un nouveau cas, pour ensuite déterminer dans quelle mesure il a rencontré ses objectifs de restructuration et d'amélioration de la qualité.

Références

[1] **Norme ISO/IEC 19761:2003,** “Software engineering -- COSMIC-FFP -- A functional size measurement method”, *2003.*

[2] **F. Gruselin, J. Vilz**, "Vérification et validation d'un système expert pour la mesure fonctionnelle (CosmicXpert)." - *Namur: Facultés universitaires Notre-Dame de la Paix, 2003.*

[3] **C. Duterme, N. Fabry,** "Automatisation de la vérification d’une base de connaissances. Application à l’interface « expert » de CosmicXpert." - *Namur: Facultés universitaires Notre-Dame de la Paix, 2003.*

[4] **Tsai, J.J.P.; Liu, A.; Juan, E.; Sahay, A**.; “Knowledge-based software architectures: acquisition, specification, and verification” - *Knowledge and Data Engineering, IEEE Transactions on Volume 11, Issue 1, Jan.-Feb. 1999 Page(s):187 – 201, Digital Object Identifier 10.1109/69.755628*

[5] **J.-M. Desharnais, A. April,** “Software Maintenance Capabilities: A Decision Support Instrument”, *in SEWORLD, 2004.*

[6] **Smolander, K**.; “Four metaphors of architecture in software organizations: finding out the meaning of architecture in practice” - *2002 International Symposium on Empirical Software Engineering Proceedings, 2002, p 211-21, Database: Inspec.*

[7] **G. Lakoff, M. Johnson**, “Metaphors We Live by.” - *Chigago: The University of Chigago Press, 1980.*

[8] **Bosch, J.; Bengtsson, P.**; “Assessing optimal software architecture maintainability” - *Software Maintenance and Reengineering, 2001. Fifth European Conference on 14-16 March 2001 Page(s):168 – 175, Digital Object Identifier 10.1109/.2001.914981*

[9] **Bass, L.; John, B.E.**; “Supporting usability through software architecture” – *Computer Volume 34, Issue 10, Oct. 2001 Page(s):113 – 115, Digital Object Identifier 10.1109/2.955105*

[10] **Jacobsen, E.E.; Kristensen, B.B.; Nowack, P.**; “Architecture=abstractions over software” - *Technology of Object-Oriented Languages and Systems, 1999. TOOLS 32. Proceedings 22-25 Nov. 1999 Page(s):89 – 99, Digital Object Identifier 10.1109/TOOLS.1999.809417*

[11] **Bosch, J.; Lundberg, L.**; “Software architecture - engineering quality attributes” - *Journal of Systems and Software, v 66, n 3, 15 June 2003, p 183-6, Database: Inspec.*

[12] **Bratthall, L.; Wohlin, C.**; “Understanding some software quality aspects from architecture and design models” - *Program Comprehension, 2000. Proceedings. IWPC 2000. 8th International Workshop on 10-11 June 2000 Page(s):27 - 34 Digital Object Identifier 10.1109/WPC.2000.852477*

[13] **Svahnberg, M.**; “An industrial study on building consensus around software architectures and quality attributes” - *Information and Software Technology, v 46, n 12, 15 Sept. 2004, p 805-18, Database: Inspec.*

[14] **V. Prince**, “Vers une informatique cognitive dans les enterprises: Le rôle central du langage” - *Paris : Masson, 190 p, 1996.*

[15] **J.-M. Desharnais**, "Application de la mesure fonctionnelle COSMIC-FFP: une approche cognitive," - *Montréal: UQAM, 2003, 201 pages.*

[16] **S. M. Ndagijimana**, “Étude du processus de measures de points de function et analyse de la documentation fonctionnelle” – *Namur : Institut d’Informatique, Facultés Universitaires Notre-Dame de la Paix, 2002.*

[17] **T. Küssing**, "Design and implementation of a diagnostic prototype to support the application of a software measurement method, mémoire de fin d'étude." - *Nuremberg: Georg-Simon-Ohm Fachhochschule, 2002.*

[18] **P. Jackson**, “Introduction to Expert Systems”, Third ed. - *New York: Addison-Wesley, 542 p, 1998.*

[19] **Khoshafian, S.; Parsaye, K.; Wong, H.K.T.**; “Intelligent database engines” - *Database Programming and Design, v 3, n 7, July 1990, p 56-60, 62, 64-5, Database: Inspec.*

[20] **Efstathiou, J.; Calinescu, A.; Blackburn, G.**; “A Web-based expert system to assess the complexity of manufacturing organizations” - *Robotics and Computer-Integrated Manufacturing, v 18, n 3-4, June-Aug. 2002, p 305-11, Database: Inspec.*

[21] **Chee, C.W.J.; Power, M.A.**; “Expert systems maintainability” - *Reliability and Maintainability Symposium, 1990. Proceedings., Annual 23-25 Jan. 1990 Page(s):415 – 418 Digital Object Identifier 10.1109/ARMS.1990.67994*

[22] **De la Sen, M.; Minambres, J.J.; Garrido, A.J.; Almansa, A.; Soto, J.C.**; “Basic theoretical results for expert systems. Application to the supervision of adaptation transients in planar robots” - *Artificial Intelligence, v 152, n 2, Feb. 2004, p 173-211, Database: Inspec.*

[23] **Xiaoyi Chi; Ma Haojun; Zhao Zhen; Peng Yinghong**; “Research on hybrid expert system application to blanking technology” - *Journal of Materials Processing Technology, v 116, n 2-3, 24 Oct. 2001, p 95-100, Database: Inspec.*

[24] **Tropper, E.; Beland, S.**; "A new expert system architecture for decision support" - *Artificial Intelligence for Industrial Applications, 1988. IEEE AI '88., Proceedings of the International Workshop on 25-27 May 1988 Page(s):251 – 257, Digital Object Identifier 10.1109/AIIA.1988.13302*

[25] **Avellis, G.; Borzacchini, L.; Cavallo, A.; Cotugno, P.; De Mastro, G.**; "A blackboard architecture for intelligent assistance in software maintenance" - *Computer-Aided Software Engineering, 1993. CASE '93., Proceeding of the Sixth International Workshop on 19-23 July 1993 Page(s):180 – 189, Digital Object Identifier 10.1109/CASE.1993.634819*

[26] **Tsang, C.H.K.; Bloor, C.**; "A medical expert system using object-oriented framework" - *Computer-Based Medical Systems, 1994., Proceedings 1994 IEEE Seventh Symposium on 10-12 June 1994 Page(s):176 – 181, Digital Object Identifier 10.1109/CBMS.1994.316007*

[27] **Demmin, A.T.; Du Zhang**; **"**A Web-based expert system for vehicle registration" - *Information Reuse and Integration, 2003. IRI 2003. IEEE International Conference on 2003 Page(s):420 – 427, Digital Object Identifier 10.1109/IRI.2003.1251446*

[28] **Rong, R.; Brooks, D.; Fu, G.; Eichen, E.**; "Web-based expert system for automated DSL loop qualification" - *Network Operations and Management Symposium, 2000. NOMS 2000. 2000 IEEE/IFIP, 10-14 April 2000 Page(s):201 – 214, Digital Object Identifier 10.1109/NOMS.2000.830385*

[29] **Lababidi, H.M.S.; Baker, C.G.J.**; "Web-based expert system for food dryer selection" - *Computers & Chemical Engineering, v 27, n 7, 15 July 2003, p 997-1009, Database: Inspec.*

[30] **Daoliang Li; Zetian Fua; Yanqing Duan**; "Fish-Expert: a Web-based expert system for fish disease diagnosis" - *Expert Systems with Applications, v 23, n 3, Oct. 2002, p 311-20, Database: Inspec.*

[31] **Hongmei Yan; Jiang, Y.; Zheng, J.; Fuc, B.; Shouzhong Xiao; Chenglin Peng**; "The Internet-based knowledge acquisition and management method to construct large-scale distributed medical expert systems" - *Computer Methods and Programs in Biomedicine, v 74, n 1, April 2004, p 1-10, Database: Inspec.*

[32] **Thomson, A.J.; Willoughby, I.**; "A Web-based expert system for advising on herbicide use in Great Britain" - *Computers and Electronics in Agriculture, v 42, n 1, Jan. 2004, p 43-9, Database: Inspec.*

[33] **Riedesel, J.D.**; "An object oriented model for expert system shell design" - *Computers and Communications, 1990. Conference Proceedings., Ninth Annual International Phoenix Conference on 21-23 March 1990 Page(s):699 – 705, Digital Object Identifier 10.1109/PCCC.1990.101688*

[34] **Koseki, Y.; Tanaka, M.; Maeda, Y.; Koike, Y.**; "Visual programming environment for hybrid expert systems" - *Expert Systems with Applications, v 10, n 3-4, 1996, p 481-6, Database: Inspec.*

[35] **Guarino, N.**; "Formal ontology and information systems" - *Formal Ontology in Information Systems. Proceedings of the First International Conference (FOIS'98), 1998, p 3-15, Database: Inspec.*

[36] **B. Biébow, S. Szulman**, "TERMINAE: a method and a tool to build a domain ontology" – *France: Université de Paris-Nord, Laboratoire d'Informatique de Paris-Nord(LIPN), Proceedings of the 11th European Knowledge Acquisition, 1999.*

[37] **M. A. Hahn, R. N. Palmer, M. S. Merrill, A. B. Lukas**, "Expert System for Prioritizing the Inspection of Sewers: Knowledge Base Formulation and Evaluation" - *Seattle : University of Washington, JOURNAL OF WATER RESOURCES PLANNING AND MANAGEMENT, MARCH/APRIL 2002.*

[38] **Desharnais, J.-M.; Abran, A.; Mayers, A.; Buglione, L.; Bevo, V.**; "Knowledge modeling for the design of a KBS in the functional size measurement domain" - *Knowledge-Based Intelligent Information Engineering Systems and Allied Technologies. KES 2002, 2002, pt. 1, p 26-31 vol.1*

[39] **van Heijst, G.; Schreiber, A.Th.; Wielinga, B.J.**; "Using explicit ontologies in KBS development" - *International Journal of Human-Computer Studies, v 46, n 2-3, Feb.-March 1997, p 183-292, Database: Inspec.*

[40] **Bourque, P.; Dupuis, R.; Abran, A.; Moore, J.W.; Tripp, L.**; "The guide to the Software Engineering Body of Knowledge" - *Software, IEEE Volume 16, Issue 6, Nov.-Dec. 1999 Page(s):35 – 44, Digital Object Identifier 10.1109/52.805471*

[41] **J. B. Dreger** , "Function Point Analysis" - *Prentice Hall, 1989.*

[42] **C.R. Symons**, "Software sizing and estimating: Mk II FPA (function point analysis)." - *Chichester, West Sussex, England; New York: Wiley, 1991.*

[43] **C. Jones**, "A Short History of Function Points and Feature Points." – USA: *Software Productivity Research Inc., 1987.*

[44] **S.A. Whitmire**, "3D Function Points: Scientific and Real-Time Extensions to Function Points." - *Proc. Pacific Northwest Software Quality Conf., 1992.*

[45] **Norme ISO/IEC 19761**, *2002* / **Norme ISO/IEC 14143,** *2000.*

[46] **Aamodt, A.; Plaza, E.**; "Case-based reasoning: foundational issues, methodological variations, and system approaches" - *AI Communications, v 7, n 1, March 1994, p 39-59, Database: Inspec.*

[47] **J. L. Kolodner,** "Case-Based Reasoning." - *San Francisco: Morgan Kaufmann, 1993.*

[48] **Boehm, B.; Basili, V.R.**; "Top 10 list [software development]" – *Computer Volume 34, Issue 1, Jan. 2001 Page(s):135 – 137, Digital Object Identifier 10.1109/2.962984*

[49] **April, A.; Abran, A.; Dumke, R.R.**; "SM^{CMM} model to evaluate and improve the quality of the software maintenance process" - *Eighth European Conference on Software Maintenance and Reengineering, 2004, p 243-8, Database: Inspec.*

[50] **F. Dumont,** "Identification des objets dans un code procédural basée sur la décomposition de graphes" – *Sherbrooke, Québec : Université de Sherbrooke, 1997.*

[51] **J.-M. Desharnais**, "LOG 510 : Contrôle de la qualité et métriques" – *cours à l'École de Technologie Supérieure, Montréal, session d'hiver 2004.*

[52] **C. Cavaness**, "Programming Jakarta Struts" – *O'Reilly, 2002.*

[53] **T. Husted, C. Dumoulin, G. Franciscus, D. Winterfeldt**, "Struts in Action" – *Greenwich, CT: Manning, 2003.*

[54] **M. Robinson, E. Finkelstein**, "Jakarta Struts for Dummies" - *New York: Wiley, 2004.*

[55] **A. April, J.-M. Desharnais, R. Dumke**, "Software Maintenance knowledge-based system (SMxpert) - A formalisation of the software maintenance ontology" – 12 p

[56] **Kitchenham, B.A.; Travassos, G.H.; von Mayrhauser, A.; Niessink, F.; Schneidewind, N.F.; Singer, J.; Takada, S.; Vehvilainen, R.; Hongji Yang**; "Towards an ontology of software maintenance" - *Journal of Software Maintenance: Research and Practice, v 11, n 6, Nov.-Dec. 1999, p 365-89, Database: Inspec.*

[57] **Ruiz, F.; Vizcaino, A.; Piattini, M.; Garcia, F.**; "An ontology for the management of software maintenance projects" - *International Journal of Software Engineering and Knowledge Engineering, v 14, n 3, June 2004, p 323-49, Database: Inspec.*

[58] **Dekleva, S.**; "Delphi study of software maintenance problems" - *Software Maintenance, 1992. Proceerdings., Conference on 9-12 Nov. 1992 Page(s):10 – 17, Digital Object Identifier 10.1109/ICSM.1992.242564*

[59] **A. April, A. Abran, R. Dumke**, "Assessment of Software Maintenance Capability: A model and its Design Process" - *presented at the IASTED 2004 Conference on Software Engineering, Innsbruck (Austria), 2004.*

[60] **SEI**, "Capability Maturity Model Integration for Software Engineering (CMMi), Version 1.1, CMU/SEI-2002-TR-028, ESC-TR-2002-028," - *SEI, Ed.: Carnegie Mellon University, 2002.*

[61] **J.-M. Desharnais, A. Abran, A. Mayers, J. Vilz, F. Gruselin**, "Verification and validation of a knowledge based system" - *KI Journal, Special Issue on Software Engineering for Knowledge-based Systems, Germany, Vol. 3, 2004.*

[62] **Tahvildari, L.; Kontogiannis, K.; Mylopoulos, J.**; "Requirements-driven software re-engineering framework" - *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on 2-5 Oct. 2001 Page(s):71 – 80, Digital Object Identifier 10.1109/WCRE.2001.957811*

[63] **R Arnold, R.S.**; "Software restructuring" - *Proceedings of the IEEE Volume 77, Issue 4, April 1989 Page(s):607 – 617, Digital Object Identifier 10.1109/5.24146*

[64] **Offutt, J.**; "Quality attributes of Web software applications" - *Software, IEEE Volume 19, Issue 2, March-April 2002 Page(s):25 – 32, Digital Object Identifier 10.1109/52.991329*

[65] **P. Pohjolainen**, "Software Testing Tools" – *University of Kuopio, Departement of Computer Science and Applied Mathematics, March 2002.*

[66] **P. Bellot**, "Techniques de test, partie 1 : Introduction et outils" – *cours de l'université d'Avignon, IUP GMI, 2001.*

[67] **N. Habra**, "Méthodologie de développement de logiciels Matières Approfondies, Chapitre 5: Principales méthodologies Agile : Rational Unified Process" - *Namur : Facultés Universitaires Notre-Dame de la Paix, cours de l'institut d'informatique, 2003-2004.*

[68] **A. Abran, J.-M. Desharnais, S. Oligny, D. St-Pierre, C. Symons**, "Measurement Manual 2.2" – *UQÀM May 2002.*

[69] **P. Morris, J.-M. Desharnais**, "Measuring ALL the Software not just what the Business Uses" – *document présenté à International Function Point Users Group, 1998, 18p.*

[70] **A.T. Schreiber, J. M. Akkermans, A. A. Anjewierden, R. de Hoog, N. R. Shadbolt, W. Van de Velde, B. J. Wielinga**, "Knowledge Engineering and Management: The CommonKADS Methodology." – *Boston: MIT Press, 2000, 510 p.*

[71] **G. Born**, "KADS: A Methodology for Developing Large AI Systems" – *United Kingdom: SD-Scicon,1990, 6 p.*

[72] **C. L. McClure** ; "The Three Rs of Software Automation" – *Prentice-Hall, Englewood Cliffs, NJ, 1992.*

[73] **M. L. Nelson** ; "A Survey of Reverse Engineering and Program Comprehension" – *Northfolk : Old Dominion University, Technical Report, Software Engineering Survey, April 19, 1996.*

Liens Chapitre 4

MVC

http://takotech.com/exercise7C.html

Web Development in J2EE Using the MVC Design Pattern

http://www.devarticles.com/c/a/Java/Web-Development-in-J2EE-Using-the-MVC-Design-Pattern/

JSP:

http://java.sun.com/products/jsp/tutorial/TagLibraries3.html#63159

Java

http://perso.wanadoo.fr/jm.doudoux/java/tutorial/chap001.htm

http://www.dailly.info/java/java.php

http://www.commentcamarche.net/langages/langages.php3

http://java.sun.com/j2se/overview.html

Struts

http://www.sitepoint.com/article/complete-mvc-puzzle-struts

http://struts.apache.org/api/

Liens Chapitre 5

Le générateur de tests d'acceptation FAUST

http://www.cetic.be/internal219.html

Cours sur les tests

https://cours.ele.etsmtl.ca/academique/mgl/mgl830/lectures/testsUsabilite.ppt (ETS, Montréal)

http://www.sco.univ-avignon.fr/forma_elt.html?code=U06-1218 Cours de U06-1218 GENIE LOGICIEL AVANCE, Université d'Avignon

http://www.smart-doc.org/modules.php?op=modload&name=Sections&file=index&req=viewarticle&artid=17&page=1 Comment tester un logiciel en tant que développeur

Vérification et validation

http://www-lor.int-evry.fr/~raffy/Poly/v&v.htm

Processus de test

http://www.infeig.unige.ch/support/se/lect/prg/tst/node8.html

http://www.artemis.jussieu.fr/wwwos2/html/dess/memoire/promo95/muller/GESTPRO.HTM

Les cycles de développement de logiciels

http://www.laltruiste.com/annexe/developpement_logiciel.html l

Tests unitaires, d'intégration et de validation

http://www.surlog.com/fr/methodes/tests.htm

IBM Rational

http://www.essi.fr/~hugues/GL/rational/robot/robot.htm Test Fonctionnel avec Rational Robot

Autres liens

[Web1] http://www.sei.cmu.edu/architecture/definitions.html

[Web2] http://www.softwarearchitectures.com/one/Designing+Architecture/78.aspx

[Web3] http://herzberg.ca.sandia.gov/jess/

[Web4] http://www.apache.org/licenses/

Annexe 1 : Architecture logique de COSMICXpert Web Edition

Cette annexe présente l'architecture à haut niveau de COSMICXpert. Cette dernière a été esquissée sur base de la documentation du logiciel (exigences, cas d'utilisation, manuel de l'utilisateur…) présente dans les travaux passés s'y rapportant. L'annexe met également en exergue la similitude entre les niveaux d'abstraction de COSMICXpert et de SMXpert.

La figure 1 présente l'architecture globale de COSMICXpert. Comme pour son successeur (cf. Chapitre 3), deux types d'acteurs sont présents : les utilisateurs et le système à base de connaissances (SBC).

Chaque type d'utilisateur (administrateur, expert et utilisateur) dispose d'une interface propre chargée de la correction des données qu'elle transmet. Les composants IHM et Validation Syntaxique remplissent ce rôle.

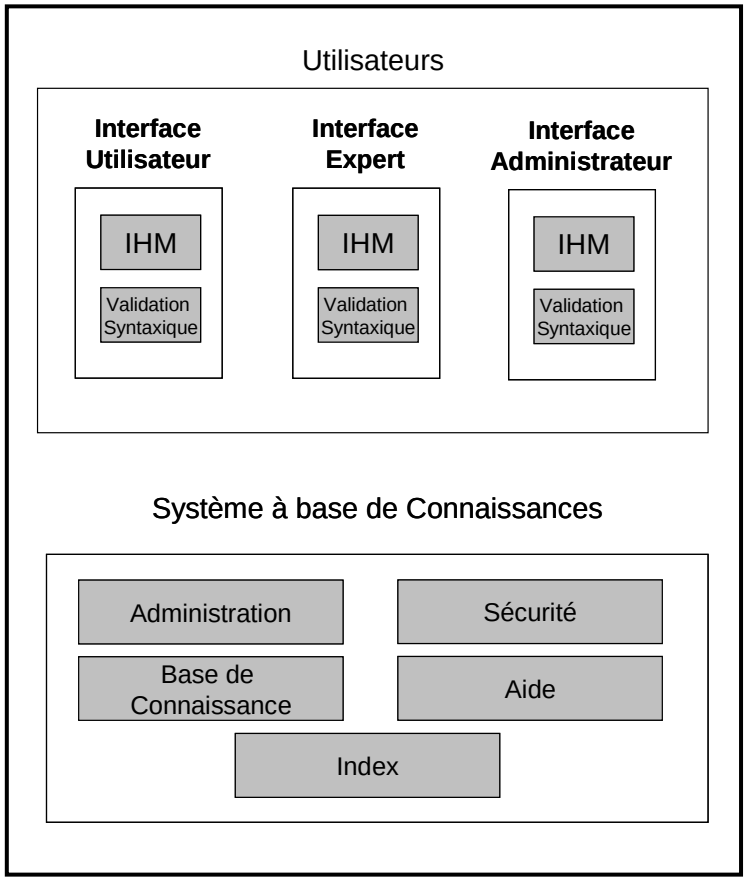


Figure 1 : Schéma de l'architecture logique de COSMICXpert.

Le **SBC** comporte cinq composants, illustrés à la figure 2. Il est à noter qu'aucun composant lié à la configuration n'apparaît ici. Ce composant constitue la principale différence, à un haut niveau d'abstraction, entre les deux logiciels.

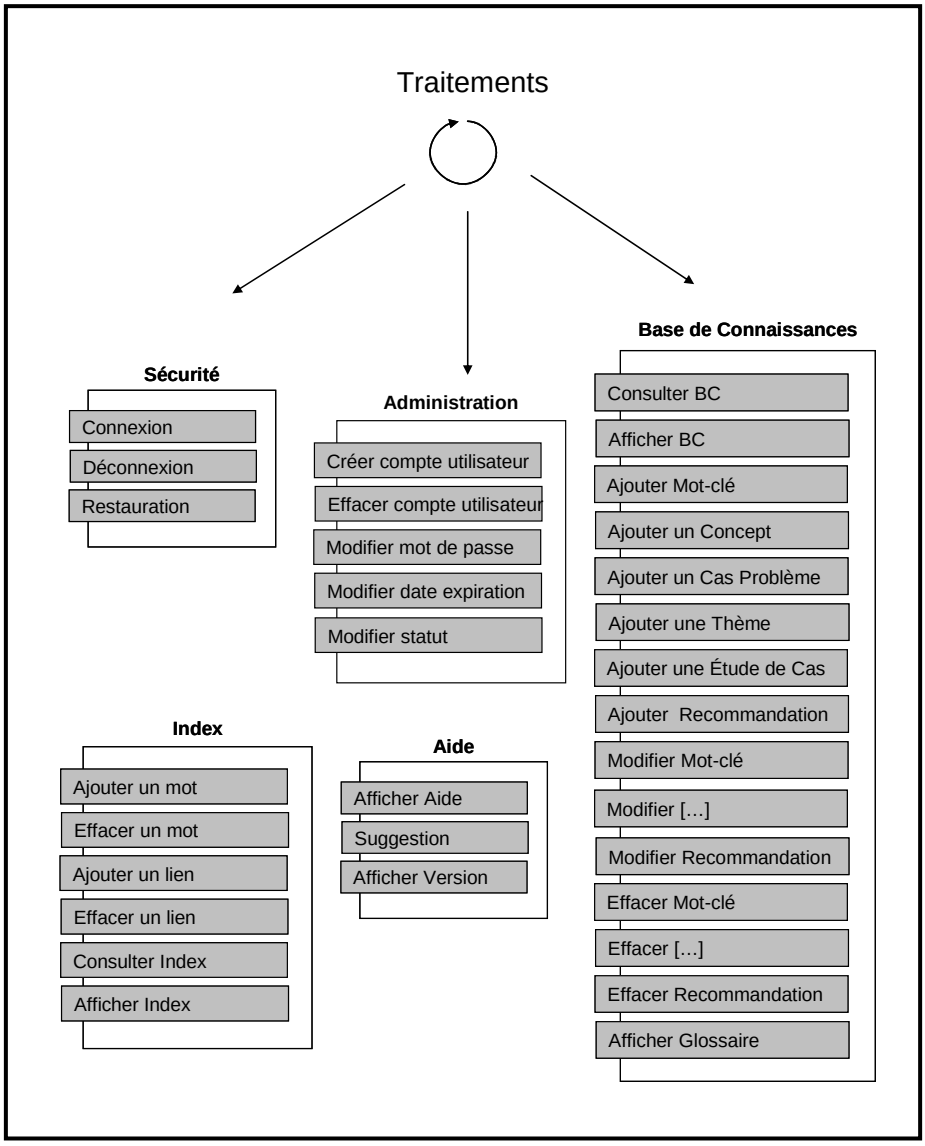


Figure 2 : Schéma des composants du SBC et leurs tâches

- **Le module Sécurité :** regroupe toutes les fonctionnalités garantissant une certaine sûreté du système. On y trouve la gestion des sessions, ainsi que des connexions et déconnexions à ces dernières. Une fonctionnalité de restauration de la base de connaissances est aussi intégrée à ce module, afin de garantir une dimension de récupération de désastres.

