N° d'ordre : 07/2010-M/INF

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique

Mémoire de magister

En Informatique

Option : Systèmes intelligents et ingénierie des logiciels

Présenté par :

MEZEGHRANE Wahiba

Sujet

La réutilisation des composants logiciels : Approche de caractérisation

Soutenu publiquement, le 16 / 03 / 2010, Devant le jury composé de :

Mme. AISSANI MOKHTARI Aicha	Professeur à l'USTHB	**Présidente**
M. AHMED NACER Mohamed	Professeur à l'USTHB	**Directeur de mémoire**
Mme. BOUKALA Malika	Professeur à l'USTHB	**Examinatrice**
M. ABDELLI Abdelkrim	Maître de conférences à l'USTHB	**Examinateur**
M. BATATA Sofiane	Maître assistant /A à l'ENSI	**Invité**

À ce que j'ai de plus cher au monde : Mes parents

Remerciements

Je remercie Dieu le tout puissant de m'avoir comblé d'opulences, qui sont à l'origine de l'accomplissement de ce travail.

Je tiens à remercier mon encadreur M. AHMED NACER Mohamed, Professeur à l'USTHB d'Alger, d'avoir accepté de diriger ce travail. Sa compétence et ses conseils m'ont été d'une grande utilité pour l'aboutissement de ce travail. Je remercie également M. BATATA Sofiane, Maître assistant à l'institut National d'Informatique, pour sa disponibilité et ses conseils.

J'adresse mes sincères remerciements à Mme. AISSANI MOKHTARI Aicha, Professeur à l'USTHB d'Alger, d'avoir accepté de présider le jury, ainsi que Mme. BOUKALA Malika, Professeur à l'USTHB d'Alger, et M. ABDELLI Abdelkrim, Maître de conférences à l'USTHB d'Alger d'avoir accepté de faire partie du jury.

Je ne pourrai finir ces remerciements sans évoquer les êtres qui me sont les plus chers : mes parents qui sans eux aucune réussite n'aurait pu être possible, toute ma famille, aussi nombreuse soit elle, pour son soutien, son affection et son amour.

Je voudrai remercier profondément mon époux pour son aide précieuse et sa présence inconditionnelle. Je lui suis infiniment reconnaissante.

Enfin, je voudrai remercier toutes les personnes qui ont contribué, de prés ou de loin et sous quelque forme que ce soit, à l'accomplissement de ce travail.

Résumé

Pour promouvoir la réutilisation et l'assemblage des composants logiciels dans le développement des systèmes, les composants qui les constituent, et qui y représentent les briques de base, devraient être facilement retrouvés et adéquatement assemblés lors d'une future réutilisation. Pour y parvenir, le choix d'une méthode pertinente et adaptée de recherche s'avère être indispensable. Nous nous sommes intéressés à la description à base de facette en association avec la description en langage naturel pour profiter à la fois des avantages de l'utilisation du vocabulaire contrôlé et non contrôlé. Nous avons fait une étude sur une douzaine de travaux afin de dégager les facettes qui ciblent la majorité des aspects du composant. Nous avons ensuite fait appel à des techniques de recherche d'information, du traitement du langage naturel et de sélection, à savoir : les modèles booléen et vectoriel, le CF (Concept Frequency), l'algorithme de Porter et les méthodes WSM[1] et AHP[2]. Dans le but de tester le système de recherche mis en œuvre, nous avons rassemblé des composants de divers sites de composants logiciels (Market places) tels que : componentsource, sourceforge, vclcomponent…etc. Pour l'évaluation de notre système de recherche, nous avons utilisé des techniques classiques d'évaluation qui consistent en le calcul du taux de précision, du rappel et du temps d'exécution des requêtes. Les résultats ont été comparés avec l'approche à base de facettes et celle en langage naturel pris séparément. Les résultats de notre proposition étaient meilleurs, en terme de précision qu'en terme de rappel avec un temps d'exécution un peu plus élevé.

Mots Clés: composants logiciels, COTS, réutilisation, caractérisation de composants, recherche de composants.

[1] WSM : Weighted Scoring Method
[2] AHP : Analytic Hierarchy Process

Abstract

In order to ensure the reuse and the assembly of software components for systems development, the components that compose them and which are the building blocks should be easily found and properly assembled in a future reuse. To achieve this, the choice of relevant and appropriate method of research is indispensable. We are interested in faceted categorization of components associated with the natural language description, in order to take advantage of both using controlled and uncontrolled vocabulary. We made a study of a dozen works to identify majority aspects of the component, we then, used the information retrieval's techniques, natural language processing and selection, i.e, the Boolean and vector models, the CF (Concept Frequency), the Porter algorithm, WSM and AHP method. Once the kernel was done, we gathered components from various sites of software components (Market place) such as: componentsource, sourceforge, vclcomponent ...etc. For the assessment of our work, we used classical techniques of evaluation: calculations of the relevance, recall and queries time-running. The results were compared with the faceted categorization approach and the natural language description. Our approach seems to have the best results in terms of recall and relevance, but with more important time-running.

Key words: software components, COTS, design for reuse, design by reuse, component's characterization, component's research.

Table des matières

Liste des figures

Introduction générale

Introduction générale

Après les technologies objet, qui ont profondément modifié l'ingénierie des systèmes logiciels en améliorant leur analyse, leur conception et leur développement, une nouvelle ère de conception de systèmes débute : l'*orienté composant.* Il s'agit de concevoir et de développer des systèmes par assemblage de composants réutilisables, à l'image des composants électroniques ou des composants mécaniques [Ous, al 05]. En effet, la complexité croissante des systèmes informatiques, et leur évolution de plus en plus rapide, ont suscité un intérêt accru pour le développement de logiciels à base de composants. Cet intérêt est motivé par la réduction des coûts et des délais de développement des applications. En d'autres termes, on prend moins de temps à acheter (et donc à réutiliser) un composant que pour le concevoir.