- **Le module Aide :** regroupe les fonctionnalités propres à guider l'utilisateur dans sa compréhension du logiciel.
- **Le module Administration :** regroupe toutes les fonctionnalités liées à la manipulation des comptes utilisateur. Celles-ci couvrent notamment l'ajout et la suppression d'un compte. De plus, le module regroupe les fonctionnalités de *modification* desdits comptes.
- **Le module Index :** regroupe les fonctionnalités propres à l'ajout et la suppression d'un mot de l'index. Il renferme également les opérations de lien entre lesdits mots et les mots-clés vers lesquels ils sont sensés orienter l'utilisateur.
- **Le module Base de Connaissances :** le plus important et le plus conséquents de modules du **SBC.** Il regroupe en effet les fonctionnalités liées à la manipulation des concepts de l'ontologie de COSMICXpert. On retrouve donc dans ce module les fonctionnalités d'ajout, de suppression et de modification des concepts topologiques, mot-clés, cas problèmes, thèmes et recommandations. Ce module renferme également les opérations de consultation de la base de connaissances, ainsi que les fonctionnalités d'affichage de celle-ci.

Annexe 2 : Cas d'utilisation de SMXpert Web Edition.

Cette annexe présente les cas d'utilisation de **SMXpert Web Edition**. Suite au diagramme des cas, illustré à la figure 1, chacun de ceux-ci est détaillé.

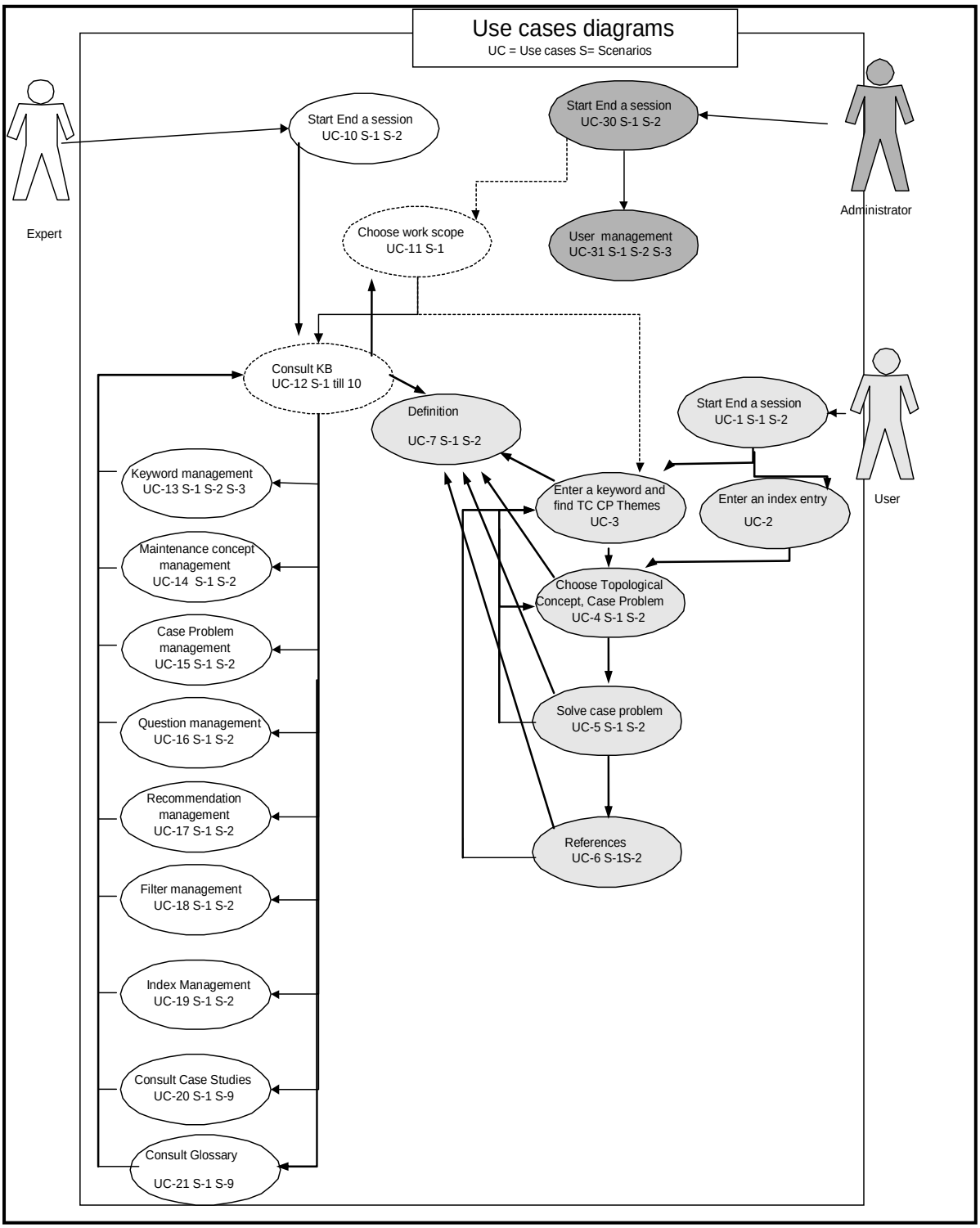


Figure 1 : Diagramme des cas d'utilisation.

Cas d'utilisations de l'interface « utilisateur »

Use Case: Start a session **Scénario 1 : Start a session**
ID: UC 1.1
Préconditions : 1. L'acteur a démarré le système, i.e. a ouvert la page Web de SMXpert. 2. L'acteur est enregistré auprès du système avec un statut « utilisateur ».
Scénario Normal : 1. Le cas d'utilisation commence quand l'acteur ouvre la page Web de SMXpert. 2. L'acteur introduit un nom d'utilisateur et un mot de passe valides 3. Le système vérifie la date d'expiration de l'utilisateur. 4. Le système affiche l'interface correspondante au statut de l'acteur
Scénarios Alternatifs : InvalidUsername ou InvalidPassword DateExpires
Postconditions : 1. L'acteur a accès à la partie « utilisateur » du système.

Use Case: Start a session ***Scénario Alternatif : InvalidUsername or Invalid Password***
ID: UC 1.2
Préconditions : 1. L'acteur a fourni un nom d'utilisateur et/ou un mot de passe non-valide(s).
Scénario Alternatif : 1. Le cas d'utilisation commence à l'étape 2 du cas *Start a session* un nom d'utilisateur et/ou un mot de passé invalide. 2. Le système informe l'acteur qu'il a introduit de mauvaises valeurs et lui demande d'introduire son nom d'utilisateur et son mot de passe à nouveau.
Postconditions : -

Use Case: Start a session ***Scénario Alternatif : DateExpires***
ID: UC 1.3
Préconditions : 1. L'acteur a fourni un nom d'utilisateur et le mot de passe d'un compte dont la date de validité est dépassée.
Scénario Alternatif : 1. Le cas d'utilisation commence à l'étape 3 du cas *Start a Session* quand le système vérifie la date d'expiration de l'utilisateur, et que cette date est dépassée. 2. Le système informe l'acteur que sa date d'utilisation a expiré et qu'il n'est plus autorisé à utiliser le système. 3. Le système lui permet de réintroduire un nouveau nom d'utilisateur et mot de passe.
Postconditions : -

Use Case: Enter an index entry
ID: UC 2.1
Préconditions : 1. L'acteur est connecté à l'interface utilisateur.
Flux des événements : 1. Le cas d'utilisation commence quand l'acteur clique sur « Search by Index » dans l'interface du système. 2. Le système demande à l'acteur d'introduire une entrée d'index. 3. L'acteur introduit ce mot et appuies sur « Search ». 4. Si le mot ne correspond pas à une entrée d'index existante → UC 2.2 5. Sinon 5.1. Le système propose, à l'acteur, une liste de mots-clefs liés à cette entrée d'index. 5.2. L'acteur en choisi un. 5.3. Le système revient à l'interface principale et affiche les concepts de maintenance liés à ce mot-clef.
Scénarios Alternatifs : Index word doesn't exist
Postconditions : 1. Les concepts de maintenance, en rapport avec la recherche, sont affichés.

Use Case: Enter an index entry
Scénario Alternatif : Index word doesn't exist
ID: UC 2.2
Préconditions : 1. L'acteur a fourni un nom d'utilisateur et le mot de passe d'un compte dont la date de validité est dépassée.
Scénario Alternatif : 1. Le cas d'utilisation commence à l'étape 4 du cas *Enter an index entry* quand le système vérifie la présence du mot dans les entrées de l'index. 2. Le système informe l'acteur que le mot n'existe pas dans l'index et lui propose de revenir à l'étape précédente. 3. L'acteur appuie sur le bouton « Back » → UC 2.1
Postconditions : -

Use Case: Choose a Keyword
ID: UC 3
Préconditions : • UC 1.1 • Existence de liens entre les mots-clefs et les concepts de maintenance.
Flux des événements : 1. Le cas d'utilisation commence quand l'acteur sélectionne un mot-clef dans la liste déroulante qui reprend l'entièreté de ceux-ci et appuie sur le bouton « Search ». 2. Si le mot-clef est lié à des concepts de maintenance : 2.1. Le système affiche ceux-ci. 2.2. Le cas d'utilisation se termine. 3. Sinon : 3.1. Le système n'affiche aucun concept de maintenance. 3.2. Le cas d'utilisation se termine.
Postconditions : 1. Les concepts de maintenance, en rapport avec la recherche, sont affichés.

Use Case: Choose a maintenance concept, case problem
ID: UC 4
Préconditions : 1. UC 3 ou UC 2.1 2. Existence de liens entre les concepts de maintenance, les cas problèmes et les questions.
Flux des événements : 1. Le cas d'utilisation commence quand l'acteur sélectionne un concept de maintenance parmi ceux affichés et appuie sur le bouton « Search ». 2. Le système affiche la liste des cas problèmes liés au concept de maintenance sélectionné. 3. L'acteur sélectionne un cas problème parmi ceux affichés et appuie sur le bouton « Search ». 4. S'il existe des questions, liées au concept de maintenance et au cas problème choisis, qui sont du niveau de l'acteur : 4.1. Le système affiche les questions, liées au concept de maintenance et au cas problème choisis, en adéquation avec le niveau de l'acteur. 4.2. Le cas d'utilisation se termine. 5. Sinon : 5.1. Le système n'affiche aucune question. 5.2. Le cas d'utilisation se termine.
Postconditions : 1. Les questions, en rapport avec la recherche, sont affichés.

Use Case: Solve questions
ID: UC 5
Préconditions : 1. UC 4 2. Existence de liens entre le concept de maintenance, les questions, le cas problème et les recommandations.
Flux des événements : 1. Le cas d'utilisation commence quand l'acteur fixe la valeur du fait pour chaque question (deux valeurs : « Yes » ou « No ») et il appuie sur « Ok ». 2. Le système affiche une recommandation. Elle est en rapport direct avec la première question à laquelle l'acteur a choisi « No » ou au concept de maintenance si tous les faits possédaient la valeur « Yes ».
Postconditions : 1. La recommandation en rapport au choix des faits est affichée.

Use Case: Redirection
ID: UC 6
Préconditions : 1. UC 5
Flux des événements : 1. Si la recommandation est liée à un mot-clef qui permet une nouvelle recherche : 1.1. Le système propose à l'acteur d'effectuer la recherche des concepts de maintenance directement, liés au nouveau mot-clef,en appuyant sur le bouton « Execute the search linked to the recommendation ». 1.2. L'acteur appuie sur le bouton « Execute the search linked to the recommendation ». 1.3. Le système efface l'entièreté de la précédente recherche et affiche les concepts de maintenance liés au mot-clef cité dans la recommandation. 2. Sinon : 2.1. Le cas d'utilisation se termine.
Postconditions : 1. Les concepts de maintenance, en rapport avec la recherche, sont affichés.

Use Case: Definition
ID: UC 7
Préconditions : 1. Existence de la définition du mot-clef/concept de maintenance/cas problème/question/recommandation et du lien hypertexte dans l'interface. 2. L'acteur est connecté à l'interface utilisateur.
Flux des événements : • Le cas d'utilisation commence quand l'acteur clique sur un lien hypertexte qui mène à la définition d'un(e) mot-clef/concept de maintenance/cas problème/question/recommandation ou sur un bouton noté « Definition ». • Le système affiche la définition demandée par l'acteur dans une nouvelle fenêtre.
Postconditions : 1. La définition est affichée.

Cas d'utilisations de l'interface « expert »

Use Case: Start a session **Scénario 1 : Start a session**
ID: UC 10.1
Préconditions : 1. L'acteur a démarré le système, i.e. a ouvert la page Web de SMXpert. 2. L'acteur est enregistré auprès du système avec un statut « expert ».
Scénario Normal : 1. Le cas d'utilisation commence quand l'acteur ouvre la page Web de SMXpert. 2. L'acteur introduit un nom d'utilisateur et un mot de passe valides 3. Le système vérifie la date d'expiration de l'utilisateur. 4. Le système affiche l'interface correspondante au statut de l'acteur
Scénarios Alternatifs : InvalidUsername ou InvalidPassword DateExpires
Postconditions : 1. L'acteur a accès à la partie « expert » du système.

Use Case: Start a session ***Scénario 1 : Start a session*** ***Scénario Alternatif 1: InvalidUsername or Invalid Password***
ID: UC 10.1.1
Préconditions : 1. L'acteur a fourni un nom d'utilisateur et/ou un mot de passe non-valide(s).
Scénario Alternatif : 1. Le cas d'utilisation commence à l'étape 2 du cas *Start a session* un nom d'utilisateur et/ou un mot de passé invalide. 2. Le système informe l'acteur qu'il a introduit de mauvaises valeurs et lui demande d'introduire son nom d'utilisateur et son mot de passe à nouveau.
Postconditions : -

Use Case: Start a session ***Scénario 1 : Start a session*** ***Scénario Alternatif 2: DateExpires***
ID: UC 10.1.2
Préconditions : 1. L'acteur a fourni un nom d'utilisateur et le mot de passe d'un compte dont la date de validité est dépassée.
Scénario Alternatif : 1. Le cas d'utilisation commence à l'étape 3 du cas *Start a Session* quand le système vérifie la date d'expiration de l'utilisateur, et que cette date est dépassée. 2. Le système informe l'acteur que sa date d'utilisation a expiré et qu'il n'est plus autorisé à utiliser le système. 3. Le système lui permet de réintroduire un nouveau nom d'utilisateur et mot de passe.
Postconditions : -

Use Case: Choose Work Scope **Scénario 1 : User scope**
ID: UC 11.1
Préconditions : 1.L'acteur a accès au système en tant qu'expert ou administrateur. 2.L'acteur est actuellement en interaction avec une interface réservée à l'administrateur ou à l'expert.
Flux des événements : 1. Le cas d'utilisation commence quand l'expert ou l'administrateur choisit « User Interface ». 2. Le système ouvre une nouvelle fenêtre utilisateur.
Postconditions : 1. L'expert/administrateur peut utiliser toutes les fonctionnalités accessibles à partir de l'interface de l'utilisateur.

Use Case: Choose Work Scope **Scénario 2 : Expert scope**
ID: UC 11.2
Préconditions : 1. L’acteur a accès au système en tant qu'administrateur.
Flux des événements : 1. Le cas d’utilisation commence quand l’administrateur choisit “Expert Interface” 2. Le système affiche l'interface de l'expert.
Postconditions : 1. L’administrateur peut utiliser toutes les fonctionnalités accessibles à partir de l’interface de l'expert.

Use Case: Choose Work Scope **Scénario 3 : Administrator scope**
ID: UC 11.3
Préconditions : 1. L’acteur a accès au système en tant qu'administrateur. 2. L'administrateur se trouve actuellement dans l'interface expert.
Flux des événements : 1. Le cas d’utilisation commence quand l’administrateur choisit “Administrator Interface” 2. Le système affiche l'interface de l'expert.
Postconditions : 1. L’administrateur peut utiliser toutes les fonctionnalités accessibles à partir de l’interface de l'expert.

Use Case: Consult KB **Scénario 1 : Consult KB**
ID: UC 12.1
Préconditions : 1. L'acteur a accès à l'interface expert.
Flux des événements : 1. Le cas d'utilisation commence lorsque l'acteur se connecte à l'interface expert ou lorsque qu'il clique sur le bouton « Keywords – Maintenance Concepts – Case Problems Tree » ou « Maintenance Concepts – Questions – Facts Tree » ou « Case Problems – Recommandations Tree ». 2. Le système affiche la base de connaissances sous la forme par défaut « Keywords – Maintenance Concepts – Case Problems Tree » ou alors sous la forme précédente demandée par l'acteur.
Postconditions : 1. L'expert peut consulter la base de connaissances sous la forme qui lui convient.

Use Case: Keyword Management **Scénario 1 : Add a keyword**
ID: UC 13.1
Préconditions : 1. L'acteur a accès à l'interface expert.
Flux des événements : 1. Le cas d'utilisation commence quand l'acteur sélectionne "Add a Keyword" dans l'interface du système. 2. Le système demande à l'acteur d'introduire le nom du nouveau mot-clé, sa description, les concepts de maintenances auxquels il sera relié, ainsi que le pourcentage correspondant à la magnitude de ce lien. 3. L'acteur introduit ces valeurs. 4. Si le mot-clé existe déjà 4.1. Le système informe l'expert que le mot-clé existe déjà. 4.2. Le cas d'utilisation se termine. 5. Sinon 5.1. Le système ajoute le nouveau mot-clé.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Keyword Management **Scénario 2 : Modify a keyword**
ID: UC 13.2
Préconditions : 1. L’acteur a accès à l'interface expert.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur choisit “modify” quand un mot-clé est sélectionné. 2. L’acteur modifie le nom et/ou la description et/ou le pourcentage de la relation avec les concepts de maintenance choisis. 3. Le système appliqué les changements au mot-clé sélectionné.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Keyword Management **Scénario 3 : Delete a keyword**
ID: UC 13.3
Préconditions : 1. L’acteur a accès à l'interface expert. 2. Le mot-clef à supprimer est sélectionné.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur choisit “delete” quand un mot-clef est sélectionné. 2. Le système demande à l’acteur une confirmation de suppression du mot-clef. 3. Si l’acteur confirme la suppression 3.1. Si tous les concepts de maintenance auxquels sont rattachés ce mot-clef sont lies avec au moins un autre mot-clé 3.1.1. Le système supprime le mot-clef. 3.2. Sinon 3.2.1. Le système informe l’acteur qu’il n’est pas autorisé à supprimer ce mot-clef. 3.2.2. Le cas d’utilisation se termine. 4. Sinon 4.1. Le cas d’utilisation se termine.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Maintenance Concept Management **Scénario 1 : Add a maintenance concept**
ID: UC 14.1
Préconditions : 1. L'acteur a accès à l'interface expert.
Flux des événements : 1. Le cas d'utilisation commence quand l'expert choisit "Add a Maintenance Concept". 2. Le système demande à l'acteur d'introduire le nom du nouveau concept de maintenance, le type de concept, les mot-clefs (et/ou noeud) associés. 3. Le système demande à l'acteur d'introduire une explication en rapport au pourquoi du concept de maintenance. 4. Le système demande à l'acteur d'introduire d'autres explications et de nouveaux cas problèmes (*Add a case problem*) et leurs recommandations correspondantes (*Add a recommendation*). 5. L'expert introduit ces valeurs. 6. Si le concept de maintenance existe déjà 6.1. Le système informe l'acteur que le concept de maintenance existe déjà. 6.2. Le cas d'utilisation se termine. 7. Le système demande à l'acteur d'introduire de nouvelles questions associées au concept (*Add a question*). 8. Le système demande à l'acteur de choisir, pour chaque cas problème, la recommandation qui sera affichée si toutes les réponses aux questions sont « Yes ». 9. Le système ajoute le nouveau concept de maintenance et tous les cas problèmes associés.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Maintenance Concept Management **Scénario 2 : Modify a maintenance concept**
ID: UC 14.2
Préconditions : 1. L'acteur a accès à l'interface expert. 2. Le concept de maintenance à modifier est sélectionné.
Flux des événements : 1. Le cas d'utilisation commence quand l'acteur choisit "modify" quand un concept de maintenance est sélectionné. 2. L'acteur modifie le nom et/ou l'explication et/ou les relations avec les mots-clés (et/ou noeuds). 3. Le système applique les changements au concept de maintenance sélectionné.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Case Problem Management **Scénario 1 : Add a case problem**
ID: UC 15.1
Préconditions : 1. L’acteur a accès à l'interface expert. 2. Le concept de maintenance, dans lequel on veut introduire le cas problème, est sélectionné.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur choisit “add” quand un concept de maintenance est sélectionné. 2. Le système demande à l’acteur d’introduire le nom du nouveau cas problème, sa description, l'étude de cas dans laquelle il se trouve et les mot-clefs/noeuds associés. 3. L’acteur introduit ces valeurs. 4. Si le cas problème existe déjà 4.1. Le système informe l’expert que le cas problème existe déjà. 4.2. Le cas d’utilisation se termine. 5. Le système demande à l'acteur de sélectionner une série de recommandations de base et propose ensuite de les modifier, de les supprimer et d'en ajouter. 6. Le système ajoute le nouveau cas problème.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Case Problem Management **Scénario 2 : Modify a case problem**
ID: UC 15.2
Préconditions : 1. L’acteur a accès à l'interface expert. 2. Le cas problème à modifier est sélectionné.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur choisit “modify” quand un cas problème est sélectionné. 2. L’acteur modifie le nom et/ou la description et/ou l'étude de cas dans laquelle il se trouve et/ou les relations avec le mot-clefs/noeuds. 3. Le système applique les changements au cas problème sélectionné.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Case Problem Management **Scénario 3 : Delete a case problem**
ID: UC 15.3
Préconditions : 1. L’acteur a accès à l'interface expert. 2. Le cas problème à supprimer est sélectionné.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur choisit “delete” quand un cas problème est sélectionné. 2. Le système demande une confirmation à l’acteur et l’informe que s’il poursuit les recommandations seront aussi supprimées. 3. Si l’acteur confirme la suppression 3.1. Si ce cas problème est le dernier cas associé à un concept de maintenance 3.1.1. Le système informe l’acteur que le cas problème ne peut pas être supprimé. 3.1.2. Le cas d’utilisation se termine. 3.2. Sinon 3.2.1. Le système supprime le cas problème et toutes les recommandations associées. 4. Sinon 4.1. Le cas d’utilisation se termine.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Question Management **Scénario 1 : Add a question**
ID: UC 16.1
Préconditions : 1. L’acteur a accès à l'interface expert. 2. Le concept de maintenance, dans lequel on veut ajouter la question, est sélectionné.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur choisit “add” quand un concept de maintenance est sélectionné. 2. Le système demande à l’acteur de donner une priorité à la question. 3. Le système demande à l’acteur d’introduire le nom de la nouvelle question, sa description et le pourcentage de l'association avec le concept de maintenance sélectionné. 4. L’acteur introduit ces valeurs. 5. Le système demande à l’acteur de choisir, pour chaque cas problème, la recommandation qui lui sera liée. 6. Si cette question existe déjà pour ce concept de maintenance et le filtre courant 6.1. Le système informe l’acteur que la question existe déjà. 6.2. Le cas d’utilisation se termine. 7. Sinon 7.1. Le système ajoute la nouvelle question.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Question Management **Scénario 2 : Modify a question**
ID: UC 16.2
Préconditions : 1. L’acteur a accès à l'interface expert. 2. La question à modifier est sélectionnée.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur choisit “modify” quand une question est sélectionnée. 2. Le système propose à l’acteur la possibilité de changer la priorité de la question. 3. L’acteur change le nom et/ou la description et/ou le pourcentage de la relation avec le concept de maintenance choisi. 4. Le système applique les changements à la question sélectionnée.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Question Management **Scénario 3 : Delete a question**
ID: UC 16.3
Préconditions : 1. L'acteur a accès à l'interface expert. 2. La question à supprimer est sélectionnée.
Flux des événements : 1. Le cas d'utilisation commence quand l'acteur choisit "delete" quand une question est sélectionnée. 2. Le système demande à l'acteur de confirmer la suppression. 3. Si l'acteur confirme la suppression de la question 3.1. Si la question est la dernière associée à un concept de maintenance. 3.1.1. Le système informe l'acteur que la question ne peut pas être supprimée. 3.1.2. Le cas d'utilisation se termine. 3.2. Sinon 3.2.1. Le système supprime la question. 4. Sinon 4.1. Le cas d'utilisation se termine.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Recommandation Management **Scénario 1 : Add a recommandation**
ID: UC 17.1
Préconditions : 1. L'acteur a accès à l'interface expert. 2. Le cas problème, dans lequel on veut ajouter la recommandation, est sélectionné.
Flux des événements : 1. Le cas d'utilisation commence quand l'acteur choisit "add" quand un cas problème est sélectionné. 2. Le système demande à l'acteur d'introduire le nom de la nouvelle recommandation, sa description ainsi que le mot-clé vers lequel la recommandation doit, le cas échéant, rediriger l'utilisateur. 3. L'acteur introduit ces valeurs. 4. Si la recommandation existe déjà 4.1. Le système informe l'acteur que la recommandation existe déjà. 4.2. Le cas d'utilisation se termine. 5. Sinon 5.1. Le système ajoute la nouvelle recommandation.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Recommandation Management **Scénario 2 : Modify a recommandation**
ID: UC 17.2
Préconditions : 1. L'acteur a accès à l'interface expert. 2. La recommandation à modifier est sélectionnée.
Flux des événements : 1. Le cas d'utilisation commence quand l'acteur choisit "modify" quand une recommandation est sélectionnée. 2. L'acteur modifie le nom et/ou la description, ainsi que le mot-clé vers lequel la recommandation redirige l'utilisateur. 3. Le système applique les changements à la recommandation.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Recommandation Management **Scénario 3 : Delete a recommandation**
ID: UC 17.3
Préconditions : 1. L'acteur a accès à l'interface expert. 2. La recommandation à modifier est sélectionné.
Flux des événements : 1. Le cas d'utilisation commence quand l'acteur choisit "delete" quand une recommandation est sélectionnée. 2. Le système demande à l'acteur de confirmer la suppression. 3. Si l'acteur confirme la suppression de la recommandation 3.1. Si le groupe de recommandations associées à ce cas problème ne contient plus que 2 recommandations 3.1.1. Le système informe l'acteur que la recommandation ne peut pas être supprimée. 3.1.2. Le cas d'utilisation se termine. 3.2. Sinon 3.2.1. Le système supprime la recommandation. 4. Sinon 4.1. Le cas d'utilisation se termine.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Modify Filter **Scénario 1 : Modify Filter**
ID: UC 18.1
Préconditions : 1. L’acteur a accès à l'interface expert.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur choisit “Set Filter”. 2. Le système demande à l’acteur de fixer les quatres valeurs (type d'utilisateur, endroit où se fait la maintenance, taille de l'organisation, niveau de maturité de l'entreprise) qui déterminent les prochaines questions qui seront introduites dans les nouveaux concepts. 3. L’acteur introduit ces valeurs. 4. Le système modifie la valeur des caractéristiques du filtre.
Postconditions : 1. Les caractéristiques du filtre sont modifiées.

Use Case: Add a Node **Scénario 1 : Add a Node**
ID: UC 19.1
Préconditions : 1. L’acteur a accès à l'interface expert. 2. L'acteur se trouve dans la partie *expert* de l'« Index ».
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur choisit “Add a new Node”. 2. Le système demande à l’acteur d’introduire le nom du nouveau noeud, sa description, et le pourcentage des relations avec les mots-clés sélectionnés. 3. L’acteur introduit ces valeurs. 4. Si le noeud existe déjà 4.1. Le système informe l’acteur que le noeud existe déjà. 4.2. Le cas d’utilisation se termine. 5. Sinon 5.1. Le système ajoute le noeud.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: Consult Case Studies **Scénario 1 : Consult case studies**
ID: UC 20.1
Préconditions : 1. L’acteur a accès à l'interface expert.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur choisit “Cases”. 2. Le système affiche les liens vers les études de cas qui sont entrées dans la base de connaissances.
Postconditions : -

Use Case: Consult Glossary **Scénario 1 : Consult glossary**
ID: UC 21.1
Préconditions : 1. L’acteur a accès à l'interface expert.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur choisit “Glossary”. 2. Le système affiche les mots-clés de la base de connaissances.
Postconditions : -

Cas d'utilisations de l'interface « administrateur »

Use Case: Start a session **Scénario 1 : Start a session**
ID: UC 10.1
Préconditions : 1. L'acteur a démarré le système, i.e. a ouvert la page Web de SMXpert. 2. L'acteur est enregistré auprès du système avec un statut « administrateur ».
Scénario Normal : 1. Le cas d'utilisation commence quand l'acteur ouvre la page Web de SMXpert. 2. L'acteur introduit un nom d'utilisateur et un mot de passe valides 3. Le système vérifie la date d'expiration de l'utilisateur. 4. Le système affiche l'interface correspondante au statut de l'acteur
Scénarios Alternatifs : InvalidUsername ou InvalidPassword DateExpires
Postconditions : 1. L'acteur a accès à la partie « administrateur » du système.

Use Case: Start a session ***Scénario Alternatif 1: InvalidUsername or Invalid Password***
ID: UC 10.1.1
Préconditions : 1. L'acteur a fourni un nom d'utilisateur et/ou un mot de passe non-valide(s).
Scénario Alternatif : 1. Le cas d'utilisation commence à l'étape 2 du cas *Start a session* un nom d'utilisateur et/ou un mot de passé invalide. 2. Le système informe l'acteur qu'il a introduit de mauvaises valeurs et lui demande d'introduire son nom d'utilisateur et son mot de passe à nouveau.
Postconditions : -

Use Case: Start a session
Scénario Alternatif 2: DateExpires
ID: UC 1.1.2
Préconditions : 1. L'acteur a fourni un nom d'utilisateur et le mot de passe d'un compte dont la date de validité est dépassée.
Scénario Alternatif : 1. Le cas d'utilisation commence à l'étape 3 du cas *Start a Session* quand le système vérifie la date d'expiration de l'utilisateur, et que cette date est dépassée. 2. Le système informe l'acteur que sa date d'utilisation a expiré et qu'il n'est plus autorisé à utiliser le système. 3. Le système lui permet de réintroduire un nouveau nom d'utilisateur et mot de passe.
Postconditions : -

Use Case: User Management
Scénario 1 : Add a user
ID: UC 2.1
Préconditions : 1. L'acteur a accès à l'interface administrateur.
Flux des événements : 1. Le cas d'utilisation commence quand l'acteur sélectionne "Add User" dans l'interface du système. 2. Le système demande à l'acteur d'introduire les valeurs en rapport au nouvel utilisateur. 3. L'acteur introduit les valeurs et appuies sur le bouton « Save ». 4. Le système ajoute l'utilisateur.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: User Management **Scénario 2 : Modify profile user**
ID: UC 2.2
Préconditions : 1. L’acteur a accès à l'interface administrateur.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur sélectionne “Edit User Profile” dans l’interface du système, à côté de l'utilisateur dont il veut effectuer les modifications. 2. Le système demande à l’expert d’introduire les nouvelles valeurs. 3. L’expert introduit les nouvelles valeurs et appuies sur le bouton « Save ». 4. Le système modifie le profil de l'utilisateur.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: User Management **Scénario 3 : Modify date**
ID: UC 2.3
Préconditions : 1. L’acteur a accès à l'interface administrateur.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur sélectionne “Change Expiration Date” dans l’interface du système, à côté de l'utilisateur dont il veut effectuer les modifications. 2. Le système demande à l’expert d’introduire une nouvelle date. 3. L’expert introduit le password et appuies sur le bouton « Save ». 4. Le système modifie la date de l'utilisateur.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: User Management **Scénario 4 : Modify password**
ID: UC 2.4
Préconditions : 1. L’acteur a accès à l'interface administrateur.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur sélectionne “Modify Password” dans l’interface du système, à côté de l'utilisateur dont il veut effectuer les modifications. 2. Le système demande à l’expert d’introduire un nouveau mot de passe. 3. L’expert introduit le password et appuies sur le bouton « Save ». 4. Le système modifie le mot de passe de l'utilisateur.
Postconditions : 1. La base de connaissances est mise à jour.

Use Case: User Management **Scénario 5 : Delete user**
ID: UC 2.5
Préconditions : 1. L’acteur a accès à l'interface administrateur.
Flux des événements : 1. Le cas d’utilisation commence quand l’acteur sélectionne “Delete User” dans l’interface du système. 2. Le système supprime l'utilisateur.
Postconditions : 1. La base de connaissancess est mise à jour.

Annexe 3 : Structures de données de SMXpert Web Edition.

Cette annexe présente les principales structures de données de la base de connaissances de SMXpert. Dans chacun des tableaux suivants, les éléments ou attributs en bleu gris indiquent des éléments ou attributs propres à SMXpert et décrits dans le Chapitre 3.

Élément	Description	Attributs	
users	Délimite le document Users	∅	
user	Délimite un utilisateur	**login**	Nom du compte
		pswd	Mot de passe du compte
		status	Statut de l'utilisateur dans le système
		expire	Date d'expiration du compte
filter	Délimite un profil utilisateur	∅	
type_user	Représente le type de l'utilisateur.	∅	
where_maintenance	Représente la localisation de la maintenance	∅	
size_organisation	Représente la taille de l'entreprise de l'utilisateur	∅	
maturity_level	Représente le niveau de maturité des processus de maintenance chez l'utilisateur.	∅	

Description du document Users

Élément	Description	Attributs	
TC	Délimite le document Concept de Maintenance	**name**	Nom du concept de maintenance
		defID	Identifiant du mot-clé définissant le concept de maintenance
Explanation	Délimite une éventuelle explication de concept de maintenance	**name**	Titre de l'explication
also	Délimite une série de mots-clés relatifs au concept de maintenance		
Klink	Représente un lien vers un mot-clé du glossaire	**idref**	Identifiant du mot-clé lié dans le glossaire
REClink	Délimite les recommandations par défaut liée à ce concept pour chaque cas problème		
reco	Représente une référence à une recommandation liée à la question	**href**	Le nom du document cas problème dans lequel trouver la recommandation
		Id	L'identifiant de la recommandation dans ce document.
Themes	Délimite les questions associées au concept de maintenance topologique		
TH	Délimite une question	**id**	Identifiant de la question
		cf	Pourcentage du lien entre la question et son concept de maintenance
question	Délimite la formulation de la question		
REClink	Délimite les recommandations liées à cette question pour chaque cas problème		
reco	Représente une référence à une recommandation liée à la question	**href**	Le nom du document cas problème dans lequel truver la recommandation
		Id	L'identifiant de la recommandation au sein de ce document.
why	Délimite l'explication de la présence de cette question		
rules	Délimite d'éventuelles règles apparentées à cette question issues du SM^{MM}		
fact	Délimite un fait	**name**	Nom du fait
		CF	Facteur permettant le calcul pour déterminer la recommandation à proposer **[pour COSMICXpert]**

Description d'un document Concept de Maintenance

Élément	Description	Attributs	
CP	Délimite le document Cas Problème	**tc**	Référence au concept de maintenance parent
		name	Nom du cas problème
		cs	Référence à l'étude de cas
Explanation	Délimite l'explication du cas problème		
problems	Explication sommaire du cas problème		
Content	Explication détaillée du cas problème		
definitions	Délimite une série de mots-clés relatifs au cas problème		
Klink	Représente un lien vers un mot-clé du glossaire	**idref**	Identifiant du mot-clé lié dans le glossaire
recommandations	Délimite la recommandation associée au cas problème		
REC	Délimite un cas de recommandation	**id**	Identifiant du cas de recommandation
		mincf	Pourcentage **[pour COSMICXpert]**
		maxcf	Pourcentage **[pour COSMICXpert]**
		redir	Identifiant du mot-clé sur laquelle redirige la recommandation au sein du document *glossary_KW*
answer	Délimite la formulation du cas de recommandation		
recom	Délimite l'explication du cas de recommandation		
Klink	Représente d'éventuels liens vers un mot-clé du glossaire	**idref**	Identifiant du mot-clé lié dans le glossaire

Description d'un document *Cas Problème*

Élément	Description	Attributs	
glossary	Délimite le document glossary_KW	∅	
keyword	Délimite un mot-clé	**id**	Identifiant du mot-clé
		name	Nom du mot-clé
définition	Délimite la définition de ce mot-clé.	∅	

Description du document *glossary_KW*

Élément	Description	Attributs	
glossary	Délimite le document glossary_INDEX	∅	
node	Délimite un mot d'index	**id**	Identifiant du mot d'index
		name	Nom du mot d'index
définition	Délimite la définition de ce mot d'index.	∅	

Description du document *glossary_INDEX*

Le document suivant établit un lien entre les mots d'index et les mots-clés vers lesquels ils conduisent.

Élément	Description	Attributs	
search	Délimite le document searching	∅	
n	Délimite un mot d'index	**IdRef**	Identifiant du mot d'index dans le document *glossary_INDEX*
k	Représente un mot-clé lié au mot d'index	**IdRef**	Identifiant du mot-clé dans le document *glossary_KW*
		cf	Le pourcentage du lien entre ce mot-clé et le mot d'index

Description du document *searching*

Le document suivant sert à modéliser les liens entre les concepts de maintenance et les mots-clés ou cas problèmes qui lui sont associés.

Élément	Description	Attributs	
xpert	Délimite le document xpert	∅	
tc	Délimite un concept de maintenance	**href**	Référence le document décrivant le concept en question
kw	Représente une référence à un mot-clé associé au concept	**cf**	Pourcentage du lien entre le mot-clé et son concept de maintenance
		IdRef	L'identifiant du mot clé dans le document *glossary_KW*
cp	Représente une référence à un cas problème associé au concept	**cf**	Pourcentage du lien entre cas problème et son concept de maintenance
		href	Référence le document décrivant le cas problème en question

Description du document *xpert*

Annexe 4 : Page JSP de la version initiale de COSMICXpert (« modifyKeyword.jsp »)

Le code source de la page JSP qui suit provient du mémoire de C. Duterme et N. Fabry [3].

Cette page correspond à la fonctionnalité de « modification d'un mot-clé ». Nous pouvons remarquer dans celle-ci les appels aux classes faisant partie de la logique d'affaires. Par exemple, les classes « Glossary.java », « Xpert.java », « TopologicalConcept.java » sont instanciées. Cette façon de procéder est contraire aux principes de séparation des différentes couches de l'application.

« modifyKeyword .jsp »

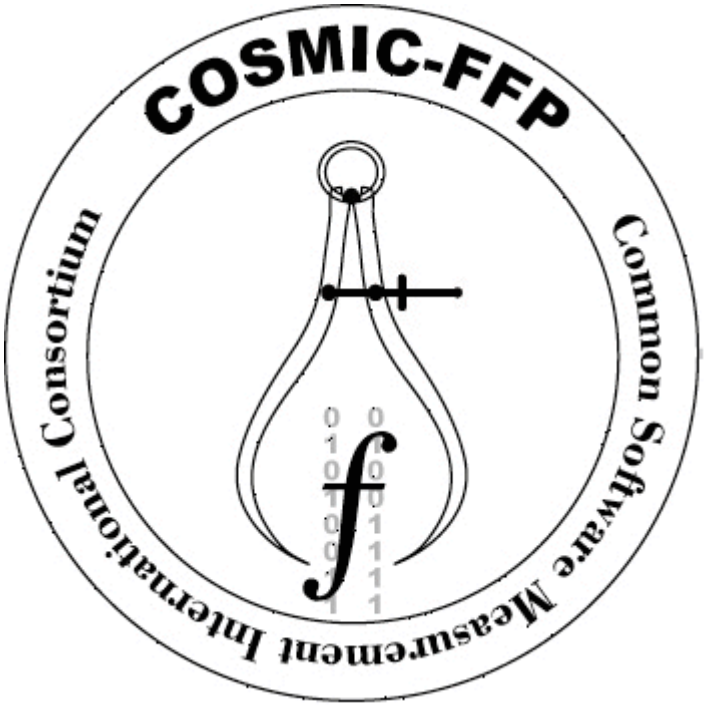
```
</script>
</head>
</html>
<body>
  <script type="text/javascript">
      document.onmousedown = noclic;
  </script>
  <form action="KeywordEXE.jsp">
      <h2>Please insert the following informations :</h2>
  <%
      //TIP : rajouter une balise hidden dans le XSL pour pouvoir passer
une variable à la prochaine page
      String choosen = request.getParameter("choosen");
      Glossary glos = new Glossary();
      glos.loadXML(application.getRealPath("KB/glossary.xml"));
      glos.setXslFile(application.getRealPath("KB/Xsl/kwInfo.xsl"));
       glos.addTransformParam("kw",choosen);
       glos.transform(out);
      Xpert xp = new Xpert();
      xp.loadXML(application.getRealPath("KB/xpert.xml"));
      xp.setXslFile(application.getRealPath("KB/Xsl/tcList.xsl"));
      xp.addTransformParam("kw",choosen);
       xp.transform(out);
  /* Il faut rajouter en champs HIDDEN les champs qui sont "disabled" (car
  ceux-ci ne sont plus passés car "disabled");
```

```
      Cad tous les TC dont le defId = au keyword qu'on modifie
      Remarque : si on donne le même nom aux champs HIDDEN,
      c'est bien : ils sont concaténer avec les CB du même nom récupérés dans le XSL
  */
      TopologicalConcept tc = new TopologicalConcept();
      String path = application.getRealPath(File.separator);
      System.out.println("path : "+path);
      Vector tcList = tc.getTcListDefId(path,choosen);
      int nbTc = tcList.size();
      System.out.println("Nombre de TC rattaché à "+choosen+" : "+Integer.toString(nbTc));
      int i=0;
      while(i<nbTc){
            String arg = (String)tcList.get(i);
            %>
            <input type="hidden" name="tcCB" value="<%=arg%>"/>
            <%
            i=i+1;
      }
  %>
      <br>
      <input name="action" type="submit" value="Modify"/>
      <input                                                    type="button" onClick="gotopage('cancel.jsp?tree=KB/Xsl/KwTcCpTree.xsl')" value="Cancel"/>
  </form>
  <script type="text/javascript">
      var config = new HTMLArea.Config();
      config.editorURL = "../COSMICXpert/htmlarea/";
      editor = new HTMLArea("description", config);
      editor.generate();
  </script>
```

Annexe 5 : Plan et rapport de tests de COSMICXpert

Voir page suivante.

CosmicXpert Web Edition



Plan et rapport de tests

Auteurs :	**Stéphane Sandron** **Benoît Vanderose**
Date :	**5 octobre 2004**

Table des Matières

Première Partie : Plan de tests.

Remarques préliminaires

Après avoir mieux appréhendé le fonctionnement de **CosmicXpert Web Edition**, nous sommes arrivé à la conclusion qu'il était possible de considérer le système dans son ensemble comme l'union de trois systèmes distincts, visant chacun un type d'utilisateur bien particulier. Par conséquent, l'idée de diviser notre plan de test en trois parties indépendantes nous est naturellement venue à l'esprit.

Nous avons dans un premier temps isolé les différentes catégories d'utilisateurs. Ces catégories sont :

- Administrateur
- Expert
- Mesureur

Nous avons finalement décidé de répartir les cas de test en trois grandes catégories correspondant aux fonctionnalités spécifiques à chaque type d'utilisateur, en y ajoutant une quatrième reprenant les fonctionnalités communes à tous. Par 'spécifique', nous entendons, par exemple, que même si un administrateur a la possibilité d'accéder à l'interface du mesureur et le l'expert, les cas de test de la catégorie **administrateur** ne reprendront pas les tests sur ces deux dernières interfaces, mais bien la partie propre à l'administrateur, c'est-à-dire la gestion des utilisateurs.

Dans la suite, les cas de tests seront identifiés par un code dont le format générique est :

COSMICXXX-**CT**-Yz

Le XXX de cette représentation correspond à un code de trois lettres renseignant sur la catégorie à laquelle se rattache le cas de test visé. La sémantique de ce code est donnée par la table suivante :

Code	**Catégorie**
ADM	Fonctionnalité propre à l'administrateur
EXP	Fonctionnalité propre à l'expert
MES	Fonctionnalité propre au mesureur
COM	Fonctionnalité commune à tous

Le Y de cette représentation correspond à un nombre identifiant le cas de test de manière unique. Pour sa part, le x fait référence à une lettre minuscule qui servira à identifier les éventuelles variantes d'un même test.

Liste des cas de test identifiés

Dans cette section, nous dressons la liste des cas qui constitueront les jalons de notre plan de test. Le tableau ci-dessous ne liste que les cas de test et non chacune de leur variante. Il constitue donc l'ossature de ce dernier et, de plus, permettra de fournir au lecteur une correspondance entre les codes de chaque cas et leur description.

ID	*Description*
COSMICADM-CT-01	Ajout d'un utilisateur à la liste des utilisateurs du système.
COSMICADM-CT-02	Suppression d'un utilisateur.
COSMICADM-CT-03	Changer le mot de passe d'un utilisateur.
COSMICADM-CT-04	Changer la date d'expiration d'un compte utilisateur.
COSMICADM-CT-05	Changer le statut d'un utilisateur.
COSMICADM-CT-06	Restaurer la base de connaissance.
COSMICADM-CT-07	Interaction avec l'interface Mesureur.
COSMICADM-CT-08	Interaction avec l'interface Expert.
COSMICEXP-CT-01	Consulter un arbre.
COSMICEXP-CT-02	Ajouter un mot-clef.
COSMICEXP-CT-03	Ajouter un concept topologique.
COSMICEXP-CT-04	Ajouter un cas problème.
COSMICEXP-CT-05	Ajouter un thème.
COSMICEXP-CT-06	Ajouter un fait.
COSMICEXP-CT-07	Ajouter une recommandation.
COSMICEXP-CT-08	Modifier un mot-clef.
COSMICEXP-CT-09	Modifier un concept topologique.
COSMICEXP-CT-10	Modifier un cas problème.
COSMICEXP-CT-11	Modifier un thème.
COSMICEXP-CT-12	Modifier un fait.
COSMICEXP-CT-13	Modifier une recommandation.
COSMICEXP-CT-14	Effacer un mot-clef.
COSMICEXP-CT-15	Modifier un concept topologique.
COSMICEXP-CT-16	Effacer un cas problème.
COSMICEXP-CT-17	Effacer un thème.

COSMICEXP-CT-18	Effacer un fait.
COSMICEXP-CT-19	Effacer une recommandation.
COSMICMES-CT-01	Consulter la définition d'un mot-clé.
COSMICMES-CT-02	Recherche des concepts topologiques en rapport à un mot-clef.
COSMICMES-CT-03	Recherche par index.
COSMICMES-CT-04	Consulter la définition des concepts topologiques/cas problèmes/thèmes/recommandations.
COSMICMES-CT-05	Obtenir une recommandation.
COSMICCOM-CT-1	Se connecter.
COSMICCOM-CT-2	Se déconnecter.

Spécifications des cas de test (partie administrateur)

Dans cette section, nous décrivons en détail les différents cas de test relatifs aux fonctionnalités de l'administrateur. Nous donnons également les différentes variantes basées sur un même test.

Description de COSMICADM-CT-01

ID :	COSMICADM-CT-1a
Description :	Ajouter un utilisateur à la liste des utilisateurs du système.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le bouton **Add User**. Il entre le login, le mot de passe, le type et, éventuellement, la date d'expiration du compte du nouvel utilisateur. Il valide en cliquant **Ok**. On veillera, dans la présente variation du cas de test, à ce que les données soient entièrement cohérentes, du point de vue de la syntaxe et de la sémantique. Ainsi, on enregistrera un utilisateur non encore existant dans le système, en veillant à respecter la syntaxe prescrite. De même, on veillera à ce qu'aucune incohérence n'apparaisse dans la date d'expiration du compte, tant du point de vue syntaxique que sémantique.
Résultat(s) attendu(s) :	On attend du système que le nouvel utilisateur soit enregistré auprès de lui. L'utilisateur peut alors accéder au système par le biais de son *login* et *mot de passe*. De plus, les caractéristiques de son compte son bien conformes à ce qui a été entré, i.e. la date d'expiration ainsi que l'interface (mesureur ou expert) auquel il a un droit d'accès.

ID :	COSMICADM-CT-1b
Description :	Ajouter un utilisateur à la liste des utilisateurs du système.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le bouton **Add User**. Il entre le login, le mot de passe, le type et, éventuellement, la date d'expiration du compte du nouvel utilisateur. Il valide en cliquant sur **Ok**. Dans cette variante, nous veillerons à introduire un utilisateur déjà présent dans le système (même login et même type d'utilisateur).
Résultat(s) attendu(s) :	Nous attendons du système qu'il refuse d'enregistrer ce nouvel utilisateur en notifiant l'administrateur de la raison de ce refus.