Il n'existe pas de définition unique pour les composants logiciels, Szyperski dans son ouvrage a réuni plus d'une dizaine de définitions [Szy 02]. Même si en apparence, celles-ci sont différentes selon le contexte et les technologies dans lesquelles elles étaient annoncées, elles ont, au fond, le même point de convergence : *la réutilisation.* Par réutilisation, on entend la possibilité de construire une nouvelle application en récupérant le code développé auparavant. Cette réutilisation peut être plus ou moins compliquée. La problématique de recherche dans la réutilisation met en relief deux aspects complémentaires. L'un d'eux concerne à développer des composants prêts à être réutilisés, ce qui correspond à l'aspect « infrastructure » de la réutilisation, appelé l'ingénierie des composants pour la réutilisation « *design for reuse* ». Alors que l'autre concerne à développer des applications en exploitant ces composants réutilisables, ce qui correspond à l'aspect « opérationnel » de la réutilisation, appelé l'ingénierie des applications par la réutilisation « *design by reuse* » [Gac, Ser 03].

Les bibliothèques de composants forment le paradigme dominant dans la réutilisation, puisqu'elles représentent l'intersection de ses deux aspects, à savoir : le design by reuse et le design for reuse. Les deux questions importantes qui doivent être posées, pour bien exploiter ces bibliothèques, sont les suivantes : de quelle manière doit-on organiser et représenter les composants logiciels qui sont disponibles dans la bibliothèque, et comment les retrouver une fois représentés et classés. Notre travail, s'inscrit dans cette perspective. Pour une exploitation optimale de ces bibliothèques, deux aspects fondamentaux doivent être soigneusement

élaborés, à savoir la manière d'organiser et de représenter les composants logiciels qui y sont disponibles, mais aussi la façon de les retrouver une fois représentés et classés.

La recherche des composants logiciels peut s'intéresser au comportement du composant [Alm, al 07], à sa description formelle [Hem, Lin 01] ou à sa représentation externe [Dia 91]. La première, et malgré ses résultats satisfaisants, affecte la performance du système de recherche puisque tous les composants doivent être exécutés. La seconde travaille sur la représentation interne du composant, celle-ci également, réussi à avoir des résultats pertinents mais elle n'est pas toujours réalisable, puisque les composants sont généralement fournis en black box. De ce fait, et pour cerner la majorité des composants, avec un temps de réponse réduit, le mieux est de se concentrer sur la représentation externe du composant. Nous nous sommes intéressés alors à la description à base de facette en association avec la description en langage naturel pour profiter à la fois des avantages de l'utilisation du vocabulaire contrôlé et non contrôlé.

Ce mémoire est structuré en quatre chapitres : Le premier chapitre rassemble les notions de base liées aux composants logiciels, ainsi que les modèles de composants, entre autres EJB[3], .NET[3] et CCM[3]. Le deuxième chapitre aborde la réutilisation des composants logiciels et ses deux aspects : le design for reuse et le design by reuse. Nous présentons dans le troisième chapitre une étude sur la description, la recherche et la sélection des composants logiciels. Le chapitre 4 présente notre problématique de recherche suivie par une proposition pour la caractérisation des composants, une expérimentation et une évaluation de notre approche. Nous terminons cette étude par une conclusion générale, ainsi que les perspectives que nous envisageons comme suite pour notre travail.

[3] Voir page 19 du mémoire

Première partie

État de l'art

Chapitre I

Les composants logiciels

I.1 Introduction

Le paradigme introduit dans les années 90 est la programmation à base de composants. Ce nouveau paradigme vise à améliorer la réutilisation, la sûreté et la flexibilité des applications. Il se base sur la notion de composant et propose de concevoir une application comme un assemblage de briques logicielles préfabriquées.

I.2 Définitions

À l'heure actuelle, il n'existe pas de normalisation pour la notion de composant. Il existe par contre, de nombreuses définitions dans la littérature. Une des définitions la plus souvent citée est celle donnée par Szyperski [Szy 02] :

I.2.1 Un composant logiciel

Un composant logiciel est une unité de composition ayant des interfaces spécifiées de façon contractuelle et possédant uniquement des dépendances de contexte explicites. Un composant peut être déployé de manière indépendante et il est sujet à composition par des tierces parties.

On trouve également les composants *COTS* (Commercial-Off-The-Shelf) qui représentent des composants logiciels commercialisés.

I.2.1.1 Les interfaces du composant

L'interface d'un composant spécifie les services fournis et requis par un type de composant. Ce dernier permet d'interagir avec son environnement, y compris avec d'autres types de composants. L'interface représente la seule partie visible d'un composant. Elle décrit entièrement ce que fait le composant et ce dont il a besoin pour fonctionner. L'interface est composée de deux éléments [Ous, al 05]:

Les points de connexion : ce sont les points d'interaction entre le composant et son environnement. Le point de connexion est également appelé port. Il peut être soit fourni (souvent appelé port service) soit requis (souvent appelé port besoin). Le port requis est le point de connexion qui exprime le fait que le composant nécessite un service pour fonctionner. Le port fourni est le point de connexion qui permet de fournir un service aux autres composants.

Les points de connexion sont les seules parties véritablement manipulables de l'extérieur et on peut seulement se connecter à eux pour recevoir le service voulu ou fournir le besoin demandé.

Les services : Ils permettent d'exprimer la sémantique des fonctionnalités fournies et requises par le composant. Un service peut utiliser plusieurs points de connexion pour exécuter sa tâche.

Techniquement, une interface est un ensemble d'opérations nommées qui vont être invoquées par le client. Le fournisseur et le client ne se connaissent pas, la spécification de l'interface devient donc le médiateur qui permet aux deux parties de travailler ensemble [Mac 04].

Syntaxiquement, les interfaces sont similaires aux descriptions de classe d'objets, elles contiennent des attributs et des opérations. Sémantiquement, les interfaces fournies spécifient les services ou les capacités qu'offrent le composant pour les autres composants [Nin 97].

I.2.1.2 L'implantation du composant

L'implantation du composant est la réalisation exécutable du composant, obéissant aux règles du modèle de composant (défini dans I.2.3). L'implantation du composant peut être fournie en code compilable, en forme binaire, …etc.