ID :	COSMICADM-CT-1c
Description :	Ajouter un utilisateur à la liste des utilisateurs du système.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le bouton **Add User**. Il entre le *login,* le *mot de passe*, le *type* et, éventuellement, la *date d'expiration du compte* du nouvel utilisateur. Il valide en cliquant sur **Ok**. Dans cette variante, nous veillerons à introduire un utilisateur de même *login* qu'un utilisateur préexistant mais dont le *type d'utilisateur* est différent.
Résultat(s) attendu(s) :	Nous attendons du système qu'il refuse d'enregistrer ce nouvel utilisateur en notifiant l'administrateur de la raison de ce refus.

ID :	COSMICADM-CT-1d
Description :	Ajouter un utilisateur à la liste des utilisateurs du système.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le bouton **Add User**. Il entre le *login,* le *mot de passe*, le *type* et, éventuellement, la *date d'expiration du compte* du nouvel utilisateur. Il valide en cliquant sur **Ok**. Dans cette variante, nous allons tenter d'introduire un utilisateur dont la date d'expiration sera inférieure à la date actuelle du système.
Résultat(s) attendu(s) :	Nous attendons du système qu'il refuse d'enregistrer ce nouvel utilisateur en notifiant la raison de ce refus ou, du moins, qu'il prenne bien en compte que cet utilisateur n'est pas actif

ID :	COSMICADM-CT-1e
Description :	Ajouter un utilisateur à la liste des utilisateurs du système.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le bouton **Add User**. Il entre le *login,* le *mot de passe*, le *type* et, éventuellement, la *date d'expiration du compte* du nouvel utilisateur. Il valide en cliquant sur **Ok**. Dans cette variante, nous veillerons à introduire un utilisateur dont la date d'expiration sera égale à la date actuelle du système.
Résultat(s) attendu(s) :	Nous attendons du système qu'il refuse d'enregistrer ce nouvel utilisateur en notifiant la raison de ce refus ou, du moins, qu'il prenne bien en compte que cet utilisateur n'est pas actif

ID :	COSMICADM-CT-1f
Description :	Ajouter un utilisateur à la liste des utilisateurs du système.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le bouton **Add User** ». Il entre le *login,* le *mot de passe*, le *type* et, éventuellement, la *date d'expiration du compte* du nouvel utilisateur. Il valide en cliquant sur **Ok**. Dans cette variante, nous veillerons à introduire un utilisateur dont la date d'expiration sera de syntaxe incorrecte.
Résultat(s) attendu(s) :	Nous attendons du système qu'il refuse d'enregistrer ce nouvel utilisateur en notifiant la raison de ce refus.

Description de COSMICADM-CT-02

ID :	COSMICADM-CT-2
Description :	Supprimer un utilisateur de la liste des utilisateurs du système.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le bouton **Delete User** de la ligne correspondant à l'utilisateur qu'il veut supprimer.
Résultat(s) attendu(s) :	Nous attendons du système qu'il efface de la liste des utilisateurs, celle qui est relative à l'utilisateur supprimé et que l'accès au système soit désormais interdit depuis avec ce compte.

Description de COSMICADM-CT-03

ID :	COSMICADM-CT-3a
Description :	Changer le mot de passe d'un utilisateur.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le bouton **Change Password** de la ligne correspondant à l'utilisateur qu'il veut modifier. Il introduit le nouveau mot de passe, différent du précédent, et valide en cliquant sur le bouton **Change Password** .
Résultat(s) attendu(s) :	Nous attendons du système qu'il mette à jour le mot de passe de l'utilisateur et ne permette plus l'accès au système avec l'ancien mot de passe.

ID :	COSMICADM-CT-3b
Description :	Changer le mot de passe d'un utilisateur.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le bouton **Change Password** de la ligne correspondant à l'utilisateur qu'il veut modifier. Dans cette variante, il introduit un nouveau mot de passe semblable au précédent et valide en cliquant sur le bouton **Change Password** .
Résultat(s) attendu(s) :	Nous n'attendons du système aucune modification de son état.

ID :	COSMICADM-CT-3c
Description :	Changer le mot de passe d'un utilisateur.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le bouton **Change Password** de la ligne correspondant à l'utilisateur qu'il veut modifier. Il n'introduit pas de nouveau mot de passe et valide en cliquant sur le bouton **Change Password** .
Résultat(s) attendu(s) :	Nous attendons du système qu'il refuse cette opération, notifiant l'utilisateur de la raison de son refus.

Description de COSMICADM-CT-04

ID :	COSMICADM-CT-4a
Description :	Changer la date d'expiration d'un compte utilisateur.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur remplit le champ **Expires(YYYY-MM-DD)** avec une date cohérente, tant du point de vue de la syntaxe que de la sémantique.
Résultat(s) attendu(s) :	A la date spécifiée, le système doit désactiver le compte utilisateur.

ID :	COSMICADM-CT-4b
Description :	Changer la date d'expiration d'un compte utilisateur.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)

Entrée :	L'administrateur remplit le champ **Expires(YYYY-MM-DD)** avec une date inférieure à la date actuelle du système, cohérente au point de vue de la syntaxe.
Résultat(s) attendu(s) :	On attend du système qu'il refuse cette date et notifie l'administrateur de la raison du refus.

ID :	COSMICADM-CT-4c
Description :	Changer la date d'expiration d'un compte utilisateur.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur remplit le champ **Expires(YYYY-MM-DD)** avec une date égale à la date actuelle du système, cohérente au point de vue de la syntaxe.
Résultat(s) attendu(s) :	On attend du système qu'il refuse cette date et notifie l'administrateur de la raison du refus.

ID :	COSMICADM-CT-4d
Description :	Changer la date d'expiration d'un compte utilisateur.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur remplit le champ **Expires(YYYY-MM-DD)** avec une date dont le format n'est pas conforme au **YYYY-MM-DD** prescrit.
Résultat(s) attendu(s) :	On attend du système qu'il refuse cette date et notifie l'administrateur de la raison du refus.

ID :	COSMICADM-CT-4e
Description :	Changer la date d'expiration d'un compte utilisateur.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur supprime le contenu du champ **Expires(YYYY-MM-DD)**.
Résultat(s) attendu(s) :	On attend du système qu'il supprime la contrainte d'expiration du compte utilisateur.

Description de COSMICADM-CT-05

ID :	COSMICADM-CT-5
Description :	Changer le statut d'un utilisateur.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur sélectionne dans la liste déroulante **Status** relative à la ligne de l'utilisateur visé, le nouveau statut qu'il veut lui accorder.
Résultat(s) attendu(s) :	On attend du système qu'il enregistre le changement de statut de l'utilisateur visé et que les privilèges associés à ce statut soient accordés à cet utilisateur à la prochaine connexion.

Description de COSMICADM-CT-06

ID :	COSMICADM-CT-6
Description :	Restaurer la base de connaissance.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le bouton **Restore Knowledge Base.**

Résultat(s) attendu(s) :	On attend du système qu'il revienne son état initial, c'est-à-dire que toutes les modifications effectuées par les utilisateurs soient invalidées.

Description de COSMICADM-CT-07

ID :	COSMICADM-CT-7a
Description :	Interaction avec l'interface Mesureur.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le lien hypertexte **Measurer Page.** Ensuite, il ferme l'interface mesureur.
Résultat(s) attendu(s) :	On attend du système que l'interface Mesureur apparaisse puis se ferme, revenant ainsi à l'état initial du test.

ID :	COSMICADM-CT-7b
Description :	Interaction avec l'interface Mesureur.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le lien hypertexte **Measurer Page.** Ensuite, il ferme l'interface administrateur, par le lien hypertexte **Logout**.
Résultat(s) attendu(s) :	On attend du système que l'interface Mesureur se ferme en même temps que l'interface Administrateur.

ID :	COSMICADM-CT-7c
Description :	Interaction avec l'interface Mesureur.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le lien hypertexte **Measurer Page.** Ensuite, il ferme l'interface administrateur, grâce à la croix du coin supérieur droit.
Résultat(s) attendu(s) :	On attend du système que l'interface Mesureur se ferme en même temps que l'interface Administrateur.

Description de COSMICADM-CT-08

ID :	COSMICADM-CT-8
Description :	Interaction avec l'interface Expert.
Pré condition :	L'administrateur doit être connecté au système (login et mot de passe correct)
Entrée :	L'administrateur clique sur le lien hypertexte **Expert Page.** Ensuite, il revient à l'interface administrateur, par le lien hypertexte adéquat.
Résultat(s) attendu(s) :	On attend du système qu'il revienne à son état initial (interface administrateur).

Spécifications des cas de test (partie expert)

Dans cette section, nous décrivons en détail les différents cas de test relatifs aux fonctionnalités de l'expert. Nous donnons également les différentes variantes basées sur un même test.

Description de COSMICEXP-CT-01

ID :	COSMICEXP-CT-1
Description :	Consulter un arbre.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert.
Entrée :	L'utilisateur clique sur le bouton **Xpert Trees.** Il sélectionne ensuite la catégorie **Keywords – Topological Concepts – Case problems Tree, Topological Concepts – Themes – Facts Tree** ou **Case Problems – Recommandations Tree**.
Résultat(s) attendu(s) :	On attend du système qu'il affiche l'arbre correct, c'est-à-dire reflétant la base de connaissance actuelle, correspondant à la décomposition choisie.

Description de COSMICEXP-CT-02

ID :	COSMICEXP-CT-2a
Description :	Ajouter un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert.
Entrée :	L'utilisateur clique sur le bouton **Edit.** Ensuite sur **Add a keyword**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Dans cette variante, nous veillerons à ce que le mot-clef soit différent de ceux existants auparavant et qu'au moins une case de la colonne **Related Topological Concept(s)** soit cochée et le pourcentage associé correctement donné.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour la base de connaissance actuelle en y ajoutant le mot-clef, ainsi que le pourcentage correct, dans le domaine du (des) concept(s) topologique(s) choisi(s) lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-2b
Description :	Ajouter un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert.
Entrée :	L'utilisateur clique sur le bouton **Edit.** Ensuite sur **Add a keyword**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Dans cette variante, nous veillerons à ce que le mot-clef soit différent de ceux existants auparavant, qu'au moins une case de la colonne **Related Topological Concept(s)** soit cochée et que, dans une de ces dernières, le pourcentage soit égal à -101 ou 101.
Résultat(s) attendu(s) :	On attend du système qu'il refuse d'ajouter le mot-clef à la base de connaissance actuelle.

ID :	COSMICEXP-CT-2c
Description :	Ajouter un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert.
Entrée :	L'utilisateur clique sur le bouton **Edit.** Ensuite sur **Add a keyword**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Dans cette variante, nous veillerons à ce que le mot-clef soit différent de ceux existants auparavant, qu'au moins une case de la colonne **Related Topological Concept(s)** soit cochée et que, dans une de ces dernières, le pourcentage soit égal à 0.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant le mot-clef, ainsi que le pourcentage correct, dans le domaine du (des) concept(s) topologique(s) choisi(s) lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-2d
Description :	Ajouter un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert.
Entrée :	L'utilisateur clique sur le bouton **Edit.** Ensuite sur **Add a keyword**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associée à chacun de ceux-ci. Nous veillerons à ce que le mot-clef soit le même qu'un mot-clef existant et qu'au moins une case de la colonne **Related Topological Concept(s)** soit cochée.
Résultat(s) attendu(s) :	On attend du système qu'il refuse d'ajouter le mot-clef à la base de connaissance actuelle.

ID :	COSMICEXP-CT-2e
Description :	Ajouter un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert.
Entrée :	L'utilisateur clique sur le bouton **Edit.** Ensuite sur **Add a keyword**. Il ne remplit aucun champ du formulaire.
Résultat(s) attendu(s) :	On attend du système qu'il refuse d'ajouter le mot-clef à la base de connaissance actuelle.

ID :	COSMICEXP-CT-2f
Description :	Ajouter un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert.
Entrée :	L'utilisateur clique sur le bouton **Edit.** Ensuite sur **Add a keyword**. L'expert annule l'ajout du mot-clef en cliquant sur le bouton **Cancel**.
Résultat(s) attendu(s) :	On attend du système qu'il ne modifie pas la base de connaissance actuelle et que l'on se retrouve sur la page de laquelle l'utilisateur a initié l'ajout.

Description de COSMICEXP-CT-03

ID :	COSMICEXP-CT-3a
Description :	Ajouter un concept topologique.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	Au cours des neuf étapes qui constitue l'ajout d'un concept topologique, l'expert remplit ***tous*** les champs obligatoires et, éventuellement, d'autres facultatifs en respectant la syntaxe et la sémantique de ces derniers. A chaque étape, il valide en cliquant sur le bouton **Next**. Nous veillerons à ce que l'ensemble des données introduites au cours de ces étapes constitue un concept topologique non encore existant.
Résultat(s) attendu(s) :	Nous attendons du système qu'il enregistre le nouveau concept topologique dans la base de connaissance. Nous pouvons vérifier ceci en consultant la complétude des arbres.

ID :	COSMICEXP-CT-3b
Description :	Ajouter un concept topologique.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	Au cours des neuf étapes qui constitue l'ajout d'un concept topologique, l'expert remplit ***tous*** les champs obligatoires et, éventuellement, d'autres facultatifs en respectant la syntaxe et la sémantique de ces derniers. A chaque étape, il valide en cliquant sur le bouton **Next**. Nous veillerons à ce que l'ensemble des données introduites au cours de ces étapes constitue un concept topologique déjà existant.
Résultat(s) attendu(s) :	Nous attendons du système qu'il refuse d'enregistrer le nouveau concept topologique dans la base de connaissance et notifie l'utilisateur de la raison de ce refus.

ID :	COSMICEXP-CT-3c
Description :	Ajouter un concept topologique.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	Au cours des neuf étapes qui constituent l'ajout d'un concept topologique, l'expert ne remplit pas certains champs obligatoires. A chaque étape, il valide en cliquant sur le bouton **Next**. Nous veillerons à ce que l'ensemble des données introduites au cours de ces étapes constitue un concept topologique non encore existant.
Résultat(s) attendu(s) :	Nous attendons du système qu'il refuse de passer à l'étape ultérieure lorsqu'un champ obligatoire est omis et notifie l'utilisateur de la raison de ce refus.

ID :	COSMICEXP-CT-3d
Description :	Ajouter un concept topologique.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	Au cours des neuf étapes qui constituent l'ajout d'un concept topologique, l'expert remplit certains champs obligatoires avec des données aberrantes, tel que des pourcentages supérieurs à 100. A chaque étape, il valide en cliquant sur le bouton **Next**.
Résultat(s) attendu(s) :	Nous attendons du système qu'il refuse de passer à l'étape ultérieure tant qu'un champ obligatoire contient des données erronées et notifie l'utilisateur de la raison de ce refus.

ID :	COSMICEXP-CT-3e
Description :	Ajouter un concept topologique.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	Cette variante vise à tester la capacité d'annulation d'une étape. Nous veillerons à ce que chaque étape soit systématiquement annulée par l'expert, par un clic sur **Cancel**.
Résultat(s) attendu(s) :	Nous attendons du système que chaque retour en arrière s'effectue normalement, sans perte d'information concernant l'étape à laquelle nous désirons revenir.

Description de COSMICEXP-CT-04

ID :	COSMICEXP-CT-4a
Description :	Ajouter un cas problème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..** . L'utilisateur clique sur **Add a Case Problem**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons, dans cette variante, à ce que le cas problème soit différent de ceux existants auparavant.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant le cas problème, ainsi que le pourcentage correct, dans le domaine du concept topologique choisi lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-4b
Description :	Ajouter un cas problème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. Nous veillerons à laisser vide les champs de chaque formulaire où il est possible de passer à une étape suivante.

Résultat(s) attendu(s) :	On attend du système qu'il refuse de passer à l'étape suivante de la procédure d'ajout du cas problème.

ID :	COSMICEXP-CT-4c
Description :	Ajouter un cas problème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur ne sélectionne pas de concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**.
Résultat(s) attendu(s) :	On attend du système qu'il avertisse l'utilisateur qu'il ne peut rajouter un cas problème que s'il est lié à un concept topologique.

ID :	COSMICEXP-CT-4d
Description :	Ajouter un cas problème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'utilisateur clique sur **Add a Case Problem**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que le cas problème soit le même qu'un autre existant déjà auparavant et absent du domaine du concept topologique sélectionné.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant le cas problème, ainsi que le pourcentage correct, dans le domaine du concept topologique choisi lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-4e
Description :	Ajouter un cas problème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'utilisateur clique sur **Add a Case Problem**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous prêterons attention à ce que le cas problème soit le même qu'un autre existant déjà auparavant et présent au sein du domaine du concept topologique sélectionné.
Résultat(s) attendu(s) :	On attend du système qu'il refuse de mettre la base de connaissance à jour et qu'il avertisse l'utilisateur que le cas problème existe déjà.

ID :	COSMICEXP-CT-4f
Description :	Ajouter un cas problème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'utilisateur clique sur **Add a Case Problem**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que le cas problème soit différent de ceux existant auparavant et que le pourcentage soit égal à -101 ou 101 (dans le premier formulaire).
Résultat(s) attendu(s) :	On attend du système qu'il refuse de passer à l'étape suivante de la procédure d'ajout du cas problème.

ID :	COSMICEXP-CT-4g
Description :	Ajouter un cas problème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..** . L'utilisateur clique sur **Add a Case Problem**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous nous assurerons que le cas problème soit différent de ceux existant auparavant et que le pourcentage soit égal à 0 (dans le premier formulaire).
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant le cas problème, ainsi que le pourcentage correct, dans le domaine du concept topologique choisi lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-4h
Description :	Ajouter un cas problème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'utilisateur clique sur **Add a Case Problem**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ne pas cocher de case dans la partie **Related Case Study** du second formulaire.
Résultat(s) attendu(s) :	On attend du système qu'il refuse de passer à l'étape suivante de la procédure d'ajout du cas problème.

ID :	COSMICEXP-CT-4i
Description :	Ajouter un cas problème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'utilisateur clique sur **Add a Case Problem**. L'expert annule l'ajout d'un cas problème en cliquant sur le bouton **Cancel** (à n'importe quel moment de la procédure).
Résultat(s) attendu(s) :	On attend du système qu'il ne modifie pas la base de connaissance actuelle et que l'on se retrouve sur la page de laquelle l'utilisateur a initié l'ajout.

Description de COSMICEXP-CT-05

ID :	COSMICEXP-CT-5a
Description :	Ajouter un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'utilisateur clique sur **Add a Theme**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que la question soit différente de celles existant auparavant.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant le thème, ainsi que le pourcentage correct des faits liés, dans le domaine du concept topologique choisi lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-5b
Description :	Ajouter un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'utilisateur clique sur **Add a Theme**. Nous veillerons a laisser vide les champs de chaque formulaire où il y a moyen de passer à une étape suivante.
Résultat(s) attendu(s) :	On attend du système qu'il refuse de passer à l'étape suivante de la procédure d'ajout d'un thème.

ID :	COSMICEXP-CT-5c
Description :	Ajouter un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur ne sélectionne pas un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add ...** L'utilisateur clique sur **Add a Theme**.

Résultat(s) attendu(s) :	On attend du système qu'il avertisse l'utilisateur de l'impossibilité d'ajouter un thème s'il n'est lié à un concept topologique.

ID :	COSMICEXP-CT-5d
Description :	Ajouter un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'utilisateur clique sur **Add a Theme**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que la question soit la même qu'un autre thème existant déjà auparavant et présent au sein du domaine du concept topologique sélectionné.
Résultat(s) attendu(s) :	On attend du système qu'il refuse de mettre la base de connaissance à jour et qu'il avertisse l'utilisateur que le thème existe déjà.

ID :	COSMICEXP-CT-5e
Description :	Ajouter un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'utilisateur clique sur **Add a Theme**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que la question soit la même qu'un autre thème existant déjà auparavant et non présent au sein du domaine du concept topologique sélectionné.
Résultat(s) attendu(s) :	On attend du système qu'il refuse de mettre la base de connaissance à jour et qu'il avertisse l'utilisateur que le thème existe déjà.

ID :	COSMICEXP-CT-5f
Description :	Ajouter un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add ...**. L'utilisateur clique sur **Add a Theme**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que la question soit différente de celles existants auparavant et que le pourcentage soit égal à -101 ou 101 (dans le premier formulaire).
Résultat(s) attendu(s) :	On attend du système qu'il refuse de passer à l'étape suivante de la procédure d'ajout du thème.

ID :	COSMICEXP-CT-5g
Description :	Ajouter un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'utilisateur clique sur **Add a Theme**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que la question soit différente de celles existants auparavant et que le pourcentage soit égal à 0 (dans le premier formulaire).
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant le thème, ainsi que le pourcentage correct des faits liés, dans le domaine du concept topologique choisi lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-5h
Description :	Ajouter un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'utilisateur clique sur **Add a Theme**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que la question soit différente de celles existants auparavant et que le pourcentage de *Yes* de la colonne *Facts* soit strictement plus grand que le pourcentage de *No* de la colonne *Facts* (du premier formulaire), tout en étant compris entre -100 et100.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant le thème, ainsi que le pourcentage correct des faits liés, dans le domaine du concept topologique choisi lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-5i
Description :	Ajouter un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add ..**. L'utilisateur clique sur **Add a Theme**. On remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que la question soit différente de celles existants auparavant et que le pourcentage de Yes de la colonne Facts soit strictement plus petit que le pourcentage de No de la colonne Facts (dans le premier formulaire), tout en étant compris entre -100 et100.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant le thème, ainsi que le pourcentage correct des faits liés, dans le domaine du concept topologique choisi lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-5j
Description :	Ajouter un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add ..**. L'utilisateur clique sur **Add a Theme**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que la question soit différente de celles existants auparavant et que le pourcentage de Yes de la colonne Facts égale le pourcentage de No de la colonne Facts (dans le premier formulaire), tout en étant compris entre -100 et100.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant le thème, ainsi que le pourcentage correct des faits liés, dans le domaine du concept topologique choisi lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-5k
Description :	Ajouter un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add ..**. L'utilisateur clique sur **Add a Theme**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que la question soit différente de celles existants auparavant, que le pourcentage de Yes de la colonne Facts et celui de No de la colonne Facts (dans le premier formulaire) soient égaux à -101 et/ou 101.
Résultat(s) attendu(s) :	On attend du système qu'il refuse de cette opération et notifie l'utilisateur de la raison de ce refus.

ID :	COSMICEXP-CT-5l
Description :	Ajouter un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree** ou l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'utilisateur clique sur **Add a Theme**. L'expert annule l'ajout d'un cas problème en cliquant sur le bouton **Cancel** (à n'importe quel moment de la procédure).
Résultat(s) attendu(s) :	On attend du système qu'il ne modifie pas la base de connaissance actuelle et que l'on se retrouve sur la page de laquelle l'utilisateur a initié l'ajout.

Description de COSMICEXP-CT-06

ID :	COSMICEXP-CT-6a
Description :	Ajouter un fait.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un thème et clique sur le bouton **Edit.** Ensuite sur **Add..**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que le fait soit différent de ceux existants auparavant dans le thème sélectionné.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant le fait, ainsi que le pourcentage correct, dans le domaine du thème choisi lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-6b
Description :	Ajouter un fait.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un thème et clique sur le bouton **Edit.** Ensuite sur **Add..**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que le fait soit le même que ceux existants auparavant dans le thème sélectionné et avec un pourcentage différent.
Résultat(s) attendu(s) :	On attend du système qu'il refuse d'ajouter le fait à la base de connaissance et qu'il revienne à la page où l'utilisateur se trouvait précédemment la tentative d'ajout d'un fait.

ID :	COSMICEXP-CT-6c
Description :	Ajouter un fait.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un thème et clique sur le bouton **Edit.** Ensuite sur **Add..**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que le fait soit le même que ceux existants auparavant dans le thème sélectionné et avec le même pourcentage.

Résultat(s) attendu(s) :	On attend du système qu'il refuse d'ajouter le fait à la base de connaissance et qu'il revienne à la page où l'utilisateur se trouvait précédemment la tentative d'ajout d'un fait.

ID :	COSMICEXP-CT-6d
Description :	Ajouter un fait.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un thème et clique sur le bouton **Edit.** Ensuite sur **Add ...**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que le fait soit différent de ceux existants auparavant dans le thème sélectionné et à ce que le pourcentage soit égal à -101 ou 101.
Résultat(s) attendu(s) :	On attend du système qu'il refuse d'ajouter le fait à la base de connaissance et qu'il revienne à la page où l'utilisateur se trouvait précédemment la tentative d'ajout d'un fait.

ID :	COSMICEXP-CT-6e
Description :	Ajouter un fait.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un thème et clique sur le bouton **Edit.** Ensuite sur **Add..**. On remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que le fait soit différent de ceux existants auparavant dans le thème sélectionné et à ce que le pourcentage soit égal à 0.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant le fait, ainsi que le pourcentage correct, dans le domaine du thème choisi lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-6f
Description :	Ajouter un fait.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un thème et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'expert annule l'ajout d'un cas problème en cliquant sur le bouton **Cancel** (à n'importe quel moment de la procédure).
Résultat(s) attendu(s) :	On attend du système qu'il ne modifie pas la base de connaissance actuelle et que l'on se retrouve sur la page de laquelle l'utilisateur a initié l'ajout.

Description de COSMICEXP-CT-07

ID :	COSMICEXP-CT-7a
Description :	Ajouter une recommandation.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Case Problems – Recommandation Tree.**
Entrée :	L'utilisateur sélectionne un cas problème et clique sur le bouton **Edit.** Ensuite sur **Add..**. Il remplit les champs du formulaire en respectant la

	syntaxe et la sémantique associées à chacun de ceux-ci.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant la recommandation, ainsi que les pourcentages corrects, dans le domaine du cas problème choisi lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-7b
Description :	Ajouter une recommandation.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Case Problems – Recommandation Tree.**
Entrée :	L'utilisateur sélectionne un cas problème et clique sur le bouton **Edit.** Ensuite sur **Add..**. On ne remplit aucun champ du formulaire.
Résultat(s) attendu(s) :	On attend du système qu'il refuse de passer à l'étape suivante de la procédure d'ajout de la recommandation.

ID :	COSMICEXP-CT-7c
Description :	Ajouter une recommandation.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Case Problems – Recommandation Tree.**
Entrée :	L'utilisateur sélectionne un cas problème et clique sur le bouton **Edit.** Ensuite sur **Add..**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que le pourcentage minimum soit plus grand que le pourcentage maximum (dans le premier formulaire).
Résultat(s) attendu(s) :	On attend du système qu'il refuse de passer à l'étape suivante de la procédure d'ajout de la recommandation.

ID :	COSMICEXP-CT-7d
Description :	Ajouter une recommandation.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Case Problems – Recommandation Tree.**
Entrée :	L'utilisateur sélectionne un cas problème et clique sur le bouton **Edit.** Ensuite sur **Add..**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que le pourcentage minimum égale le pourcentage maximum (dans le premier formulaire).
Résultat(s) attendu(s) :	On attend du système qu'il refuse de passer à l'étape suivante de la procédure d'ajout de la recommandation.

ID :	COSMICEXP-CT-7e
Description :	Ajouter une recommandation.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Case Problems – Recommandation Tree.**
Entrée :	L'utilisateur sélectionne un cas problème et clique sur le bouton **Edit.** Ensuite sur **Add..**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que le pourcentage minimum égale le pourcentage maximum (dans le premier formulaire) et soit également égal à 0.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour sa base de connaissance actuelle en y ajoutant la recommandation, ainsi que les pourcentages corrects, dans le domaine du cas problème choisi lors de son insertion. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-7f
Description :	Ajouter une recommandation.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Case Problems – Recommandation Tree.**
Entrée :	L'utilisateur sélectionne un cas problème et clique sur le bouton **Edit.** Ensuite sur **Add..**. Il remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que le pourcentage minimum égale -101 et/ou le pourcentage maximum égale 101 (dans le premier formulaire).
Résultat(s) attendu(s) :	On attend du système qu'il refuse de passer à l'étape suivante de la procédure d'ajout de la recommandation.

ID :	COSMICEXP-CT-7g
Description :	Ajouter une recommandation.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Case Problems – Recommandation Tree.**
Entrée :	L'utilisateur sélectionne un thème et clique sur le bouton **Edit.** Ensuite sur **Add..**. L'expert annule l'ajout d'un cas problème en cliquant sur le bouton **Cancel** (à n'importe quel moment de la procédure).
Résultat(s) attendu(s) :	On attend du système qu'il ne modifie pas la base de connaissance actuelle et que l'on se retrouve sur la page de laquelle l'utilisateur a initié l'ajout.

Description de COSMICEXP-CT-08

ID :	COSMICEXP-CT-8a
Description :	Modifier un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree**.
Entrée :	L'utilisateur sélectionne un mot clef et clique sur le bouton **Edit.** Ensuite sur **Modify..**. Il modifie alors les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour la base de connaissance actuelle en y modifiant le mot-clef, ainsi que les informations qui lui sont relatives, dans le domaine du (des) concept(s) topologique(s) dont il fait partie. La nouvelle version doit remplacer la version précédente. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-8b
Description :	Modifier un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree**.
Entrée :	L'utilisateur sélectionne un mot clef et clique sur le bouton **Edit.** Ensuite sur **Modify ...**. Il ne modifie aucun champ.
Résultat(s) attendu(s) :	On attend du système qu'il ne change en rien la base de connaissance.

ID :	COSMICEXP-CT-8c
Description :	Modifier un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree**.
Entrée :	L'utilisateur sélectionne un mot clef et clique sur le bouton **Edit.** Ensuite sur **Modify..**. Il efface l'entièreté des champs.
Résultat(s) attendu(s) :	On attend du système qu'il refuse de modifier le mot-clef et nous empêche de continuer la procédure de modification.

ID :	COSMICEXP-CT-8d
Description :	Modifier un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree**.
Entrée :	L'utilisateur sélectionne un mot clef et clique sur le bouton **Edit.** Ensuite sur **Modify..**. Il modifie les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous veillerons à ce que l'entièreté des cases de la colonne **Related Topological Concept(s)** soient décochées.
Résultat(s) attendu(s) :	On attend du système qu'il refuse de modifier le mot-clef et empêche de continuer la procédure de modification.

ID :	COSMICEXP-CT-8e
Description :	Modifier un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree**.
Entrée :	L'utilisateur sélectionne un mot clef et clique sur le bouton **Edit.** Ensuite sur **Modify..**. Il modifie les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous remplacerons certains pourcentages par les valeurs -101 ou 101.
Résultat(s) attendu(s) :	On attend du système qu'il refuse de modifier le mot-clef et nous empêche de continuer la procédure de modification.

ID :	COSMICEXP-CT-8f
Description :	Modifier un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree**.
Entrée :	L'utilisateur sélectionne un mot clef et clique sur le bouton **Edit.** Ensuite sur **Modify..**. Il modifie les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci. Nous remplacerons certains pourcentages par la valeur 0.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour la base de connaissance actuelle en y modifiant le mot-clef, ainsi que les informations qui lui sont relatives, dans le domaine du (des) concept(s) topologique(s) dont il fait partie. La nouvelle version doit remplacer la version précédente. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

ID :	COSMICEXP-CT-8g
Description :	Modifier un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree**.
Entrée :	L'utilisateur sélectionne un mot clef et clique sur le bouton **Edit.** Ensuite sur **Modify..**. L'expert annule l'ajout d'un cas problème en cliquant sur le bouton **Cancel** (à n'importe quel moment de la procédure).
Résultat(s) attendu(s) :	On attend du système qu'il ne modifie pas la base de connaissance actuelle et que l'on se retrouve sur la page de laquelle l'utilisateur a initié l'ajout.

Description de COSMICEXP-CT-09

ID :	COSMICEXP-CT-9
Description :	Modifier un concept topologique.

Ce cas de test ne sera pas développé plus avant dans le cadre de ce rapport dans la mesure où la fonctionnalité s'y rapportant n'est pas implémentée dans le système.

Description de COSMICEXP-CT-10

ID :	COSMICEXP-CT-10a
Description :	Modifier un cas problème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Case Problems - Recommandations Tree**.
Entrée :	L'utilisateur sélectionne un cas problème et clique sur le bouton **Edit.** Ensuite sur **Modify..**. Il modifie les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour la base de connaissance actuelle en y modifiant le cas problème, ainsi que les informations qui lui sont relatives, dans le domaine du concept topologique dont il fait partie. La nouvelle version doit remplacer la version précédente. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

Les variantes de ce cas de test sont identiques à celles du cas de test COSMICEXP-CT-04.

Description de COSMICEXP-CT-11

ID :	COSMICEXP-CT-11a
Description :	Modifier un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton

	Edit. Ensuite sur **Modify..**. Il modifie les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour la base de connaissance actuelle en y modifiant le thème, ainsi que les informations qui lui sont relatives, dans le domaine du concept topologique dont il fait partie. La nouvelle version doit remplacer la version précédente. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

Les variantes de ce cas de test sont identiques à celles du cas de test COSMICEXP-CT-05.

Description de COSMICEXP-CT-12

ID :	COSMICEXP-CT-12a
Description :	Modifier un fait.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un fait et clique sur le bouton **Edit.** Ensuite sur **Modify..**. On modifie les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour la base de connaissance actuelle en y modifiant le fait, ainsi que les informations qui lui sont relatives, dans le domaine du thème dont il fait partie. La nouvelle version doit remplacer la version précédente. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

Les variantes de ce cas de test sont identiques à celles du cas de test COSMICEXP-CT-06.

Description de COSMICEXP-CT-13

ID :	COSMICEXP-CT-13a
Description :	Modifier une recommandation.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Case Problems – Recommandation Tree.**
Entrée :	L'utilisateur sélectionne une recommandation et clique sur le bouton **Edit.** Ensuite sur **Modify..**. On remplit les champs du formulaire en respectant la syntaxe et la sémantique associées à chacun de ceux-ci.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour la base de connaissance actuelle en y modifiant la recommandation, ainsi que les informations qui lui sont relatives, dans le domaine du cas problème dont il fait partie. La nouvelle version doit remplacer la version précédente. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

Les variantes de ce cas de test sont identiques à celles du cas de test COSMICEXP-CT-07.

Description de COSMICEXP-CT-14

ID :	COSMICEXP-CT-14
Description :	Effacer un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Keywords – Topological Concepts – Case Problems Tree**.
Entrée :	L'utilisateur sélectionne un mot clef et clique sur le bouton **Edit.** Ensuite sur **Delete..**.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour la base de connaissance actuelle en y supprimant le mot-clef, ainsi que les informations qui lui sont relatives, dans le domaine du (des) concept(s) topologique(s) dont il fait partie. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

Description de COSMICEXP-CT-15

ID :	COSMICEXP-CT-15
Description :	Effacer un concept topologique.

Ce cas de test ne sera pas développé plus avant dans le cadre de ce rapport dans la mesure où la fonctionnalité s'y rapportant n'est pas implémentée dans le système.

Description de COSMICEXP-CT-16

ID :	COSMICEXP-CT-16
Description :	Effacer un cas problème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Case Problems - Recommandations Tree**.
Entrée :	L'utilisateur sélectionne un cas problème et clique sur le bouton **Edit.** Ensuite sur **Delete..**.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour la base de connaissance actuelle en supprimant le cas problème, ainsi que les informations qui lui sont relatives, dans le domaine du concept topologique dont il fait partie. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

Description de COSMICEXP-CT-17

ID :	COSMICEXP-CT-17
Description :	Effacer un thème.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un concept topologique et clique sur le bouton **Edit.** Ensuite sur **Delete..**.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour la base de connaissance actuelle en y supprimant le thème, ainsi que les informations qui lui sont relatives, dans le domaine du concept topologique dont il fait partie. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

Description de COSMICEXP-CT-18

ID :	COSMICEXP-CT-18
Description :	Effacer un fait.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Topological Concepts – Themes – Facts Tree**.
Entrée :	L'utilisateur sélectionne un fait et clique sur le bouton **Edit.** Ensuite sur **Delete..**.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour la base de connaissance actuelle

	en y supprimant le fait, ainsi que les informations qui lui sont relatives, dans le domaine du thème dont il fait partie. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

Description de COSMICEXP-CT-19

ID :	COSMICEXP-CT-19
Description :	Effacer une recommandation.
Pré condition :	L'utilisateur doit avoir eu accès à une page Expert, depuis l'interface administrateur ou directement en tant qu'expert. Il doit également visualiser l'arbre **Case Problems – Recommandation Tree.**
Entrée :	L'utilisateur sélectionne une recommandation et clique sur le bouton **Edit.** Ensuite sur **Delete..**.
Résultat(s) attendu(s) :	On attend du système qu'il mette à jour la base de connaissance actuelle en y supprimant la recommandation, ainsi que les informations qui lui sont relatives, dans le domaine du cas problème dont il fait partie. Une manière de le vérifier est de consulter l'exactitude des nouveaux arbres.

Spécifications des cas de test (partie mesureur)

Dans cette section, nous décrivons en détail les différents cas de test relatifs aux fonctionnalités du mesureur. Nous donnons également les différentes variantes basées sur un même test.

Description de COSMICMES-CT-1

ID :	COSMICMES-CT-1a
Description :	Consulter la définition d'un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Mesureur, depuis l'interface administrateur, l'interface expert ou directement en tant que mesureur.
Entrée :	L'utilisateur sélectionne un mot-clef et clique sur le bouton **Definition** (dans le premier cadre). On veillera à bien se trouver dans la première étape de la mesure.
Résultat(s) attendu(s) :	On attend du système qu'il affiche la définition du mot-clef sélectionné.

ID :	COSMICMES-CT-1b
Description :	Consulter la définition d'un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Mesureur, depuis l'interface administrateur, l'interface expert ou directement en tant que mesureur.
Entrée :	A n'importe qu'elle étape de la mesure, l'utilisateur sélectionne un mot-clef et clique sur le bouton **Definition** (dans le premier cadre).
Résultat(s) attendu(s) :	On attend du système qu'il affiche la définition du mot-clef sélectionné.

Description de COSMICMES-CT-2

ID :	COSMICMES-CT-2
Description :	Recherche des concepts topologiques en rapport à un mot-clef.
Pré condition :	L'utilisateur doit avoir eu accès à une page Mesureur, depuis l'interface administrateur, l'interface expert ou directement en tant que mesureur.
Entrée :	L'utilisateur sélectionne un mot-clef et clique sur le bouton **Search** (dans le premier cadre). Nous veillerons à ce que chaque mot-clef soit testé.
Résultat(s) attendu(s) :	On attend du système qu'il affiche, dans le deuxième cadre, l'ensemble des concepts topologiques liés au mot-clef sélectionné et en concordance avec la base de connaissance actuelle.

Description de COSMICMES-CT-3

ID :	COSMICMES-CT-3a
Description :	Recherche par index.
Pré condition :	L'utilisateur doit avoir eu accès à une page Mesureur, depuis l'interface administrateur, l'interface expert ou directement en tant que mesureur.
Entrée :	L'utilisateur clique sur le bouton **Search by Index** (dans le premier cadre). Il entre un mot-clef et clique sur le bouton **Search**. Ensuite, il sélectionne un résultat et clique sur le bouton **Search**.
Résultat(s) attendu(s) :	On attend du système qu'il affiche, dans le deuxième cadre, l'ensemble

	des concepts topologiques liés au mot-clef sélectionné et en concordance avec la base de connaissance actuelle.

ID :	COSMICMES-CT-3b
Description :	Recherche par index.
Pré condition :	L'utilisateur doit avoir eu accès à une page Mesureur, depuis l'interface administrateur, l'interface expert ou directement en tant que mesureur.
Entrée :	L'utilisateur clique sur le bouton **Search by Index** (dans le premier cadre). Il n'entre aucun mot-clef et clique sur le bouton **Search**.
Résultat(s) attendu(s) :	On attend du système qu'il affiche un message d'erreur avertissant du manque et qu'il permette une nouvelle recherche par index.

ID :	COSMICMES-CT-3c
Description :	Recherche par index.
Pré condition :	L'utilisateur doit avoir eu accès à une page Mesureur, depuis l'interface administrateur, l'interface expert ou directement en tant que mesureur.
Entrée :	L'utilisateur clique sur le bouton **Search by Index** (dans le premier cadre). Il entre un mot-clef et clique sur le bouton **Search**. Ensuite, il ne sélectionne pas de résultat et clique sur le bouton **Search**.
Résultat(s) attendu(s) :	On attend du système qu'il affiche un message d'erreur avertissant de la non sélection et qu'il permette une nouvelle recherche par index.

ID :	COSMICMES-CT-3d
Description :	Recherche par index.
Pré condition :	L'utilisateur doit avoir eu accès à une page Mesureur, depuis l'interface administrateur, l'interface expert ou directement en tant que mesureur.
Entrée :	L'utilisateur clique sur le bouton **Search by Index** (dans le premier cadre). Il entre un mot-clef inexistant et clique sur le bouton **Search**.
Résultat(s) attendu(s) :	On attend du système qu'il affiche un message d'erreur avertissant du manque et qu'il permette une nouvelle recherche par index.

Description de COSMICMES-CT-4

ID :	COSMICMES-CT-4
Description :	Consulter la définition des concepts topologiques/cas problèmes/thèmes/recommandations.
Pré condition :	L'utilisateur doit avoir eu accès à une page Mesureur, depuis l'interface administrateur, l'interface expert ou directement en tant que mesureur. Il se trouve dans n'importe quelle étape de la mesure.
Entrée :	L'utilisateur clique sur le lien hypertexte représentant le concept topologique/cas problème/thème/recommandation dont il désire connaître la définition.
Résultat(s) attendu(s) :	On attend du système qu'il affiche une définition correcte par rapport au concept sélectionné.

Description de COSMICMES-CT-5

ID :	COSMICMES-CT-5
Description :	Obtenir une recommandation.
Pré condition :	L'utilisateur doit avoir eu accès à une page Mesureur, depuis l'interface administrateur, l'interface expert ou directement en tant que mesureur.
Entrée :	L'utilisateur effectue une mesure et rempli les champs adéquats en respectant la syntaxe et la sémantique de ces derniers.
Résultat(s) attendu(s) :	On attend du système qu'il affiche une recommandation adéquate aux entrées que nous lui aurons donné.

Spécifications des cas de test (partie commune)

Dans cette section, nous décrivons en détail les différents cas de test relatifs aux fonctionnalités commune à chaque type d'utilisateur. Nous donnons également les différentes variantes basées sur un même test.

Description de COSMICCOM-CT-1

ID :	COSMICCOM-CT-1a
Description :	Se connecter.
Pré condition :	Le système est prêt. L'utilisateur dispose d'un compte encore valide sur le système (du point de vue de la date).
Entrée :	L'utilisateur entre un nom d'utilisateur et un mot de passe valides. Il clique sur le bouton **Ok** pour se connecter.
Résultat(s) attendu(s) :	On attend du système qu'il affiche la page de démarrage en rapport au niveau d'accréditation de l'utilisateur (administrateur, expert ou mesureur).

ID :	COSMICCOM-CT-1b
Description :	Se connecter.
Pré condition :	Le système est prêt. L'utilisateur ne dispose plus d'un compte encore valide sur le système (du point de vue de la date).
Entrée :	L'utilisateur entre un nom d'utilisateur et un mot de passe valides. Il clique sur le bouton **Ok** pour se connecter.
Résultat(s) attendu(s) :	On attend du système qu'il avertisse l'utilisateur que son compte est dépassé et qu'il n'est plus autorisé à se connecter au système.

ID :	COSMICCOM-CT-1c
Description :	Se connecter.
Pré condition :	Le système est prêt.
Entrée :	L'utilisateur entre un nom d'utilisateur et/ou un mot de passe invalide(s). Il clique sur le bouton **Ok** pour se connecter.
Résultat(s) attendu(s) :	On attend du système qu'il avertisse l'utilisateur qu'une des deux ou les deux données sont invalides et qu'il demande à nouveau de les réintroduire.

Description de COSMICCOM-CT-2

ID :	COSMICCOM-CT-2
Description :	Se déconnecter.
Pré condition :	L'utilisateur doit s'être connecté en tant qu'administrateur ou expert.
Entrée :	L'utilisateur clique sur le bouton **Logout** de la page (en rapport exclusivement à l'interface expert ou administrateur).
Résultat(s) attendu(s) :	On attend du système qu'il se déconnecte du système et revienne au point de départ, c'est-à-dire à la page où l'utilisateur est capable de se connecter au système.

Seconde partie : Rapport de Tests

Remarques préliminaires

Dans cette partie, nous présentons un rapport des tests qui ont été effectués sur le système **CosmicXpert**. Les rapports présentés dans la section suivante sont une version simplifiée de la grille de tests effective (qui sera fournie en annexe) comprenant tout le détail des tests mais plus difficilement lisible. Pour chaque ligne de ces tableaux, le code du test (ou de sa variante) est indiqué. De même, dans la colonne intitulée **Testé**, nous indiquons, par **O**ui ou **N**on, si le test à bien été mené ou pas. Il y a également la présence d'une valeur qui est une fraction représentant le nombre de conditions remplies sur le nombre de conditions attendues. Dans certains cas, les tests n'ont pas été effectués en raison d'une implémentation de la partie relative (et cette absence est prise en compte dans les ratios de fin de section). Dans d'autres cas, nous avons choisi de ne pas mener les tests en raison d'une redondance flagrante qui aurait faussé nos calculs. Cette subtile différence sera indiquée dans la grille de tests détaillée.

Nous avons répartis les cas par type d'utilisateur et au terme de chaque tableau se trouve un calcul en pourcentage de la complétude de ladite partie. De même, nous indiquons quelle quantité de fautes *critiques* ou *non-critiques* est présente.

Par faute critique, nous entendons des fautes qui empêche le système de donner un résultat correct tandis que par faute non-critique, nous signifions des fautes qui mènent à un fonctionnement correct mais accompagné d'un effet non prévu.

Rapports de Tests (version synthétique)

Résultats des tests de la partie Administrateur

ID	*Description*	*Testé*	*Résultat*
COSMICADM-CT-1a	Ajout d'un utilisateur à la liste des utilisateurs du système.	O	2.5 / 3
COSMICADM-CT-1b	Ajout d'un utilisateur à la liste des utilisateurs du système.	O	1 / 1
COSMICADM-CT-1c	Ajout d'un utilisateur à la liste des utilisateurs du système.	O	1 / 1
COSMICADM-CT-1d	Ajout d'un utilisateur à la liste des utilisateurs du système.	O	2 / 2
COSMICADM-CT-1e	Ajout d'un utilisateur à la liste des utilisateurs du système.	O	2 / 2
COSMICADM-CT-1f	Ajout d'un utilisateur à la liste des utilisateurs du système.	O	0 / 1
COSMICADM-CT-2	Suppression d'un utilisateur.	O	2 / 2
COSMICADM-CT-3a	Changer le mot de passe d'un utilisateur.	**O**	1 / 1
COSMICADM-CT-3b	Changer le mot de passe d'un utilisateur.	**O**	1 / 1
COSMICADM-CT-3c	Changer le mot de passe d'un utilisateur.	**O**	1 / 1
COSMICADM-CT-4a	Changer la date d'expiration d'un compte utilisateur.	**O**	1 / 1
COSMICADM-CT-4b	Changer la date d'expiration d'un compte utilisateur.	**O**	0.5 / 1
COSMICADM-CT-4c	Changer la date d'expiration d'un compte utilisateur.	**O**	0.5 / 1
COSMICADM-CT-4d	Changer la date d'expiration d'un compte utilisateur.	**O**	0 / 1
COSMICADM-CT-4e	Changer la date d'expiration d'un compte utilisateur.	**O**	0 / 1
COSMICADM-CT-05	Changer le statut d'un utilisateur.	**O**	2 / 2
COSMICADM-CT-06	Restaurer la base de connaissance.	**O**	0 / 1
COSMICADM-CT-7a	Interaction avec l'interface Mesureur.	**O**	1 / 1
COSMICADM-CT-7b	Interaction avec l'interface Mesureur.	**O**	0 / 1
COSMICADM-CT-7c	Interaction avec l'interface Mesureur.	**O**	0 / 1

COSMICADM-CT-08	Interaction avec l'interface Expert.	**O**	1 / 1

Pour cette partie, nous évaluons le système correct à **72,2 %** .