Un composant ne contient pas uniquement des classes. Il peut même ne pas en contenir du tout. Il peut :

– contenir simplement des procédures classiques et des variables globales ;

– être écrit entièrement dans une approche de programmation par fonctions ;

– utilise un langage assembleur ;

I.2.1.3 Les différentes abstractions d'un composant

L'abstraction d'un composant correspond à la visibilité de son implantation. On trouve trois sortes d'abstractions [Mac 04] [Buc, Wec 97]:

***Blackbox (Boîte noire)* :** Le client ne connaît aucun détail au-delà des interfaces et de leurs spécifications. La réutilisation d'une boîte noire se réfère au concept de réutilisation de l'implantation en reliant les interfaces entre elles.

***Whitebox (Boîte blanche)* :** L'implantation d'une boîte blanche est entièrement disponible et peut donc être étudiée afin d'augmenter sa compréhension. La réutilisation d'une boîte blanche s'apparente à l'utilisation d'un fragment de logiciel à travers ses

interfaces, tout en tenant compte de la compréhension acquise par la connaissance de l'implémentation, ainsi qu'une éventuelle modification.

Graybox (Boîte grise) : il s'agit d'un niveau de transparence intermédiaire entre les composants de type boite noire et ceux de type boite blanche. Les détails d'implémentation peuvent être révélés pour comprendre la réalisation du composant, mais ne peuvent être sujets à des modifications émanant de l'extérieur [Ous, al 05].

I.2.1.4 Les propriétés du composant

Les composants doivent vérifier certaines propriétés : ils doivent être encapsulables, descriptifs, remplaçables et extensibles. Ces quatre propriétés doivent être respectées par les développeurs [Ha, Tra 05].

Un composant doit être encapsulable

L'encapsulation signifie que le code ou l'implémentation du composant est caché à son utilisateur. L'utilisateur doit faire usage de l'une des interfaces du composant pour accéder au code ou l'implémentation. L'interface présente à l'utilisateur ce que le composant offre comme services. Ceci fait souvent référence à la séparation de «quoi» du «comment». Le plus grand avantage de l'encapsulation est que le composant est en mesure de changer ou de modifier son implémentation, tout en maintenant la même interface vers ses utilisateurs.

Un composant doit être descriptif

Puisque les utilisateurs ne peuvent pas interagir avec l'implémentation ou le code du composant, sauf par le biais des interfaces, l'information sur le composant doit être facilement obtenue et comprise. Elle doit décrire les trois parties principales du composant : son interface(s), sa mise en œuvre (s) et son déploiement.

Un composant doit être remplaçable

Avec les deux propriétés précédemment définies, qui sont l'encapsulation du composant et sa description, il est possible de remplacer un élément avec un autre tant que les interfaces restent les mêmes.

Un composant doit être extensible

Cette propriété signifie qu'il est possible d'agrandir ou d'élargir l'éventail des services du composant sans affecter les utilisateurs. Il existe deux types d'extensions :

- **Ajout d'interfaces :** Le développeur du composant peut étendre les services par l'ajout ou la modification de son implémentation. Les interfaces ne peuvent pas être changées puisqu'elles sont déjà en service. De nouvelles interfaces doivent être développées pour les nouveaux services.
- **Délégation de responsabilité** : Un composant utilisateur n'a pas accès à l'implémentation. Un nouveau composant peut être créé pour offrir de nouveaux services. Ce nouveau composant délègue les responsabilités de tous les services existants au composant original.

I.2.2 Une architecture

L'architecture d'un composant logiciel (voir *Figure 1.1*) spécifie ses entrées et ses sorties afin de faciliter la description de son comportement (services offerts) quelque soient les langages de programmation utilisés [Had 03].

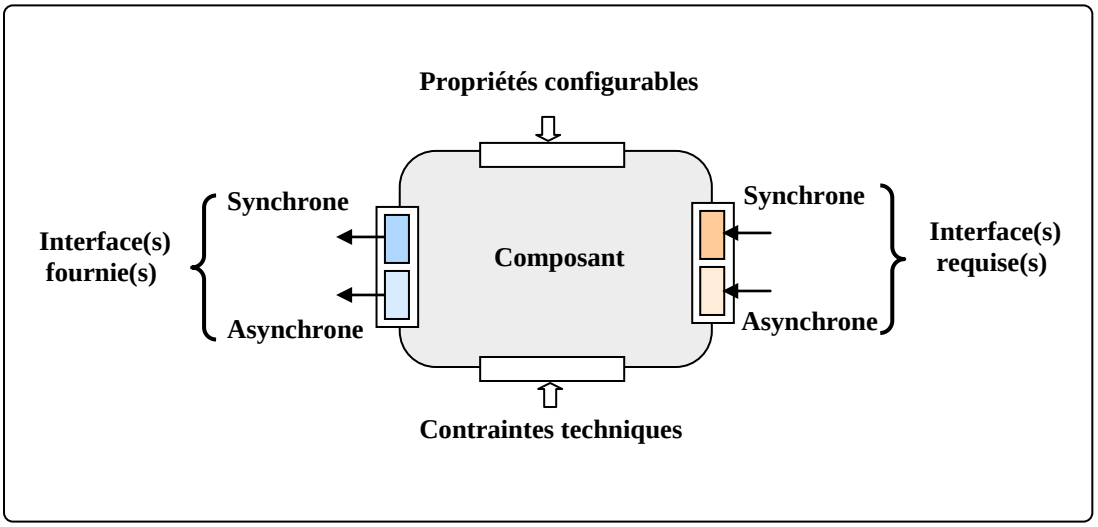

Figure 1.1 : Architecture d'un composant logiciel [Had 03]

Comme le présente la figure ci-dessus, un composant logiciel possède principalement les trois éléments suivants :

-Les interfaces fournies (*sorties*) et **les interfaces requises** (*entrées*), en mode synchrone ou asynchrone : ce qui définit ses moyens mis en œuvre pour coopérer. Ces moyens peuvent être des opérations (des fonctions promises aux clients) ou des propriétés.

-Les propriétés configurables : généralement, ce sont des attributs. Elles permettent d'adapter et de personnaliser le composant dans des contextes d'exécutions spécifiques.

-Les contraintes techniques (QoS : *Quality of Services*), qui peuvent être : la sécurité, la persistance, les transactions… etc.

I.2.3 Un modèle de composants

Consiste en un ensemble de conventions à respecter dans la construction et l'utilisation des composants. L'objectif de ces conventions est de permettre de définir et de gérer d'une manière uniforme les composants. Elles couvrent toutes les phases du cycle de vie d'un système d'information à base de composants : la conception, l'implantation, l'assemblage, le déploiement et l'exécution [Kha 05].

I.3 Les modèles de composants

Plusieurs efforts de standardisation sont actuellement en cours, et ont conduits à l'élaboration de plateformes complètes de composants. Les modèles qui ont le plus marqué leur présence sur le monde industriel sont : les modèles EJB (Entreprise JavaBeans) de la société Sun Microsystems, CCM (CORBA (Common Object Request Broker Architecture) Component Model) du consortium OMG (Object Management Group) et .NET de la société Microsoft. Les modèles de composants sont également connus sous le nom de *Structure d'accueil* ou de *Standard de développement*. Nous présentons quelques modèles de composants tout en adoptant la classification proposée dans [Ozc 07] et [Ous, al 05].

I.3.1 Modèle de composants industriels

I.3.1.1 Modèle de composants de présentation

Les modèles à composants de présentation sont principalement dédiés au développement rapide d'interfaces graphiques et n'implantent pas les notions de conteneur : espace de référence dans lequel on dépose et on assemble des composants logiciels qui peuvent ainsi y interagir, et qui sert de moule pour créer des applications [Dic 08]. Le représentant principal de ce type de modèle est le modèle des JavaBeans.

I.3.1.1.1 JavaBeans

Principalement dédiés aux applets et aux applications côté client en Java. Ces applications Java peuvent être construites à partir des composants provenant de multiples constructeurs et ont la particularité, grâce aux interfaces java, d'avoir la spécification et l'implémentation séparées. Les Beans sont caractérisés par leur facilité de paramétrage qui simplifie leur adaptation, leur méthode de communication via des évènements et garantissent les propriétés de persistance, d'introspection et de configuration. Ces composants sont développés selon un moule assez restrictif mais qui permet de les manipuler facilement avec des outils visuels [Mos 02], [Col 04a].

I.3.1.2 Modèle de Composants métiers

Les composants métiers sont des composants logiciels gros grain, destinés aux systèmes d'information distribués. Afin d'en faciliter le développement, la distribution et la réutilisation de ces composants, leurs conceptions sont basées entre autres sur le principe de séparation du code applicatif (ou fonctionnel) du code technique (ou non-fonctionnel). On distingue principalement trois modèles à composants métiers les plus utilisés dans l'industrie : EJB, CCM et .NET. Chacun de ces modèles correspond à une vision particulière de la programmation.

I.3.1.2.1 Entreprise JavaBeans (EJB) de Sun

Les composants proposent une solution pour un développement rapide d'applications déployées sur Internet. La spécification EJB ne précise pas la manière d'implanter un composant, mais plutôt un moule de conception utilisé par le développeur, pour définir des blocs fonctionnels et des interfaces précises. Le composant obtenu est réutilisable dans n'importe quel environnement Java, configurable de façon limitée et simple. EJB repose sur une architecture multi-niveaux composée de [Mos 02] :

- Un serveur EJB : Fournit aux conteneurs les services de bas niveau tels que la sécurité et la persistance. Il intercepte les requêtes des clients.
- Un conteneur EJB : Fournit l'environnement d'exécution d'EJB et joue le rôle d'intermédiaire entre l'EJB et le serveur EJB.
- L'interface locale : Permet de gérer le cycle de vie d'un EJB.
- L'interface distante : Contient les méthodes "métier" qu'encapsule un EJB. Elle représente la logique du composant.

EJB utilise d'autres technologies Java telles que :

-JNDI (Java Naming Directory Interface) pour l'identification d'objets ;

- JTS (Java Transaction Server) pour la gestion de transactions et ;

- JDBC (Java Data Base Connectivity) pour l'accès aux bases de données.

I.3.1.2.2 CORBA Component Model (CCM) d'OMG

Les composants CCM sont destinés à la construction et le déploiement de composants côté serveur, pour le développement d'applications d'entreprise. Les composants CCM résident dans le serveur, exposent leurs services sous forme d'interfaces écrites en OMG IDL[4] et utilisent CORBA *(Common Object Request Broker Architecture)* comme infrastructure de communication. L'architecture de CCM ressemble à celle des EJB mais offre une plus grande indépendance des langages de programmation.

Les ports CCM permettent aux composants d'interopérer avec les clients et les autres composants. Il y a quatre types de ports :

- Facettes : Des interfaces exposées par le composant qui peuvent être invoquées de façon synchrone ou asynchrone par les clients ou par d'autres composants.
- Attributs : Des valeurs utilisées pour la configuration d'un composant.
- Récipients : Servent à l'assemblage avec d'autres composants à qui on délègue certaines opérations.
- Evènements : Actions déclenchées suivant certaines conditions. Un composant peut émettre des évènements (event sources) ou en recevoir (event sink).

L'architecture CCM repose sur :

- Des conteneurs qui hébergent les composants.
- Des interfaces locales gérant le cycle de vie des composants.
- Les interfaces de rappel (callback interfaces) représentent les liens entre le composant et son conteneur.

CCM offre aussi un ensemble de services dont la gestion de transactions, la sécurité, la notification (dispatching des évènements) et la persistance [Mos 02].

[4] OMG IDL: Object Management Group Interface Definition Language

I.3.1.2.3 Les composants .NET de Microsoft

Le modèle .NET, à l'origine de Microsoft, fait suite aux modèles de composants précédemment proposés par la société Microsoft : COM (Component Object Model), DCOM (Distributed Component Object Model) et COM+. C'est une plateforme visant à simplifier le développement d'applications hautement distribuées sur le web. Ce modèle vise également à unifier la programmation des produits "de bureau" et les services Internet, ainsi que les produits destinés aux serveurs vendus par Microsoft.