Nombre d'erreurs non-critiques : 3

Nombre d'erreurs critiques : 6

Résultats des tests de la partie Expert

ID	*Description*	*Testé*	*Résultat*
COSMICEXP-CT-01	Consulter un arbre.	**O**	1 / 1
COSMICEXP-CT-2a	Ajouter un mot-clef.	**O**	1 / 1
COSMICEXP-CT-2b	Ajouter un mot-clef.	**O**	1 / 1
COSMICEXP-CT-2c	Ajouter un mot-clef.	**O**	1 / 1
COSMICEXP-CT-2d	Ajouter un mot-clef.	**O**	1 / 1
COSMICEXP-CT-2e	Ajouter un mot-clef.	**O**	1 / 1
COSMICEXP-CT-2f	Ajouter un mot-clef.	**O**	1 / 1
COSMICEXP-CT-3a	Ajouter un concept topologique.	**O**	1 / 1
COSMICEXP-CT-3b	Ajouter un concept topologique.	**O**	1 / 1
COSMICEXP-CT-3c	Ajouter un concept topologique.	**O**	1 / 1
COSMICEXP-CT-3d	Ajouter un concept topologique.	**O**	1 / 1
COSMICEXP-CT-3e	Ajouter un concept topologique.	**O**	0 / 1
COSMICEXP-CT-4a	Ajouter un cas problème.	**O**	0 / 1
COSMICEXP-CT-4b	Ajouter un cas problème.	**O**	1 / 1
COSMICEXP-CT-4c	Ajouter un cas problème.	**O**	1 / 1
COSMICEXP-CT-4d	Ajouter un cas problème.	**O**	0 / 1
COSMICEXP-CT-4e	Ajouter un cas problème.	**O**	0 / 1
COSMICEXP-CT-4f	Ajouter un cas problème.	**O**	1 / 1
COSMICEXP-CT-4g	Ajouter un cas problème.	**O**	1 / 2
COSMICEXP-CT-4h	Ajouter un cas problème.	**O**	1 / 1
COSMICEXP-CT-4i	Ajouter un cas problème.	**O**	1 / 1
COSMICEXP-CT-5a	Ajouter un thème.	**O**	0.5 / 1
COSMICEXP-CT-5b	Ajouter un thème.	**O**	1 / 1

COSMICEXP-CT-5c	Ajouter un thème.	**O**	1 / 1
COSMICEXP-CT-5d	Ajouter un thème.	**O**	1 / 1
COSMICEXP-CT-5e	Ajouter un thème.	**O**	1 / 1
COSMICEXP-CT-5f	Ajouter un thème.	**O**	0.5 / 1
COSMICEXP-CT-5g	Ajouter un thème.	**O**	0.5 / 1
COSMICEXP-CT-5h	Ajouter un thème.	**O**	0.5 / 1
COSMICEXP-CT-5i	Ajouter un thème.	**O**	0.5 / 1
COSMICEXP-CT-5j	Ajouter un thème.	**O**	0.5 / 1
COSMICEXP-CT-5k	Ajouter un thème.	**O**	1 / 1
COSMICEXP-CT-5l	Ajouter un thème.	**O**	0 / 1
COSMICEXP-CT-6a	Ajouter un fait.	**O**	0.5 / 1
COSMICEXP-CT-6b	Ajouter un fait.	**O**	0.5 / 1
COSMICEXP-CT-6c	Ajouter un fait.	**O**	0.5 / 1
COSMICEXP-CT-6d	Ajouter un fait.	**O**	0.5 / 1
COSMICEXP-CT-6e	Ajouter un fait.	**O**	0.5 / 1
COSMICEXP-CT-6f	Ajouter un fait.	**O**	0 / 1
COSMICEXP-CT-7a	Ajouter une recommandation.	**O**	1 / 1
COSMICEXP-CT-7b	Ajouter une recommandation.	**O**	1 / 1
COSMICEXP-CT-7c	Ajouter une recommandation.	**O**	1 / 1
COSMICEXP-CT-7d	Ajouter une recommandation.	**O**	1 / 1
COSMICEXP-CT-7e	Ajouter une recommandation.	**O**	1 / 1
COSMICEXP-CT-7f	Ajouter une recommandation.	**O**	1 / 1
COSMICEXP-CT-7g	Ajouter une recommandation.	**O**	1 / 1
COSMICEXP-CT-8a	Modifier un mot-clef.	**O**	1 / 1
COSMICEXP-CT-8b	Modifier un mot-clef.	**O**	1 / 1
COSMICEXP-CT-8c	Modifier un mot-clef.	**O**	1 / 1
COSMICEXP-CT-8d	Modifier un mot-clef.	**O**	1 / 1
COSMICEXP-CT-8e	Modifier un mot-clef.	**O**	1 / 1
COSMICEXP-CT-8f	Modifier un mot-clef.	**O**	1 / 1
COSMICEXP-CT-8g	Modifier un mot-clef.	**O**	1 / 1
COSMICEXP-CT-09	Modifier un concept topologique.	**N**	
COSMICEXP-CT-10a	Modifier un cas problème.	**O**	0 / 1
COSMICEXP-CT-10b	Modifier un cas problème.	**N**	
COSMICEXP-CT-10c	Modifier un cas problème.	**N**	
COSMICEXP-CT-10d	Modifier un cas problème.	**N**	
COSMICEXP-CT-10e	Modifier un cas problème.	**N**	
COSMICEXP-CT-10f	Modifier un cas problème.	**N**	
COSMICEXP-CT-10g	Modifier un cas problème.	**N**	
COSMICEXP-CT-10h	Modifier un cas problème.	**N**	

COSMICEXP-CT-10i	Modifier un cas problème.	**N**	
COSMICEXP-CT-11	Modifier un thème.	O	0 / 1
COSMICEXP-CT-11b	Modifier un thème.	**N**	
COSMICEXP-CT-11c	Modifier un thème.	**N**	
COSMICEXP-CT-11d	Modifier un thème.	**N**	
COSMICEXP-CT-11e	Modifier un thème.	**N**	
COSMICEXP-CT-11f	Modifier un thème.	**N**	
COSMICEXP-CT-11g	Modifier un thème.	**N**	
COSMICEXP-CT-11h	Modifier un thème.	**N**	
COSMICEXP-CT-11i	Modifier un thème.	**N**	
COSMICEXP-CT-11j	Modifier un thème.	**N**	
COSMICEXP-CT-11k	Modifier un thème.	**N**	
COSMICEXP-CT-	Modifier un thème.	**N**	

11l			
COSMICEXP-CT-12a	Modifier un fait.	**O**	0.5 / 1
COSMICEXP-CT-12b	Modifier un fait.	**N**	
COSMICEXP-CT-12c	Modifier un fait.	**N**	
COSMICEXP-CT-12d	Modifier un fait.	**N**	
COSMICEXP-CT-12e	Modifier un fait.	**N**	
COSMICEXP-CT-12f	Modifier un fait.	**N**	
COSMICEXP-CT-13a	Modifier une recommandation.	**O**	1 / 1
COSMICEXP-CT-13b	Modifier une recommandation.	**N**	
COSMICEXP-CT-13c	Modifier une recommandation.	**N**	
COSMICEXP-CT-13d	Modifier une recommandation.	**N**	
COSMICEXP-CT-13e	Modifier une recommandation.	**N**	
COSMICEXP-CT-13f	Modifier une recommandation.	**N**	
COSMICEXP-CT-13g	Modifier une recommandation.	**N**	
COSMICEXP-CT-14	Effacer un mot-clef.	**O**	0 / 1
COSMICEXP-CT-15	Effacer un concept topologique.	**N**	
COSMICEXP-CT-16	Effacer un cas problème.	**O**	1 / 1
COSMICEXP-CT-17	Effacer un thème.	**O**	0.5 / 1
COSMICEXP-CT-18	Effacer un fait.	**O**	0 / 1
COSMICEXP-CT-19	Effacer une recommandation.	**O**	1 / 1

Pour cette partie, nous évaluons le système correct à **70,6 %** .

Nombre d’erreurs non-critiques : 13

Nombre d’erreurs critiques : 11

Résultats des tests de la partie Mesureur

ID	*Description*	*Testé*	*Résultat*
COSMICMES-CT-1a	Consulter la définition d'un mot-clé.	**O**	1 / 1
COSMICMES-CT-1b	Consulter la définition d'un mot-clé.	**O**	1 / 1
COSMICMES-CT-2	Recherche des concepts topologiques en rapport à un mot-clef.	**O**	1 / 1
COSMICMES-CT-3a	Recherche par index.	**O**	1 / 1
COSMICMES-CT-3b	Recherche par index.	**O**	1 / 1
COSMICMES-CT-3c	Recherche par index.	**O**	0 / 1
COSMICMES-CT-3d	Recherche par index.	**O**	1 / 1
COSMICMES-CT-4	Consulter la définition des concepts topologiques / cas problèmes / thèmes / recommandations.	**O**	1 / 1
COSMICMES-CT-5	Obtenir une recommandation.	**O**	1 / 1

Pour cette partie, nous évaluons le système correct à **88,8 %** .

Nombre d'erreurs non-critiques : 0

Nombre d'erreurs critiques : 1

Résultats des tests de la partie Commune

ID	*Description*	*Testé*	*Résultat*
COSMICCOM-CT-1a	Se connecter.	**O**	1 / 1
COSMICCOM-CT-1b	Se connecter.	**O**	0.5 / 1
COSMICCOM-CT-1c	Se connecter.	**O**	1 / 1

COSMICCOM-CT-2	Se déconnecter.	**O**	1 / 1

Pour cette partie, nous évaluons le système correct à **87,5 %** .

Nombre d'erreurs non-critiques : 1

Nombre d'erreurs critiques : 0

Remarques finales : les erreurs redondantes.

Si les cas de tests développés ci-dessus représentent un bon balayage des exigences fonctionnelles imposées à **CosmicXpert**, il ne couvre pas les exigences non-fonctionnelles. Nous terminerons en listant les erreurs de ce type; rencontrées durant notre phase de tests.

Notons donc qu'il subsiste :

- ✓ Des erreurs liées à l'utilisation des verrous.
- ✓ Des erreurs de *design* liées à l'interface homme-machine.
- ✓ Des erreurs liées à l'utilisation des *pop-up menus*, bloqués par certains systèmes d'exploitation et/ou pares-feu
- ✓ Un problème d'affichage de données pour certaines images.

Rapports de Tests (version exhaustive)

Voir pages suivantes.

ID	Description	Entrées	Résultats attendus	Résultats obtenus	Testé	Commentaires
COSMICADM-CT-1a	Ajouter un utilisateur à la liste des utilisateurs du système.	Login : vincent Password : coucou Status :expert Date X : 2004-09-23	✓ utilisateur enregistré. ✓ accès au système. ✓ caractéristiques conformes : 1. date d'expiration. 2. type d'utilisateur.	✓ utilisateur enregistré. ✓ accès au système. ✓ caractéristiques conformes : 1. date d'expiration. 2. type d'utilisateur.	OUI	La fenêtre ne se rafraîchit pas après l'introduction du nouvel utilisateur. L'administrateur doit se reconnecter au système pour voir les changement.
COSMICADM-CT-1b	Ajouter un utilisateur à la liste des utilisateurs du système.	Login : idrissa Password : idrissa Status : measurer Date X : Ø	✓ refus de l'enregistrement.	✓ refus de l'enregistrement.	OUI	
COSMICADM-CT-1c	Ajouter un utilisateur à la liste des utilisateurs du système.	Login : idrissa Password : idrissa Status :Expert Date X : Ø	✓ refus de l'enregistrement.	✓ refus de l'enregistrement.	OUI	
COSMICADM-CT-1d	Ajouter un utilisateur à la liste des utilisateurs du système.	Login : jacques Password : coucou Status : admin Date X : 2004-08-12	✓ utilisateur enregistré. ✓ pas d'accès au système.	✓ utilisateur enregistré. ✓ pas d'accès au système.	OUI	
COSMICADM-CT-1e	Ajouter un utilisateur à la liste des utilisateurs du système.	Login : barth Password : coucou Status : expert Date X : 2004-09-22	✓ utilisateur enregistré. ✓ pas d'accès au système.	✓ utilisateur enregistré. ✓ pas d'accès au système.	OUI	

COSMICADM-CT-1f	Ajouter un utilisateur à la liste des utilisateurs du système.	Login : renson Password : coucou Status : measurer Date X : 2004_12_23	✓ refus de l'enregistrement.	✓ refus de l'enregistrement.	OUI	Le système accepte d'enregistrer un utilisateur dont la date d'expiration est syntaxiquement incorrecte
COSMICADM-CT-2	Supprimer un utilisateur de la liste des utilisateurs du système.	Clic sur **Delete User** de la ligne *vincent*	✓ utilisateur effacé. ✓ plus d'accès au système.	✓ utilisateur effacé. ✓ plus d'accès au système.	OUI	Le rafraîchissement s'effectue correctement contrairement au cas de l'ajout.
COSMICADM-CT-3a	Changer le mot de passe d'un utilisateur.	Sélecion : idrissa password : coucou	✓ mise à jour du password	✓ mise à jour du password	OUI	Le compte utilisateur est bien mis à jour et l'ancien mot de passe n'est plus actif.
COSMICADM-CT-3b	Changer le mot de passe d'un utilisateur.	Sélecion : idrissa password : coucou	✓ aucun changement	✓ aucun changement	OUI	
COSMICADM-CT-3c	Changer le mot de passe d'un utilisateur.	Sélecion : idrissa password : Ø	✓ refus de l'opération	✓ refus de l'opération	OUI	Le refus s'est produite comme attendu et la notification aussi.

COSMICADM-CT-4a	Changer la date d'expiration d'un compte utilisateur.	Sélecion : barth date : 2004-09-23	✓ mise à jour de la date	✓ mise à jour de la date ❶	OUI	
COSMICADM-CT-4b	Changer la date d'expiration d'un compte utilisateur.	Sélecion : barth date : 2004-09-10	✓ refus de l'opération	✓ refus de l'opération ❶	OUI	Le système accepte cette date aberrante mais désactive néanmoins le compte de l'utilisateur. Il ne s'agit donc pas d'un problème critique au point de vue du fonctionnement du système mais d'une incohérence d'un point de vue logique.
COSMICADM-CT-4c	Changer la date d'expiration d'un compte utilisateur.	Sélecion : alain date : 2004-09-22	✓ refus de l'opération	✓ refus de l'opération ❶	OUI	Le système accepte cette date aberrante mais désactive néanmoins le compte de l'utilisateur. Il ne s'agit donc pas d'un problème critique au point de vue du fonctionnement du système mais d'une incohérence d'un point de vue logique.
COSMICADM-CT-4d	Changer la date d'expiration d'un compte utilisateur.	Sélecion : log510 date : 2004-12+05	✓ refus de l'opération	✓ refus de l'opération ❶	OUI	Le système accepte cette date aberrante et désactive le compte de l'utilisateur.
COSMICADM-CT-4e	Changer la date d'expiration d'un compte utilisateur.	Sélecion : alain date : Ø	✓ annulation de la contrainte	✓ annulation de la contrainte ❶	OUI	Lorsqu'on efface la date afin d'enlever la contrainte, la modification n'est pas prise en compte et l'ancienne contrainte reste active.

COSMICADM-CT-5	Changer le statut d'un utilisateur.	Sélecion : user1 status : expert	✓ mise à jour ✓ caractéristiques conformes : 1. type d'utilisateur.	✓ mise à jour ✓ caractéristiques conformes : 1. type d'utilisateur.	OUI	
COSMICADM-CT-6	Restaurer la base de connaissance.	Clic sur bouton : **Restore Knowledge Base.**	✓ Restauration de la base de connaissance.	✓ Restauration de la base de connaissance.	OUI	Aucune modification n'a été annulée
COSMICADM-CT-7a	Interaction avec l'interface Mesureur.	Clic sur lien : **Measurer Page**	✓ Navigation entre les pages.	✓ Navigation entre les pages.	OUI	
COSMICADM-CT-7b	Interaction avec l'interface Mesureur.	Clic sur lien : **Measurer Page** Clic sur le lien : l**oggout**	✓ Navigation entre les pages.	✓ Navigation entre les pages.	OUI	L'interface mesureur reste ouverte alors que la session se ferme quelques le sessions se ferment quelques minutes plus tard.
COSMICADM-CT-7c	Interaction avec l'interface Mesureur.	Clic sur lien : **Measurer Page** Clic sur le bouton : **X** en haut à droite.	✓ Navigation entre les pages.	✓ Navigation entre les pages.	OUI	L'administrateur possède encore une page mesureur alors que sa session est sensée être close.
COSMICADM-CT-8	Interaction avec l'interface Expert.	Clic sur lien : **Expert Page** Clic sur le lien : **Aministrator Page**.	✓ Navigation entre les pages.	✓ Navigation entre les pages.	OUI	La navigation entre page Expert et Administrateur se fait correctement.

Légende

Fonctionnement normal

Fonctionnement ok mais anomalie mineure

Fonctionnement Ko

Insuffisance de l’Interface Homme-Machine

❶ : La nouvelle date n’est active que lorsqu’on clique sur la fenêtre, hors de la zone de texte correspondante.

❷ : Dans les cas problèmes, nous avons remarqué une problème d’affichage d’une image pour le cas ***Change Customer Display Customer Data***

Recommandations :

Interface :

❶ : Ajout d’un bouton de validation pour la date dans le panneau de l’administrateur.

ID	Description	Entrées	Résultats attendus	Résultats obtenus	Testé	Commentaires
COSMICMES-CT-1a	Consulter la définition d'un mot-clef.	Sélection d'un mot-clé Clic sur **Définition**	✓ Affichage de la définition.	✓ Affichage de la définition.	OUI	Fonctionne pour chaque mot-clé.
COSMICMES-CT-1b	Consulter la définition d'un mot-clef.	Sélection « exit » Clic sur **Search** Sélection CT « exit » Sélection CP « How to identify an exit » Clic sur **Définition**	✓ Affichage de la définition.	✓ Affichage de la définition.	OUI	
COSMICMES-CT-2	Recherche des concepts topologiques en rapport à un mot-clef.	Sélection d'un mot-clé Clic sur **Search**	✓ Affichage des concepts topologiques.	✓ Affichage des concepts topologiques.	OUI	Chaque mot-clé a été testé et validé par vérification dans l'arbre adéquat.
COSMICMES-CT-3a	Recherche par index.	Index : Entry Clic sur **Search** Sélection : Entry Clic sur **Search**	✓ Affichage du résultat	✓ Affichage du résultat	OUI	
COSMICMES-CT-3b	Recherche par index.	Index : Ø	✓ Refus et notification.	✓ Refus et notification.	OUI	

		Clic sur **Search**				
COSMICMES-CT-3c	Recherche par index.	Index : Polling Clic sur **Search** Sélection : Ø Clic sur **Search**	✓ Message d'erreur	✓ Message d'erreur	OUI	Le système n'avertit pas de la non sélection d'un mot-clé associée à l'index. Il efface, de plus, le premier cadre.
COSMICMES-CT-3d	Recherche par index.	Index : boolean Clic sur **Search**	✓ Message d'erreur	✓ Message d'erreur	OUI	Le système spécifie bien que l'entrée est inexistante.
COSMICMES-CT-4	Consulter la définition des concepts topologiques/cas problèmes/thèmes/ recommandations.	Sélection d'un concept topologique (hypertexte)	✓ affichage des concepts topologiques.	✓ affichage des concepts topologiques. ❷	OUI	Chaque concept a été testé et validé.
COSMICMES-CT-5	Obtenir une recommandation.	Keyword : write Clic sur **Search** Sélection CT : write Sélection CP: Change Customer Store Item Data Réponses : YYYY	✓ affichage de la recommandation	✓ affichage de la recommandation	OUI	La recommandation s'affiche bien. Si la base de connaissance est cohérente, alors le résultat obtenu est correct.

ID	Description	Entrées	Résultats attendus	Résultats obtenus	Testé	Commentaires
COSMICCOM-CT-1a	Se connecter.	Login : jmd1 Password : jmd1	✓ Utilisateur connecté.	✓ Utilisateur connecté.	OUI	
COSMICCOM-CT-1b	Se connecter.	Login : barth Password : coucou	✓ Avertissement d'expiration.	✓ Avertissement d'expiration.	OUI	Le compte est bien bloqué mais aucune notification ne précise que cela est du à une expiration de date.
COSMICCOM-CT-1c	Se connecter.	Login : jmd1 Password : miti	✓ Avertissement erreur syntaxique.	✓ Avertissement erreur syntaxique.	OUI	
COSMICCOM-CT-2	Se déconnecter.		✓ Utilisateur déconnecté.	✓	OUI	

ID	Description	Entrées	Résultats attendus	Résultats obtenus	Testé	Commentaires
COSMICEXP-CT-1	Consulter un arbre.	Clic sur **Xpert Trees** Sélection d'un arbre	✓ affichage de l'arbre demandé.	✓ affichage de l'arbre demandé.	OUI	Testé pour chacun des trois arbres.
COSMICEXP-CT-2a	Ajouter un mot-clef.	Keyword: Miooo Description: Miooo RTC(jusque Exit): 12% RTC (Exit) non coché RTC restant : 23%	✓ ajout du mot clé	✓ ajout du mot clé	OUI	
COSMICEXP-CT-2b	Ajouter un mot-clef.	Keyword: tyrol Description: tyrol Data Group : 101% Entry : 23% Exit : 23%	✓ refus de l'enregistrement	✓ refus de l'enregistrement	OUI	Un autre test avec Data Group à – 101% a été effectué avec le même résultat attendu et obtenu.
COSMICEXP-CT-2c	Ajouter un mot-clef.	Keyword: tyrol Description: tyrol Data Group : 0% Entry : 23% Exit : 23%	✓ ajout du mot clé	✓ ajout du mot clé	OUI	
COSMICEXP-CT-2d	Ajouter un mot-clef.	Keyword: tyrol Description: tyrol Trigger Event : 34%	✓ refus de l'enregistrement	✓ refus de l'enregistrement	OUI	

COSMICEXP-CT-2e	Ajouter un mot-clef.	Aucun champ rempli	✓ refus de l'enregistrement	✓ refus de l'enregistrement	**OUI**	
COSMICEXP-CT-2f	Ajouter un mot-clef.	Clic sur **Cancel**	✓ aucune modification de la base et retour à la page précédente	✓ aucune modification de la base et retour à la page précédente	**OUI**	
COSMICEXP-CT-3a	Ajouter un concept topologique.	Un concept topologique inventé, non encore existant.	✓ ajout du concept topologique	✓ ajout du concept topologique	**OUI**	
COSMICEXP-CT-3b	Ajouter un concept topologique.	Un concept topologique existant.	✓ refus de l'enregistrement	✓ refus de l'enregistrement	**OUI**	
COSMICEXP-CT-3c	Ajouter un concept topologique.	Un concept topologique non existant avec des champs obligatoires omis.	✓ refus du passage à l'étape suivante	✓ refus du passage à l'étape suivante	**OUI**	Lors du passage à chaque étape, une vérification des champs obligatoires est effectuée. Un avertissement apparaît si le besoin s'en fait sentir.
COSMICEXP-CT-3d	Ajouter un concept topologique.	Un concept topologique non existant. Percentage : 101%	✓ refus de l'enregistrement	✓ refus de l'enregistrement	**OUI**	A chaque étape, il y a une vérification et un avertissement lorsqu'il y a une valeur erronée. Un autre test avec Percentage à – 101% a été effectué avec le même résultat attendu et obtenu.
COSMICEXP-CT-3e	Ajouter un concept topologique.	Clic sur **Cancel** à chaque étape	✓ retour à l'étape précédente	✓ retour à l'étape précédente	**OUI**	Un stack d'erreur apparaît après avoir cliqué sur Cancel à chaque étape, bien que la base de connaissance ne soit pas modifiée.

COSMICEXP-CT-4a	Ajouter un cas problème.	Cas problème non existant.	✓ ajout du cas problème	✓ ajout du cas problème	**OUI**	Un stack d'erreur apparaît lors du choix d'un ensemble de recommandations. La base de connaissance n'a pas été modifiée.
COSMICEXP-CT-4b	Ajouter un cas problème.	Aucun champ rempli	✓ refus du passage à l'étape suivante	✓ refus du passage à l'étape suivante	**OUI**	
COSMICEXP-CT-4c	Ajouter un cas problème.	Aucun concept topologique n'a été sélectionné précédemment à la tentative d'ajout.	✓ refus d'accès à la procédure d'ajout	✓ refus d'accès à la procédure d'ajout	**OUI**	
COSMICEXP-CT-4d	Ajouter un cas problème.	Cas problème existant et absent du domaine du concept topologique sélectionné.	✓ ajout du cas problème	✓ ajout du cas problème	**OUI**	La base de connaissance n'a pas été modifiée. Comme nous n'arrivons pas à la fin de la procédure, on ne peut pas savoir si la base de connaissance aurait été modifiée.
COSMICEXP-CT-4e	Ajouter un cas problème.	Cas problème existant et présent dans le domaine du concept topologique sélectionné.	✓ refus de l'enregistrement	✓ refus de l'enregistrement	**OUI**	La base de connaissance n'a pas été modifiée. Comme nous n'arrivons pas à la fin de la procédure, on ne peut pas savoir si la base de connaissance aurait été modifiée.
COSMICEXP-CT-4f	Ajouter un cas problème.	Cas problème non existant et absent du domaine du	✓ refus du passage à l'étape suivante	✓ refus du passage à l'étape suivante	**OUI**	Un autre test avec Percentage à – 101% a été effectué avec le même résultat attendu et obtenu.

		concept topologique sélectionné. Percentage : 101%				
COSMICEXP-CT-4g	Ajouter un cas problème.	Cas problème non existant et absent du domaine du concept topologique sélectionné. Percentage : 0 %	✓ accepter le 0 ✓ ajout du cas problème	✓ accepter le 0 ✓ ajout du cas problème	**OUI**	Le passage entre les différentes étapes avec le 0 fonctionne bien mais la procédure d'ajout fait face à un stack d'erreur lors du choix de l'ensemble des recommandations.
COSMICEXP-CT-4h	Ajouter un cas problème.	Cas problème existant et absent du domaine du concept topologique sélectionné. Pas de Related Case Study en rapport avec.	✓ refus du passage à l'étape suivante	✓ refus du passage à l'étape suivante	**OUI**	
COSMICEXP-CT-4i	Ajouter un cas problème.	Clic sur **Cancel** à chaque étape	✓ retour à l'étape précédente	✓ retour à l'étape précédente	**OUI**	
COSMICEXP-CT-5a	Ajouter un thème.	Sélection : Boundary Percentage : 54% Question :	✓ mise à jour de la base	✓ mise à jour de la base	**OUI**	Un stack d'erreur apparaît après avoir actionner le lien de retour, bien que la base soit mise à jour.