L'infrastructure .NET est basée sur plusieurs couches dont :

- Les classes de base (Base Classes) : une API fournissant de manière centralisée des fonctionnalités telles que la gestion de chaînes de caractères, de processus…etc.
- XML et données (XML and Data) : fournit un accès unifié aux bases de données du marché.
- WinForms : destinée au développement d'IU (Interfaces Utilisateurs) pour les applications de bureau.
- WebForms et ASP.NET : Une technologie permettant le développement d'applications web d'une manière similaire aux applications locales.

I.3.3 Modèle de Composants académiques

La communauté académique a proposé ces dernières années un nombre considérable d'ADL pour décrire et raisonner sur les architectures logicielles à base de composants [Ous, al 05].

I.3.3.1 Les ADLs (*Architecture Description Language*)

Les langages de description d'architecture sont des langages formels. Ils représentent un support pour la description de la structure de l'application. Ils permettent la définition d'un vocabulaire précis et commun pour les acteurs devant travailler autour de la spécification liée à l'architecture (architectes, concepteurs, développeurs, intégrateurs et testeurs). Les ADLs spécifient les composants de l'architecture de manière abstraite, sans entrer dans les détails d'implantation. Ils définissent de manière explicite les interactions entre composants d'un système et fournissent un support de modélisation pour aider les concepteurs à structurer et composer les différents éléments. Les ADLs offrent aussi des facilités de réutilisation des composants et des moyens de description de la composition. Ceci est assuré par la description

des dépendances entre composants et les règles de communication à respecter. Parmi les représentants actuels nous citons : Fractal, Darwin, C2, Wrigh…etc.

Les concepts de base d'une description architecturale sur lesquels s'accordent l'ensemble des travaux portant sur les ADLs reposent sur les composants, les connecteurs, les configurations architecturales et les styles architecturaux [Acc 02] [Ous, al 05].

Les composants : Un composant est une unité de calcul ou de stockage à laquelle est associée une unité d'implantation. Le composant est un lieu de calcul et possède un état. Il peut être simple ou composé. Sa taille peut aller de la fonction mathématique à une application complète.

Les connecteurs : représentent une abstraction définissant l'interaction entre des composants. L'interface de ceux-ci définit un ensemble de rôles. Chaque rôle caractérise les rôles joués par un composant participant à l'interaction.

Les configurations architecturales: la configuration est la description d'une composition de composants et de connecteurs. Les composants sont liés aux connecteurs. Cette liaison s'établit en reliant le port du composant au rôle du connecteur.

Les styles architecturaux : le style d'une architecture permet de décrire un ensemble de propriétés communes à une famille de systèmes. Il permet de décrire un vocabulaire commun en définissant un ensemble de types de connecteurs et de composants et un ensemble de propriétés et de contraintes partagées par toutes les configurations appartenant à ce style.

Nous présentons ci-dessous un exemple d'ADL nommé Fractal.

I.3.3.1.1 FRACTAL

Fractal est issu du laboratoire DTL/ASR de FTR&D, maintenu actuellement par le consortium ObjectWeb (*http : //www.objectweb.org*) [Dep 05]. C'est un modèle hiérarchique qui distingue deux sortes de composants, les primitifs et les composites. Les composants composites sont formés à partir d'autres composants primitifs ou composites. Ce qui permet une construction récursive qui s'arrête au niveau des composants primitifs et qui sont directement programmés dans un langage de programmation donné. Le modèle FRACTAL permet le partage de composants, c'est-à-dire que plusieurs composites peuvent contenir une même instance d'un composant, que ce soit un composite ou un primitif. Les seuls points d'accès des composants sont leurs interfaces. Un composant peut avoir plusieurs interfaces. Deux types d'interfaces sont distingués : les interfaces clients permettent d'émettre des opérations vers d'autres composants alors que les interfaces serveurs permettent d'en

recevoir. En plus de la nature client ou serveur, des propriétés de contingence (obligatoire ou optionnelle) et de cardinalité (acceptation de connexions simples ou multiples) sont attribuées aux interfaces. Ces propriétés permettent d'améliorer le niveau de description de l'architecture logicielle et enrichissent entre autres les vérifications architecturales que l'on peut implanter [Ozc 07].

La spécification de Fractal définit un modèle de composant, mais aussi un Framework d'interface de programmation. Nous présentons uniquement le modèle de composant, qui se base sur cinq concepts clés : composant, contenu, contrôleur, interface et liaison [Dep 05].

– **Composant :** un composant Fractal est formé de deux parties : un contrôleur et un contenu.

– **Contenu :** le contenu d'un composant est constitué d'un nombre fini de composants, appelés sous composants. Ces sous-composants sont sous le contrôle du composant englobant. Le modèle de composant Fractal est récursif grâce à cette notion de sous-composant, la récursivité s'arrête au niveau d'un composant primitif.

– **Contrôleur :** la partie contrôleur du composant Fractal est destinée à contrôler le comportement du composant, en interceptant les signaux entrants et sortants d'un composant, en donnant une représentation de son état interne, ou encore en suspendant ou arrêtant l'activité d'un de ses sous-composants.

– **Interface :** un composant Fractal interagit avec son environnement par le biais d'opérations via des points d'accès appelés interfaces. La visibilité d'une interface est définie par le contrôleur : il peut rendre visible une interface d'un sous-composant depuis l'extérieur ou bien la cacher. Les opérations peuvent être de deux types : invocation avec ou sans résultat en retour. Les interfaces peuvent être de deux types : client ou serveur. Le nombre d'interfaces d'un composant peut être variable tout au long de son cycle de vie.

– Liaison : une liaison est une connexion entre deux ou plusieurs composants. Le modèle Fractal définit deux types de liaisons : les liaisons dites primitives et les liaisons dites composites. Une liaison primitive est réalisée au niveau du langage d'implémentation. Elle peut être établie entre une interface client et une interface serveur. Une liaison composite est une combinaison de composants et de liaisons primitives : cela implique qu'une liaison composite est aussi un composant.

Plusieurs plates-formes de développement Fractal existent [Cou 06]. On peut citer en particulier, THINK pour le langage C, ProActive pour le langage Java, FracTalk pour Smalltalk, Plasma pour C++ …etc.

I.4 Conclusion

Nous avons présenté dans ce chapitre les concepts de base liés à la programmation à base de composants, avec tout d'abord la notion de composant logiciel et les différentes définitions liées à ce terme, suivi d'un aperçu des principales approches sur le marché industriel des composants, dont les leaders sont l'OMG (CCM), Sun (EJB) et Microsoft (.Net).

La notion de composant ou de brique logicielle est intrinsèquement liée à la notion de réutilisation. Le prochain chapitre sera consacré à la réutilisation et ses différentes facettes.

Chapitre II

L'ingénierie de la réutilisation

II.1 Introduction

Le développement des systèmes logiciels complexes avec les approches usuelles dont lesquelles la construction d'un nouveau système parte de rien « From scrach » est plus difficile, plus coûteux et moins fiable. Selon B. W. Boehm dans [Boe 87], le facteur qui puisse en pratique faire augmenter la productivité des informaticiens est de réduire la quantité de code nécessaire pour réaliser une fonctionnalité donnée, en d'autres termes « Programmer moins pour programmer mieux ». L'approche permettant d'atteindre cet objectif est celle de construire un système en réutilisant des composants existants ayant été produits à l'occasion de développements antérieurs [Gac, Ser 03].

Le concept de réutilisation a été évoqué par M. McIlroy [McI 68], dans une conférence de l'OTAN consacrée à la crise de logiciels où il a proposé une solution basée sur la réutilisation de bibliothèques de composants logiciels. Depuis, cette idée a évolué et la réutilisation est devenue une discipline et un thème de recherche à part entière.

II.2 L'ingénierie de la réutilisation

Les composants logiciels et la réutilisation logicielle se complémentent parfaitement, l'utilisation des composants pour le développement des applications conduit automatiquement à la réutilisation logicielle (mais pas suffisante). La recherche dans la réutilisation évolue automatiquement vers la composition logicielle et par conséquent vers les composants [Sam 97].

La notion de réutilisation logicielle dans la littérature a été traitée selon différents points de vue.

Cooper la définie comme étant la faculté qu'ait un composant logiciel préalablement développé, à être utilisé de nouveau ou à être utilisé répétitivement dans des applications autres que celles pour laquelle il a été développé à l'origine, avec ou sans modifications [Alm, al 07].

Krueger dans [Kru 92] l'a définie comme étant le processus de création de système logiciel à partir d'unités logicielles existantes plutôt que de construire à partir de zéro ou de rien.

Selon M McIlroy, la problématique de recherche dans la réutilisation met en relief deux aspects complémentaires. Celui qui vise à développer des composants prêts à être réutilisés, ce qui correspond à l'aspect « infrastructure » de la réutilisation, appelé l'ingénierie des composants pour la réutilisation « design for reuse ». Celui qui vise à développer des

applications en exploitant ces composants réutilisables, ce qui correspond à l'aspect « opérationnel » de la réutilisation, appelé l'ingénierie des applications par la réutilisation « design by reuse ».

II.2.1 L'ingénierie des composants pour la réutilisation

Selon Gacemi [Gac, Ser 03] et Chikhi [Chi 98], La problématique de génération des composants réutilisables demande de résoudre trois sous problèmes qui s'enchaînent au cours de la mise en œuvre de ce processus, à savoir : l'identification, la spécification (ou la représentation) et l'organisation des composants réutilisables. Ceci est illustré sur la figure 2.1.

Selon Kim [Kim 05], cette phase de génération de composants est considérée comme les pré-conditions pour une réutilisation basée composant. Elle propose de répondre à des questions du genre : Comment organiser ces composants, comment les stocker, quelle forme doivent-ils avoir, où les trouver…etc.

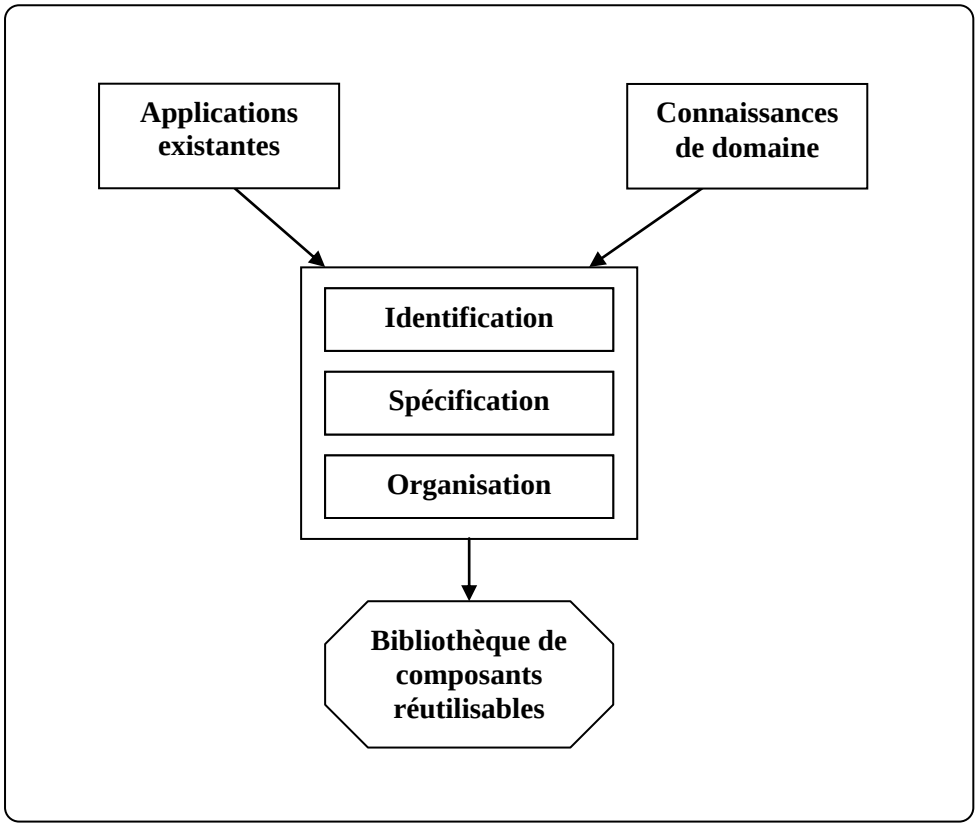

Figure 2.1: Ingénierie des composants réutilisables « Design for reuse » [Gac, Ser 03]