		« Identification of Processes » Yes : 45% No : 57%				
COSMICEXP-CT-5b	Ajouter un thème.	Sélection : Data Group Tous les champs sont vides.	✓ refus de l'opération	✓ refus de l'opération	**OUI**	
COSMICEXP-CT-5c	Ajouter un thème.	Aucun concept sélectionné	✓ refus de l'opération	✓ refus de l'opération	**OUI**	
COSMICEXP-CT-5d	Ajouter un thème.	Sélection : Data Group Percentage : 54% Question : « Identification of Processes » Yes : 23% No : 57%	✓ refus de l'opération	✓ refus de l'opération	**OUI**	
COSMICEXP-CT-5e	Ajouter un thème.	Sélection : Boundary Percentage : 54% Question : « Identification of Processes » Yes : 23% No : 57%	✓ refus de l'opération	✓ mise à jour de la base	**OUI**	

COSMICEXP-CT-5f	Ajouter un thème.	Sélection : Data Group Percentage : 101% Question : « Identification of Processes » Yes : 34% No : -23%	✓ refus de l'opération	✓ refus de l'opération	**OUI**	Un stack d'erreur apparaît bien que l'opération se déroule correctement
COSMICEXP-CT-5g	Ajouter un thème.	Sélection : Data Group-MIS Percentage : 0% Question : « Identification of Processes » Yes : 34% No : -23%	✓ mise à jour de la base	✓ mise à jour de la base	**OUI**	Un stack d'erreur apparaît bien que l'opération se déroule correctement
COSMICEXP-CT-5h	Ajouter un thème.	Sélection : Entry Percentage : 23% Question : « Identification of Transfers» Yes : 25% No : -16%	✓ mise à jour de la base	✓ mise à jour de la base	**OUI**	Un stack d'erreur apparaît bien que l'opération se déroule correctement
COSMICEXP-CT-5i	Ajouter un thème.	Sélection : Exit Percentage : 23% Question :	✓ mise à jour de la base	✓ mise à jour de la base	**OUI**	Un stack d'erreur apparaît bien que l'opération se déroule correctement

		« Identification of moves» Yes : -76% No : 85 %				
COSMICEXP-CT-5j	Ajouter un thème.	Sélection : Read Percentage : 23% Question : « Identification of miti» Yes : 12% No : 12%	✓ mise à jour de la base	✓ mise à jour de la base	**OUI**	Un stack d'erreur apparaît bien que l'opération se déroule correctement
COSMICEXP-CT-5k	Ajouter un thème.	Sélection : Data Group Percentage : 34% Question : « Search of data movement » Yes : 101% No : -101%	✓ refus de l'opération	✓ refus de l'opération	**OUI**	
COSMICEXP-CT-5l	Ajouter un thème.	Clic sur Cancel	✓ retour à la page précédente	✓ retour à la page précédente	**OUI**	Un stack d'erreur apparaît.
COSMICEXP-CT-6a	Ajouter un fait	Sélection : Data Group Real Time/ Persistance of Data Group Name: plus ou moins	✓ ajout du fait	✓ ajout du fait	**OUI**	L'ajout s'effectue bien mais un stack d'erreur apparaît néanmoins.

		Percentage : 51%				
COSMICEXP-CT-6b	Ajouter un fait	Sélection : Boundary/ Id of Triggering Event Name: Yes Percentage : 44%	✓ refus de l'opération	✓ refus de l'opération	**OUI**	L'ajout, comme on le désirait, ne s'effectue pas mais un stack d'erreur apparaît néanmoins.
COSMICEXP-CT-6c	Ajouter un fait	Boundary/ Id of Triggering Event Name: Yes Percentage : 85%	✓ refus de l'opération	✓ refus de l'opération	**OUI**	L'ajout ne s'effectue pas mais un stack d'erreur apparaît néanmoins.
COSMICEXP-CT-6d	Ajouter un fait	Sélection : Data Group Real Time/ Persistance of Data Group Name: mouais Percentage : 101%	✓ refus de l'opération	✓ refus de l'opération	**OUI**	L'ajout ne s'effectue pas mais un stack d'erreur apparaît néanmoins.
COSMICEXP-CT-6e	Ajouter un fait	Sélection : Data Group Real Time/ Persistance of Data Group Name: mouais Percentage : 0%	✓ ajout du fait	✓ ajout du fait	**OUI**	L'ajout s'effectue bien mais un stack d'erreur apparaît néanmoins.
COSMICEXP-CT-6f	Ajouter un fait	Clic sur **Cancel**	✓ retour à la page précédente sans modification de la base	✓ retour à la page précédente sans modification de la base	**OUI**	Un stack d'erreur apparaît.
COSMICEXP-CT-7a	Ajouter une recommandation	Sélection : Customer Entity	✓ ajout de la recommandation dans le domaine du cas problème	✓ ajout de la recommandation dans le domaine du cas problème	**OUI**	

		Answer : Did you answer? Min % : 23 % Max % : 45 %	choisi	choisi		
COSMICEXP-CT-7b	Ajouter une recommandation	Aucun champ rempli	✓ refus du passage à l'étape suivante	✓ refus du passage à l'étape suivante	**OUI**	
COSMICEXP-CT-7c	Ajouter une recommandation	Sélection : Customer Entity Answer : Does it suit you? Min % : 89 % Max % : 23 %	✓ refus du passage à l'étape suivante	✓ refus du passage à l'étape suivante	**OUI**	
COSMICEXP-CT-7d	Ajouter une recommandation	Sélection : Customer Entity Answer : Does it suit you? Min % : 51 % Max % : 51 %	✓ refus du passage à l'étape suivante	✓ refus du passage à l'étape suivante	**OUI**	On obtient le message « Maximum percentage must be greater than the minimum percentage ».
COSMICEXP-CT-7e	Ajouter une recommandation	Sélection : Customer Entity Answer : Does it suit you? Min % : 0 % Max % : 0 %	✓ refus du passage à l'étape suivante	✓ refus du passage à l'étape suivante	**OUI**	On obtient le message « Maximum percentage must be greater than the minimum percentage ».
COSMICEXP-CT-7f	Ajouter une recommandation	Sélection : Customer Entity	✓ refus du passage à l'étape suivante	✓ refus du passage à l'étape suivante	**OUI**	

		Answer : Does it suit you? Min % : -101 % Max % : 0 %				
COSMICEXP-CT-7g	Ajouter une recommandation	Clic sur **Cancel** à chaque étape	✓ retour à la page précédente sans modification de la base	✓ retour à la page précédente sans modification de la base	**OUI**	
COSMICEXP-CT-8a	Modifier un mot clef	Sélection : Read Sélection : Data group Keyword : Data groups RTC : Read 50 % -> 51 %	✓ modification du mot clef dans la base de connaissance	✓ modification du mot clef dans la base de connaissance	**OUI**	
COSMICEXP-CT-8b	Modifier un mot clef	Sélection : Read Sélection : Data groups Aucune modification	✓ base de connaissance inchangée	✓ base de connaissance inchangée	**OUI**	
COSMICEXP-CT-8c	Modifier un mot clef	Sélection : Read Sélection : Data groups Effacement de l'entièreté des champs	✓ refus de la modification	✓ refus de la modification	**OUI**	Des champs obligatoires sont effacés.
COSMICEXP-CT-8d	Modifier un mot clef	Sélection : Read Sélection : Data	✓ refus de modification	✓ refus de modification	**OUI**	Impossibilité de tout décocher dans la partie Related Topological

		groups On décoche l'entièreté des cases de la colonne Related Topological Concept(s)				Concept(s). Donc, comme l'ensemble des cases ne peut pas être décoché, le mécanisme de protection est suffisant pour enregistrer la modification du mot clef.
COSMICEXP-CT-8e	Modifier un mot clef	Sélection : Read Sélection : Data groups Certains pourcentages de la colonne RTC sont placés à 101 %	✓ refus de modification	✓ refus de modification	**OUI**	Test effectué également avec la valeur -101 %.
COSMICEXP-CT-8f	Modifier un mot clef	Sélection : Read Sélection : Data groups Certains pourcentages de la colonne RTC sont placés à 0 %	✓ mise à jour de la base	✓ mise à jour de la base	**OUI**	
COSMICEXP-CT-8g	Modifier un mot clef	Clic sur **Cancel** à chaque étape	✓ retour à la page précédente sans modification de la base	✓ retour à la page précédente sans modification de la base	**OUI**	
COSMICEXP-CT-9	Modifier un concept topologique				**NON**	Non implémenté.
COSMICEXP-CT-	Modifier un cas	Sélection :	✓ modification du cas	✓ modification du cas	**OUI**	Message qui confirme le bon

10a	problème	Boundary Sélection : User Modification de ce cas problème	problème dans la base de connaissance	problème dans la base de connaissance		déroulement de la modification mais la base de connaissance est intacte. Aucunes modifications n'ont été appliquées.
COSMICEXP-CT-10b	Modifier un cas problème				**NON**	Cf. **COSMICEXP-CT-4b**
COSMICEXP-CT-10c	Modifier un cas problème				**NON**	Cf. **COSMICEXP-CT-4c**
COSMICEXP-CT-10d	Modifier un cas problème				**NON**	Cf. **COSMICEXP-CT-4d**
COSMICEXP-CT-10e	Modifier un cas problème				**NON**	Cf. **COSMICEXP-CT-4e**
COSMICEXP-CT-10f	Modifier un cas problème				**NON**	Cf. **COSMICEXP-CT-4f**
COSMICEXP-CT-10g	Modifier un cas problème				**NON**	Cf. **COSMICEXP-CT-4g**
COSMICEXP-CT-10h	Modifier un cas problème				**NON**	Cf. **COSMICEXP-CT-4h**
COSMICEXP-CT-10i	Modifier un cas problème				**NON**	Cf. **COSMICEXP-CT-4i**
COSMICEXP-CT-11a	Modifier un thème	Sélection : Boundary Sélection : Identification of triggering event	✓ modification du thème dans la base de connaissance	✓ modification du thème dans la base de connaissance	**OUI**	Un stack d'erreur apparaît et donc handicape la vérification de la modification d'un thème.

		Modification du thème				
COSMICEXP-CT-11b	Modifier un thème				**NON**	Cf. **COSMICEXP-CT-5b**
COSMICEXP-CT-11c	Modifier un thème				**NON**	Cf. **COSMICEXP-CT-5c**
COSMICEXP-CT-11d	Modifier un thème				**NON**	Cf. **COSMICEXP-CT-5d**
COSMICEXP-CT-11e	Modifier un thème				**NON**	Cf. **COSMICEXP-CT-5e**
COSMICEXP-CT-11f	Modifier un thème				**NON**	Cf. **COSMICEXP-CT-5f**
COSMICEXP-CT-11g	Modifier un thème				**NON**	Cf. **COSMICEXP-CT-5g**
COSMICEXP-CT-11h	Modifier un thème				**NON**	Cf. **COSMICEXP-CT-5h**
COSMICEXP-CT-11i	Modifier un thème				**NON**	Cf. **COSMICEXP-CT-5i**
COSMICEXP-CT-11j	Modifier un thème				**NON**	Cf. **COSMICEXP-CT-5j**
COSMICEXP-CT-11k	Modifier un thème				**NON**	Cf. **COSMICEXP-CT-5k**
COSMICEXP-CT-11l	Modifier un thème				**NON**	Cf. **COSMICEXP-CT-5l**
COSMICEXP-CT-	Modifier un fait	Sélection :	✓ modification du fait dans la base de connaissance	✓ modification du fait dans la base de connaissance	**OUI**	Un stack d'erreur apparaît mais la

12a		Boundary Sélection : Identification of triggering event Sélection : Yes Modification : 85 % -> 79 %				base de connaissance a été modifiée correctement.
COSMICEXP-CT-12b	Modifier un fait				**NON**	Cf. **COSMICEXP-CT-6b**
COSMICEXP-CT-12c	Modifier un fait				**NON**	Cf. **COSMICEXP-CT-6c**
COSMICEXP-CT-12d	Modifier un fait				**NON**	Cf. **COSMICEXP-CT-6d**
COSMICEXP-CT-12e	Modifier un fait				**NON**	Cf. **COSMICEXP-CT-6e**
COSMICEXP-CT-12f	Modifier un fait				**NON**	Cf. **COSMICEXP-CT-6f**
COSMICEXP-CT-13a	Modifier une recommandation	Sélection : Customer Entity Recommandation : You answer positively to each theme. Min % : 98 % → 82 % Max % : 100 % →	✓ modification de la recommandation dans la base de connaissance	✓ modification de la recommandation dans la base de connaissance	**OUI**	

		100 %				
COSMICEXP-CT-13b	Modifier une recommandation				**NON**	Cf. **COSMICEXP-CT-7b**
COSMICEXP-CT-13c	Modifier une recommandation				**NON**	Cf. **COSMICEXP-CT-7c**
COSMICEXP-CT-13d	Modifier une recommandation				**NON**	Cf. **COSMICEXP-CT-7d**
COSMICEXP-CT-13e	Modifier une recommandation				**NON**	Cf. **COSMICEXP-CT-7e**
COSMICEXP-CT-13f	Modifier une recommandation				**NON**	Cf. **COSMICEXP-CT-7f**
COSMICEXP-CT-13g	Modifier une recommandation				**NON**	Cf. **COSMICEXP-CT-7g**
COSMICEXP-CT-14	Effacer un mot-clef	Sélection : Boundary Mot-clef : User	✓ retrait du mot-clef du domaine du concept topologique sélectionné dans la base de connaissance	✓ retrait du mot-clef du domaine du concept topologique sélectionné dans la base de connaissance	**OUI**	Un avertissement nous avertit qu'il est impossible de supprimer le mot-clef dans le concept topologique car il existe dans un autre de ceux-ci. Un autre test avec comme entrées : Sélection : Data Group Real Time Mot-clef : Real Time Software L'effacement s'est effectué correctement car le mot-clef n'était présent dans aucun autre concept topologique.
COSMICEXP-CT-15	Effacer un				**NON**	Non implémenté.

	concept topologique					
COSMICEXP-CT-16	Effacer un cas problème	Sélection : Data Group Real Time Cas problème : Cooking mode	✓ retrait du cas problème et de ses références dans la base de connaissance	✓ retrait du cas problème et de ses références dans la base de connaissance	**OUI**	
COSMICEXP-CT-17	Effacer un thème	Sélection : Boundary Thème : Identification of the triggering event	✓ retrait d'un thème du domaine du concept topologique sélectionné dans la base de connaissance	✓ retrait d'un thème du domaine du concept topologique sélectionné dans la base de connaissance	**OUI**	La confirmation de la suppression du thème n'est pas explicite. Le thème, précédemment sélectionné, est bien effacé mais on obtient un stack d'erreur à la fin de la procédure.
COSMICEXP-CT-18	Effacer un fait	Sélection : Data Group Thème : Persistence of data group Fait : Yes	✓ retrait d'un fait lié à un thème du concept topologique sélectionné dans la base de connaissance	✓ retrait d'un fait lié à un thème du concept topologique sélectionné dans la base de connaissance	**OUI**	Un stack d'erreur apparaît et le fait ciblé est toujours dans la base de connaissance.
COSMICEXP-CT-19	Effacer une recommandation	Sélection : Customer Entity Recommandation : You answer positively to each theme. Min % : 98 % → 82 % Max % : 100 % → 100 %	✓ retrait d'une recommandation du domaine du concept topologique sélectionné dans la base de connaissance	✓ retrait d'une recommandation du domaine du concept topologique sélectionné dans la base de connaissance	**OUI**	

Annexe 6 : « struts-config.xml »

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<struts-config>
      <!-- ========== Form Bean Definitions
================================== -->

      <form-beans>

            <form-bean  name="indexForm"
                        type="application.index.AddEntryForm"/>

            <form-bean  name="modindForm"
                        type="application.index.ModifyEntryForm"/>

            <form-bean  name="delindForm"
                        type="application.index.DeleteEntryForm"/>

            <form-bean  name="loginForm"
                        type="application.security.LoginForm"/>

            <form-bean  name="adduForm"
                        type="application.administration.AddUserForm"/>

            <form-bean  name="chgpsdForm"
                        type="application.administration.ModifyPwdForm"/>

            <form-bean  name="chgdatForm"
                        type="application.administration.ModifyDateForm"/>

            <form-bean  name="chgproForm"

      type="application.administration.ModifyProfileForm"/>

            <form-bean  name="chgstaForm"

      type="application.administration.ModifyStatusForm"/>

            <form-bean  name="delusrForm"
                        type="application.administration.DeleteUserForm"/>

            <form-bean  name="addkwForm"
                        type="application.KBase.AddKeywordForm"/>

            <form-bean  name="modkeForm"
                        type="application.KBase.ModifyKeywordForm"/>

            <form-bean  name="delkwForm"
                        type="application.KBase.DeleteKeywordForm"/>

            <form-bean  name="addcoForm"
                        type="application.KBase.AddConceptForm"/>
```

```
            <form-bean  name="addcpForm"
                        type="application.KBase.AddCpForm"/>

            <form-bean  name="modcpForm"
                        type="application.KBase.ModifyCpForm"/>

            <form-bean  name="delcpForm"
                        type="application.KBase.DeleteCpForm"/>

            <form-bean  name="addqsForm"
                        type="application.KBase.AddQuestionForm"/>

            <form-bean  name="modqsForm"
                        type="application.KBase.ModifyQuestionForm"/>

            <form-bean  name="delthForm"
                        type="application.KBase.DeleteQuestionForm"/>

            <form-bean  name="addfaForm"
                        type="application.KBase.AddFactForm"/>

            <form-bean  name="modfaForm"
                        type="application.KBase.ModifyFactForm"/>

            <form-bean  name="delfaForm"
                        type="application.KBase.DeleteFactForm"/>

            <form-bean  name="addreForm"
                        type="application.KBase.AddRecommandationForm"/>

            <form-bean  name="modreForm"
                        type="application.KBase.ModifyRecommandationForm"/>

            <form-bean  name="delreForm"
                        type="application.KBase.DeleteRecommandationForm"/>

            <form-bean  name="setfilForm"
                        type="application.config.SetFilterForm"/>

      </form-beans>
      <!-- ========== Action Mapping Definitions
============================== -->
      <action-mappings>

            <action     path="/index"
                  type="application.index.AddEntryAction"
                  name="indexForm"
                        scope="request"
                  input="/pages/xpert/report.jsp"
                  validate="true">
                  <forward name="failure_addentry_present"
path="/pages/xpert/associateToKeyword.jsp"/>
                  <forward name="success_addentry"
path="/pages/xpert/success.jsp"/>
            </action>

            <action     path="/modind"
                  type="application.index.ModifyEntryAction"
                  name="modindForm"
                        scope="request"
```

```
                input="/pages/xpert/deleteIndexEXE.jsp"
                validate="true">
                <forward name="failure_modifyentry_newname"
path="/pages/xpert/KwNodeIndTree.jsp"/>
                <forward name="failure_modifyentry_notexist"
path="/pages/xpert/KwNodeIndTree.jsp"/>
                <forward name="failure_modifyentry_nonodes"
path="/pages/xpert/KwNodeIndTree.jsp"/>
                <forward name="success_modifyentry"
path="/pages/xpert/success.jsp"/>
            </action>

            <action     path="/delind"
                type="application.index.DeleteEntryAction"
                name="delindForm"
                      scope="request"
                input="/pages/xpert/deleteIndexEXE.jsp"
                validate="true">
                <forward name="failure_deleteentry_notexist"
path="/pages/xpert/deleteIndexEXE.jsp"/>
                <forward name="failure_deleteentry_nonodes"
path="/pages/xpert/deleteIndexEXE.jsp"/>
                <forward name="success_deleteentry"
path="/pages/xpert/success.jsp"/>
            </action>


            <action     path="/login"
                type="application.security.LoginAction"
                name="loginForm"
                      scope="request"
                input="/login.jsp"
                validate="true">
                <forward name="failure" path="/login.jsp"/>
                <forward name="success_admin"
path="/pages/admin/main.jsp"/>
                <forward name="success_user"
path="/pages/users/main.jsp"/>
                <forward name="success_expert"
path="/pages/xpert/main.jsp"/>
            </action>

            <action     path="/addu"
                type="application.administration.AddUserAction"
                name="adduForm"
                      scope="request"
                input="/pages/admin/addUser.jsp"
                validate="true">
                <forward name="failure_adduser_present"
path="/pages/admin/error.jsp"/>
                <forward name="success_adduser"
path="/pages/admin/confirm.jsp"/>
            </action>

            <action     path="/chgpsd"
                type="application.administration.ModifyPwdAction"
                name="chgpsdForm"
                      scope="request"
                input="/pages/admin/chgPsw.jsp"
                validate="true">
```

```
                  <forward name="failure_modifypassword"
path="/pages/admin/error.jsp"/>
                  <forward name="success_modifypassword"
path="/pages/admin/confirm.jsp"/>
            </action>

            <action     path="/chgdat"
                  type="application.administration.ModifyDateAction"
                  name="chgdatForm"
                        scope="request"
                  input="/pages/admin/chgDat.jsp"
                  validate="true">
                  <forward name="failure_modifydate_date"
path="/pages/admin/error.jsp"/>
                  <forward name="success_modifydate"
path="/pages/admin/confirm.jsp"/>
            </action>

            <action     path="/chgsta"
                  type="application.administration.ModifyStatusAction"
                  name="chgstaForm"
                        scope="request"
                  input="/pages/admin/main.jsp"
                  validate="true">
                  <forward name="success_modifystatus"
path="/pages/admin/main.jsp"/>
            </action>

            <action     path="/chgpro"
                  type="application.administration.ModifyProfileAction"
                  name="chgproForm"
                        scope="request"
                  input="/pages/admin/chgPro.jsp"
                  validate="true">
                  <forward name="error" path="/pages/admin/error.jsp"/>
                  <forward name="success_modifypro"
path="/pages/admin/confirm.jsp"/>
            </action>

            <action     path="/delusr"
                  type="application.administration.DeleteUserAction"
                  name="delusrForm"
                        scope="request"
                  input="/pages/admin/main.jsp"
                  validate="true">
                  <forward name="success_deleteuser"
path="/pages/admin/main.jsp"/>
            </action>


            <action     path="/addkw"
                  type="application.KBase.AddKeywordAction"
                  name="addkwForm"
                        scope="request"
                  input="/pages/xpert/report.jsp"
                  validate="true">
                  <forward name="failure_addkeyword_present"
path="/pages/xpert/main.jsp"/>
                  <forward name="success_addkeyword"
path="/pages/xpert/main.jsp"/>
            </action>
```

```
            <action     path="/modke"
                  type="application.KBase.ModifyKeywordAction"
                  name="modkeForm"
                        scope="request"
                  input="/pages/xpert/report.jsp"
                  validate="true">
                  <forward name="failure_modifykeyword_notexist"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_modifykeyword_newname"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_modifykeyword_nokws"
path="/pages/xpert/success.jsp"/>
                  <forward name="ok_ke" path="/pages/xpert/success.jsp"/>
            </action>

            <action     path="/delkw"
                  type="application.KBase.DeleteKeywordAction"
                  name="delkwForm"
                        scope="request"
                  input="/pages/expert/delete.jsp"
                  validate="true">
                  <forward name="failure_deletekeyword_notexist"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_deletekeyword_last"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_deletekeyword_nokws"
path="/pages/xpert/success.jsp"/>
                  <forward name="ok_del" path="/pages/xpert/success.jsp"/>
            </action>


            <action     path="/addco"
                  type="application.KBase.AddConceptAction"
                  name="addcoForm"
                        scope="request"
                  input="/pages/xpert/report.jsp"
                  validate="true">
                  <forward name="ok_kw" path="/pages/xpert/addTC.jsp"/>
                  <forward name="ok_tc"
path="/pages/xpert/addTC_step2.jsp"/>
                  <forward name="ok_2"
path="/pages/xpert/tcaddCpStep1.jsp"/>
                  <forward name="ok_3a"
path="/pages/xpert/addTC_step3b.jsp"/>
                  <forward name="ok_3"
path="/pages/xpert/addTC_step4b.jsp"/>

                  <forward name="failure_addconcept_newname"
path="/pages/xpert/main.jsp"/>
                  <forward name="ok" path="/pages/xpert/main.jsp"/>
            </action>