II.2.1.1 L'identification

L'identification des composants réutilisables a pour objectif de trouver les connaissances invariantes entre plusieurs applications. Il s'agit ici, à partir de l'ensemble des buts, activités et objets définis dans le modèle de domaine, d'exprimer les différents problèmes de modélisation conceptuelle que l'on aura à résoudre. Chaque problème de modélisation conceptuelle définit un composant dans lequel on intègre une solution et le contexte qui discrimine cette solution par rapport aux autres solutions du même problème. On distingue deux approches qui visent l'identification [Gac, Ser 03] [Chi 98] :

Approche par rétro-ingénierie (ou ingénierie inverse) **:** elle tire son origine de l'analyse du matériel, où la pratique qui consiste à extraire les schémas de produits finis est une activité courante. L'approche par rétro-ingénierie logicielle est donc basée sur l'analyse et le recyclage des produits logiciels existants pour en extraire les composants et leurs interactions, ainsi que pour la définition d'une représentation du système dans une forme différente ou dans un niveau d'abstraction plus élevé [Chi, Cro 90].

Approche par l'analyse du domaine : l'analyse du domaine énonce ce qui peut être fait dans une série de problèmes dans un domaine, elle est définit comme l'activité d'identification des objets et des opérations des classes, des systèmes similaires dans un domaine particulier [Nei 80].

II.2.1.2 La spécification

Les composants logiciels doivent être clairement et proprement spécifiés avant d'être classés dans la bibliothèque de composants, afin de les réutiliser le plus possible. Deux aspects sont à distinguer, à savoir [Alm, al 07]:

-La spécification qui consiste à définir les différentes informations et connaissances utiles et indispensables lors de l'assemblage et l'intégration des composants, par exemple les contraintes à respecter. On peut assimiler cet aspect à un mode d'emploi du composant ;

-Et la spécification qui consiste à définir les connaissances qui servent comme critères de recherche de composants logiciels dans les bibliothèques de composants. Plusieurs approches ont été présentées pour résoudre le problème de la spécification des composants pour la recherche. Nous citons à titre d'exemple, celles basées sur les mots-clés ou celle basées sur les facettes.

II.2.1.3 L'organisation

Contrairement à la spécification qui s'intéresse à un composant pris individuellement, l'organisation s'intéresse à l'ensemble des composants et les relations sémantiques entre ces composants, permettant ainsi de traduire un processus de recherche.

Cette étape est utile dans la mesure où elle permet de faciliter la recherche et la sélection d'un composant devant satisfaire les spécifications des besoins d'un problème de développement donné. Cette organisation est nécessaire pour deux raisons :

• Les solutions d'une même intention sont éparpillées au sein de plusieurs composants, en fonction du contexte dans lequel leur réutilisation est pertinente. Il apparaît donc nécessaire de fournir un mécanisme permettant de relier entre eux les composants associés à une même intention ;

• La réutilisation des solutions fournies par certains composants peut nécessiter la résolution d'autres intentions. Or, la structure seule des composants de domaine ne permet pas de mettre en corrélation une solution et les éventuels problèmes qu'elle peut soulever. Ici aussi, il faut proposer un mécanisme permettant de lier une solution fournie par un composant aux composants dont l'intention porte sur un fragment de cette solution.

Deux types de liens peuvent être distingués pour organiser les composants :

• Ceux permettant de rassembler les composants répondant à un même problème et de les organiser en fonction de leur contexte. Nous avons :

- les liens de raffinement : ont pour but, d'une part de rassembler les composants répondant à la même intention et, d'autre part de hiérarchiser ces composants en fonction de leur contexte de façon à en faciliter la recherche et la sélection lors de la manipulation.
- les liens d'alternatives : Cette forme de lien offre ainsi un mécanisme permettant de repérer immédiatement au sein d'un ensemble de composants quels sont les composants offrant des alternatives de solution par rapport à un composant particulier sélectionné.

• Ceux permettant de lier les solutions des composants aux intentions d'autres composants, permettent ainsi de supporter l'assemblage de composants. Nous avons les liens d'utilisation et les liens d'extension [Ram 01].

II.2.2 L'ingénierie des composants par la réutilisation

Après la production des composants réutilisables, leur utilisation n'est pas systématique mais passe par la résolution de trois sous problématiques (la Recherche, l'adaptation et l'intégration) qui sont plus ou moins difficiles selon le produit à réutiliser [Gac, Ser 03], [Puj, al 04], [Chi 98] et [Puj, al 06].
(Voir figure 2.2)