            <action     path="/addcp"
                  type="application.KBase.AddCpAction"
                  name="addcpForm"
                        scope="request"
                  input="/pages/xpert/report.jsp"
                  validate="true">
```

```
                  <forward name="failure_addcp_present"
path="/pages/xpert/success.jsp"/>
                  <forward name="ok_cp" path="/pages/xpert/success.jsp"/>
            </action>

            <action     path="/modcp"
                  type="application.KBase.ModifyCpAction"
                  name="modcpForm"
                        scope="request"
                  input="/pages/xpert/report.jsp"
                  validate="true">
                  <forward name="failure_modifycp_probpath"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_modifycp_notexist"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_modifycp_nocps"
path="/pages/xpert/success.jsp"/>
                  <forward name="ok_cp" path="/pages/xpert/success.jsp"/>
            </action>

            <action     path="/delcp"
                  type="application.KBase.DeleteCpAction"
                  name="delcpForm"
                        scope="request"
                  input="/pages/expert/delete.jsp"
                  validate="true">
                  <forward name="failure_deletecp_last"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_deletecp_notexist"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_deletecp_nocps"
path="/pages/xpert/success.jsp"/>
                  <forward name="ok_del" path="/pages/xpert/success.jsp"/>
            </action>


            <action     path="/addqs"
                  type="application.KBase.AddQuestionAction"
                  name="addqsForm"
                        scope="request"
                  input="/pages/xpert/report.jsp"
                  validate="true">
                  <forward name="failure_addquestion_present"
path="/pages/xpert/success.jsp"/>
                  <forward name="ok_th" path="/pages/xpert/success.jsp"/>
            </action>

            <action     path="/modqs"
                  type="application.KBase.ModifyQuestionAction"
                  name="modqsForm"
                        scope="request"
                  input="/pages/xpert/report.jsp"
                  validate="true">
                  <forward name="failure_modifyquestion_notexist"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_modifyquestion_exist"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_modifyquestion_noquestions"
path="/pages/xpert/success.jsp"/>
                  <forward name="ok_th" path="/pages/xpert/success.jsp"/>
            </action>
```

```
            <action     path="/delth"
                  type="application.KBase.DeleteQuestionAction"
                  name="delthForm"
                        scope="request"
                  input="/pages/expert/delete.jsp"
                  validate="true">
                  <forward name="failure_deletequestion_notexist"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_deletequestion_last"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_deletequestion_noquestions"
path="/pages/xpert/success.jsp"/>
                  <forward name="ok_del" path="/pages/xpert/success.jsp"/>
            </action>


            <action     path="/addfa"
                  type="application.KBase.AddFactAction"
                  name="addfaForm"
                        scope="request"
                  input="/pages/xpert/report.jsp"
                  validate="true">
                  <forward name="failure_addfact_present"
path="/pages/xpert/success.jsp"/>
                  <forward name="ok_fa" path="/pages/xpert/success.jsp"/>
            </action>

            <action     path="/modfa"
                  type="application.KBase.ModifyFactAction"
                  name="modfaForm"
                        scope="request"
                  input="/pages/xpert/report.jsp"
                  validate="true">
                  <forward name="failure_modifyfact_newname"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_modifyfact_notyesno"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_modifyfact_notexist"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_modifyfact_nofacts"
path="/pages/xpert/success.jsp"/>
                  <forward name="ok_fa" path="/pages/xpert/success.jsp"/>

            </action>

            <action     path="/delfa"
                  type="application.KBase.DeleteFactAction"
                  name="delfaForm"
                        scope="request"
                  input="/pages/expert/delete.jsp"
                  validate="true">
                  <forward name="failure_deletefact_newname"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_deletefact_notyesno"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_deletefact_last"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_deletefact_notexist"
path="/pages/xpert/success.jsp"/>
```

```
                  <forward name="failure_deletefact_nofacts"
path="/pages/xpert/success.jsp"/>
                  <forward name="ok_del" path="/pages/xpert/success.jsp"/>
            </action>


            <action     path="/addre"
                  type="application.KBase.AddRecommandationAction"
                  name="addreForm"
                        scope="request"
                  input="/pages/xpert/report.jsp"
                  validate="true">
                  <forward name="failure_addrecommandation_present"
path="/pages/xpert/success.jsp"/>
                  <forward name="ok_re" path="/pages/xpert/success.jsp"/>
            </action>

            <action     path="/modre"
                  type="application.KBase.ModifyRecommandationAction"
                  name="modreForm"
                        scope="request"
                  input="/pages/xpert/report.jsp"
                  validate="true">
                  <forward name="failure_modifyrecommandation_newrecom"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_modifyrecommandation_notexist"
path="/pages/xpert/success.jsp"/>
                  <forward
            name="failure_modifyrecommandation_norecommandations"
                        path="/pages/xpert/success.jsp"/>
                  <forward name="ok_mo" path="/pages/xpert/success.jsp"/>

            </action>

            <action     path="/delre"
                  type="application.KBase.DeleteRecommandationAction"
                  name="delreForm"
                        scope="request"

                  validate="true">
                  <forward name="failure_deleterecommandation_last"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_deleterecommandation_stilllinks"
path="/pages/xpert/success.jsp"/>
                  <forward name="failure_deleterecommandation_notexist"
path="/pages/xpert/success.jsp"/>
                  <forward
name="failure_deleterecommandation_norecommandations"
                        path="/pages/xpert/success.jsp"/>
                  <forward name="ok_del" path="/pages/xpert/success.jsp"/>
            </action>

            <action     path="/setfil"
                  type="application.config.SetFilterAction"
                  name="setfilForm"
                        scope="request"
                  input="/pages/xpert/SetFilter.jsp"
                  validate="true">
                  <forward name="ok" path="/pages/xpert/SetFilter.jsp"/>
            </action>
```

```
      </action-mappings>


      <!-- ========== Message Resources Definitions
============================ -->
      <message-resources     null="false"
                             parameter="ApplicationResources"/>

</struts-config>
```

Annexe 7 : Précision sur l’architecture logique

Le Figure suivante illustre un diagramme de robustesse. Il met en jeu l'évaluateur (utilisateur de base du logiciel SMXpert) interagissant avec le système pour consulter la base de connaissances et, de ce fait, obtenir une recommandation.

L'ensemble des cas d'utilisation liés à la consultation de la base de connaissances de la part de l'évaluateur est illustré par cette Figure 1. Il est notable que la même structure est valable pour la consultation et les modifications de la base de connaissances par l’expert. Ce fait nous conduit à découper en un seul composant « Base de Connaissances » l’ensemble des traitements liés à celle-ci.

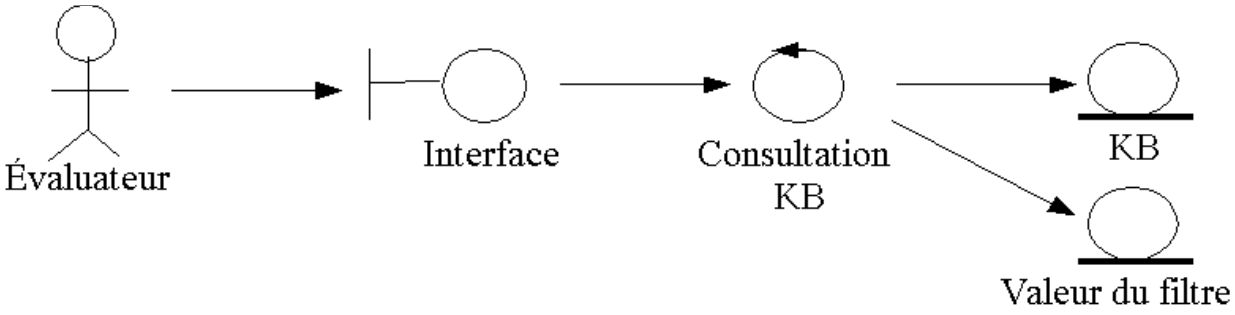


Figure 1 : Diagramme de robustesse pour la consultation par un évaluateur.

A propos des diagrammes de séquencement, les interactions entre les différents acteurs et le système correspondent à un même canevas (consultation, ajout, modification, suppression). Ce sont les mêmes que celles de COSMICXpert. Nous ne les avons donc pas développé plus avant.

Annexe 8 : Précisions sur le Chapitre 4

8.1 Java

Créé à la fin des années 1980 par l'entreprise Sun Microsystems, Java est un langage de programmation orienté objet indépendant de toute plate-forme. Cette portabilité provient du fait que les classes compilées de Java ne sont pas directement compréhensibles et exécutables par une configuration donnée. En effet, une interprétation est nécessaire. Celle-ci est effectuée par la « Java Virtual Machine » (qui, elle, est dépendante de la plate-forme) en vue de permettre l'exécution des classes sur une configuration spécifique. Cette précédente indépendance est un avantage vu que l'ensemble des serveurs composant Internet forme un parc d'ordinateurs aux configurations hétérogènes. Le fait que Java soit un langage orienté objet permet la création de composants qui peuvent être utilisés sans se préoccuper de leur implémentation, seules les spécifications de l'interface extérieure qu'ils présentent sont importantes. Ce phénomène a permis d'engendrer un grand nombre de bibliothèques qui évitent d'implémenter une nouvelle fois des mécanismes particuliers (parser un fichier xml, …) ou communs. La richesse de ce langage ne cesse donc d'augmenter.

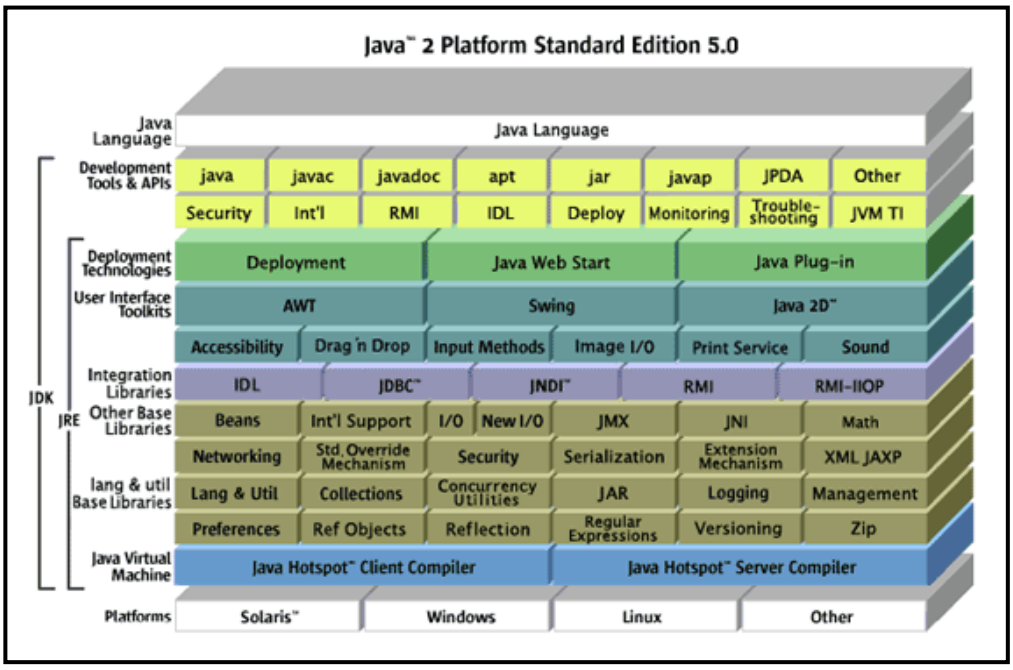


Figure 8.1 : Diagramme conceptuel de la plateforme Java 2 Standard Edition 5.0 [Sun Microsystems]

8.2 Tag Libraries

Dans la spécification v1.1 de la technologie des JavaServer Pages, il est prévu l'existence des « custom actions » qui sont des modules réutilisables (l'implémentation étant localisée à un endroit bien précis) ayant la capacité d'accéder à des objets de langages de programmation. Le but de leur utilisation est la modification de la page affichée.

Au sein des pages JSP, les « custom actions » sont appelées grâce à des « custom tags » insérés au dans le code de celles-ci. Quant à la « Tag Library », c'est un ensemble de « custom tags » ayant un lien entre-eux (affichage, interactions avec une base de données, …). Par exemple, la « Tag Library » *x* regroupe des traitements liés aux fichiers XML. Dans le cadre de SMXpert, les « Tag Libraries » suivantes ont été utilisées : *c*, *x*, *fmt* et *JSTL*.

A chaque « Tag Library » est associé un fichier contenant la description des « custom tags » qu'il regroupe. Il s'appelle le « Tag Library Descriptor » (*.tld).

L'avantage de ces « custom tags » est de simplifier les pages JSP de par la suppression du code source Java qui aurait été nécessaire pour atteindre les mêmes objectifs. De par leur réutilisation, cela permet d'éviter les erreurs et la redondance de l'implémentation des fonctionnalités (directement codées dans les pages) qui auraient pu se retrouver dans plusieurs pages.

8.3 Apache Tomcat

Tomcat est développé par Sun et est une implémentation « open-source » des spécifications des J2EE Java servlets et des JavaServer Pages. Il est fortement utilisé car il est considéré comme étant l'implémentation officielle de référence des technologies précédentes.

Servlet/JSP Spec	Apache Tomcat version
2.4/2.0	5.5.9
2.3/1.2	4.1.31
2.2/1.1	3.3.2

Figure 8.2 : Différentes versions du serveur Tomcat et les versions des spécifications des servlet et des pages JSP respectées [Apache Software Foundation]

Il peut également servir de Web Server pour afficher des pages statiques qui ne nécessitent pas de compilation pour pouvoir être transmises à l'utilisateur. De plus, Tomcat n'est pas obligé d'effectuer cette tâche car il est doté d'une fonctionnalité qui permet de l'interfacer avec un Web Server plus commun comme Apache Web Server. Son domaine d'activité devient le traitement des requêtes entraînant des réponses dynamiques qui mettent en jeu des Java Servlets et des JavaServer Pages.

8.4 Historique de Struts

Le créateur de Struts s'appelle Craig R. McClanahan. Il est actuellement employé par Sun Microsystems. Dès 1998, il fut impliqué dans les technologies liées aux servlets et aux JSP (« Java Server Pages »). Remarquant les mauvaises utilisations des scriptlets dans les pages JSP, il eut l'idée de se concentrer sur le sujet des architectures adéquates pour les applications Web. C'est ainsi qu'il décida de créer un cadre de référence.

Au même moment, il joignit les groupes d'experts travaillant sur les spécifications des servlet 2.2-2.3 et des JSP 1.1-1.2. Craig R. McClanahan a également travaillé sur le serveur Web Tomcat. Ce n'est qu'à la fin de mai 2000 que la première version du code de Struts fut écrite. Il est à noter que le nom du cadre de référence provient de ce que l'on pourrait considérer comme l'armature ou la charpente d'une maison ou d'un pont.

La propriété du code de Struts a été cédée à l'organisation « Apache Software Foundation ». C'est une association à but non lucratif qui offre un certain support à ses membres et une entité légale pour protéger ses ressources. Actuellement, elle regroupe 16 projets dont le projet Jakarta qui a pour but de créer et maintenir des solutions du côté serveur basées sur la plateforme Java. Ils atteignent une qualité commerciale mais ont comme caractéristique d'être « open-source » et gratuits. Struts fait partie des 22 sous-projets de Jakarta.

Finalement, les différentes applications qui seront développées à l'avenir en utilisant Struts devront respecter la licence de l'organisation « Apache Software Foundation » [Web4].

8.5 Les packages de Struts

Détails des différents packages :

- action : Classes du contrôleur, certaines classes de la vue (ActionForm, ActionMessages) et d'autres classes requises pour le fonctionnement du cadre de référence.
- actions : Classes ayant un lien avec les classes Action, comme la classe DispachtAction.
- config : Classes de configuration qui sont des représentations en mémoire du fichier de configuration de Struts.
- taglib : Classes gérants les tags des bibliothèques de tags de Struts.
- tiles : Classes utilisées par le cadre de référence « Tiles ».
- upload : Classes qui permettent le download et l'upload de fichiers entre le navigateur et l'application Web.
- util : Classes d'utilité générale utilisées par l'entièreté du cadre de référence.
- validator : Classes d'extension spécifiques à Struts et utilisées par celui-ci lorsque l'on déploie le validateur. Il est à noter que les classes et interfaces se trouvent dans le package « common », séparé de Struts.

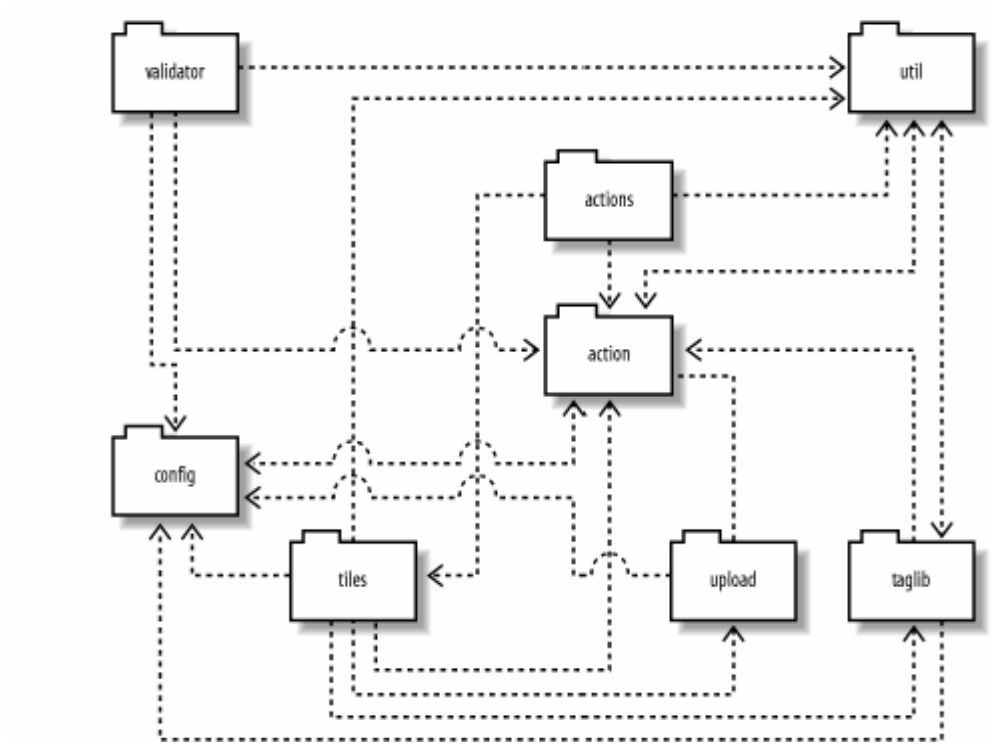


Figure 8.3 : Les huit packages composant le cadre de référence Struts [52]

8.6 Exemple de gestion des erreurs

```
public class AddFactForm extends ActionForm {
```

```
private String path = null;
private String idTH = null;
private String name = null;
private String cf = null;

public void reset(ActionMapping mapping, HttpServletRequest request)
{
    path = "";
    idTH = "";
    name = "";
    cf = "";
}
    public ActionErrors validate( ActionMapping mapping,
                                  HttpServletRequest request)
{
    ActionErrors errors = new ActionErrors();

    boolean wrong = false;
    boolean missing = false;

    if((name == null) || (name.length() < 1)){
                errors.add( "missing",
                            new ActionError("error.missing.Fname"));
          missing=true;
    }
    if((cf == null) || (cf.length() < 1)){
          errors.add( "missing",
                            new ActionError("error.missing.Fpct"));
          missing=true;
    }
    else{
          int pct = Integer.parseInt(cf);
          if (pct>100 || pct<0) {
                errors.add( "wrong",
                                  new ActionError("error.wrong.Prct"));
                wrong=true;
          }
    }

    return errors;
}
```

```
  //les quatres méthodes set et les quatres méthodes get

}
```

Dans cette classe de type ActionForm, dont les méthodes *set* et *get* ont été intentionnellement enlevées, on peut observer les vérifications qui seront effectuées lors de l'ajout d'un fait à une question de la base de connaissance. L'ActionForward compte quatre attributs. Selon la structure de l'application, les attributs « path » et « idTH » sont toujours présents et ne nécessitent pas de contrôle. Cependant, en ce qui concerne le nom (« name ») et le pourcentage (« cf ») du fait, il y a une vérification de l'existence de ceux-ci. Ensuite, on contrôle que le pourcentage soit bien compris entre 0 et 100.

8.7 Scope des objets

Il existe 4 types de scopes :

Request scope : À chaque fois que le client émet une requête, le Web Container créée un objet qui implémente l'interface javax.servlet.http.HttpServletRequest. Dans celui-ci, d'autres objets peuvent être stockés et récupérés le temps de l'existence de celle-là (c'est-à-dire jusqu'à ce que la réponse soit renvoyée au client). Les objets sont stockés sous forme d'une liste de pairs du type <String, Objet>. Trois méthodes principales sont nécessaires à l'utilisation de l'objet HttpServletRequest :

- public void setAttribute(String name, Object obj)
- public Object getAttribute(String name)
- public void removeAttribute(String name)

La visibilité de cet objet et de ceux qu'il encapsule est réservée aux ressources ayant accès à la requête.

Session scope : A un moment qui dépend de l'impémentation de l'application Web (en général, lors de la réception de la première requête d'un utilisateur), le Web Container créée un objet qui implémente l'interface javax.servlet.http.HttpSession. Cet objet est particulièrement pratique car il permet d'identifier l'utilisateur tout au long des requêtes dont il est l'émetteur et d'ainsi paramétrer l'affichage de certaines informations. Par exemple, un objet de type Locale est stocké dans la session et permet d'afficher les pages dans la langue de

l'utilisateur. La durée de vie de celle-ci correspond à la période d'absence de requêtes de la part de l'utilisateur qui lui est lié. Cette période est paramétrable au sein de l'« application deployment descriptor ». En plus des trois méthodes utiles du scope précédent, une quatrième vient de rajouter qui permet de détruire l'objet HttpSession à tout moment :

- public void invalidate()

La visibilité de cet objet est limitée aux instances de classes qui traitent les requêtes provenant d'un utilisateur dont c'est la session. Elle est bien séparée des sessions des autres utilisateurs. Un danger en rapport aux objets HttpSession est que le Web Container ne fournit pas de synchronisation. Si on veut éviter que les objets stockés soient modifiés, il faut créer son propre mécanisme de gestion de la concurrence ou être sûrs que l'accès concurrent à ces objets ne pose pas de problème.

Application scope : Lors du lancement du Web Container et pour chaque application Web, le Web Container créée un objet qui implémente l'interface javax.servlet.ServletContext. Les objets qui seront stockés dans cet objet pourront être accédés jusqu'à ce que l'application se termine. La visibilité du ServletContext et des objets qu'il encapsule est réservée à l'ensemble des utilisateurs et des classes de l'application Web. Ce scope est très utile car on peut stocker les objets qui vont servir à accéder à la couche de persistance. De ce fait, nous pouvons plus facilement contrôler les accès à cette couche et donc, gérer la concurrence. Les méthodes d'utilisation sont les mêmes que pour les scopes précédents.

Page scope : Lorsque l'on utilise une page JSP, le Web Container crée un objet qui implémente l'interface javax.servlet.jsp.PageContext. Cet objet est détruit lorsque la réponse à la requête est envoyée et que cela entraîne l'affichage d'une nouvelle page. L'objet PageContext est uniquement utilisable dans la page au sein de laquelle il a été créé avec les mêmes méthodes que précédemment.

8.8 CustomRequestProcessor

Dans le cas de SMXpert, nous avons créé une classe CustomRequestProcessor (source.framework.CustomRequestProcessor) qui étend la classe RequestProcessor. Notre classe écrase la méthode standard de Struts car il est obligatoire de la réécrire à partir du moment où l'utilisateur peut changer la configuration de sa Locale comme bon lui semble.

```
- protected void processLocale(   HttpServletRequest request,
                                  HttpServletResponse response) {

          if (!appConfig.getControllerConfig().getLocale( )){
                return;
          }

          HttpSession session = request.getSession( );
          Locale sessionLocale =
                      (Locale)session.getAttribute(Action.LOCALE_KEY);

    Locale requestLocale = request.getLocale( );

          if (sessionLocale == null || (sessionLocale != requestLocale)){
                session.setAttribute(Action.LOCALE_KEY, requestLocale );
          }
}
```

Dans l'implémentation de la classe CustomRequestProcessor, le premier test permet de connaître l'état de l'attribut « locale » dans l'élément « controller » du fichier « struts-confg.xml ». Dans le cas où sa valeur est « false », la méthode ne va pas plus loin et se termine. Dans le cas contraire, elle va récupérer l'objet Locale (non nécessairement présent) qui est stocké dans la Session et celui contenu dans la requête de l'utilisateur. Enfin, l'objet Locale de la requête est enregistré dans la Session au cas où il ne serait pas déjà présent ou serait différent du premier.