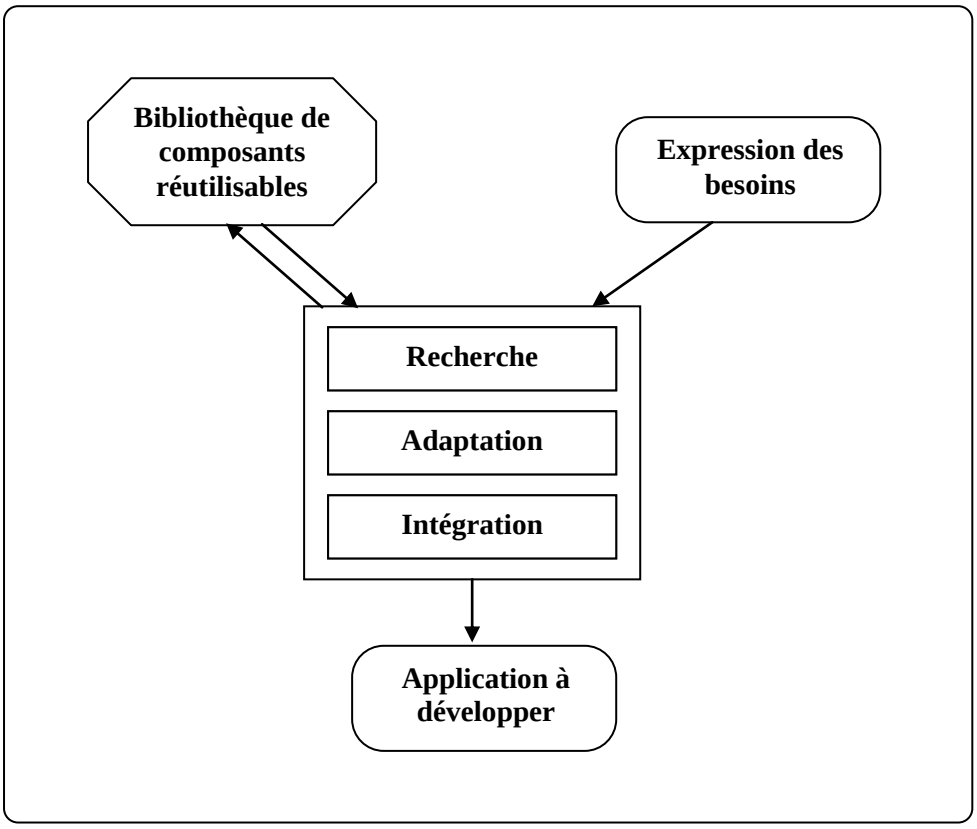

Figure 2.2 : Ingénierie par la réutilisation « Design by reuse » [Gac, Ser 03]

II.2.2.1 La recherche et la sélection

La recherche consiste à trouver un composant qui satisfait les critères de recherche et qui répond à un problème particulier de développement.

Il existe fondamentalement deux approches de systèmes de recherche de composants : l'approche par navigation et l'approche par requêtes.

-Dans l'approche par navigation, le système de recherche d'information exploite les liens sémantiques, qui peuvent exister entre les composants, pour faire la navigation ;

Dans l'approche par requêtes, le principe de base consiste à faire correspondre une requête à une collection de composants et à retourner à l'utilisateur les composants pertinents. Le succès de ce type de systèmes dépend en partie de la qualité et de la quantité d'informations associées à la requête et à la spécification des composants présents dans la base de composants. Les systèmes à base de requêtes sont classés, selon les techniques de recherche de composants qu'ils adoptent, comme suit :

-classification externe : indexe une représentation externe des composants. Cette représentation est souvent préparée manuellement sous forme textuelle (par exemple, une documentation en langage naturel).

-recherche structurelle : exploite les propriétés structurelles des composants comme les signatures et les spécifications des services offerts par les composants.

-recherche comportementale : exploite la propriété d'exécutabilité des composants logiciels pour les classer. La classification se fait en mesurant certains critères dynamiques comme le temps d'exécution ou la mémoire consommée, ou encore en utilisant les traces d'exécution.

Une perte de l'information est souvent ressentie au niveau de formulation de la requête, puisque l'utilisateur ne réussit pas toujours à exprimer ses besoins correctement, il est donc nécessaire de trouver des mécanismes permettant d'aider l'utilisateur à bien formuler sa requête. Une autre difficulté au niveau de cette étape réside dans le fait que dans la plupart des situations, il n'y a pas qu'un seul composant candidat qui répond au besoin, mais plusieurs. Ceci est dû à la non-conformité exacte entre les spécifications des composants et les critères de recherche. Dans ce cas, il est nécessaire de mettre en œuvre un mécanisme de sélection par contexte, qui permet d'opérer une sélection encore plus fine sur cet ensemble de composants, afin d'en dégager le composant le mieux adapté au cadre de résolution exprimé.

Certains auteurs considèrent la sélection de composant comme une étape à part entière dans le processus de développement par la réutilisation, d'autres l'inclut dans l'étape de recherche [Puj, Ram 04].

II.2.2.2 L'adaptation

Pour permettre au composant acquis d'être utilisé dans des contextes différents ou de ne pas être en situation conflictuelle avec d'autres composants, une adaptation de ce dernier s'avère nécessaire**.**

Il existe une grande diversité dans les approches proposées pour l'adaptation des composants, certaines se concentrent sur l'adaptation des services offerts par les composants logiciels et d'autres se consacrent à l'adaptation structurelle du composant logiciel. La première impose l'accès à l'implémentation du composant, pour effectuer les modifications nécessaires afin d'adapter le service. Quant à la dernière, elle consiste à modifier la structure du composant tout en préservant ses services et son comportement. Elle peut concerner la structure externe d'un composant (ses ports, ses interfaces…etc.) et / ou sa structure interne (ses sous-composants, ses classes d'implémentation, ses connexions internes…etc.). Ainsi, l'adaptation structurelle externe peut être utilisée pour faciliter l'assemblage de composants logiciels ayant des structures externes hétérogènes [Bas, al 07], [Her 05].

L'adaptation du composant peut être réalisée par le client ou le fournisseur du composant, On peut distinguer :

- Une approche qui consiste à fournir des techniques qui permettent au client d'adapter un composant : seul le client du composant produit un effort pour l'adaptation.

- Une approche où le fournisseur du composant autorise une souplesse d'utilisation de son composant par le client. L'adaptation est facilitée car prévue par le concepteur du composant : l'effort d'adaptation est produit/supporté à la fois par le client et le fournisseur.

II.2.2.3 L'intégration

Dans le développement à base de composants, l'application logicielle est construite par une collection de composants logiciels interconnectés. Pour mettre en évidence la structure de l'application, l'assemblage des composants peut être décrit à l'aide des langages de description d'architectures, ADL (Architecture Description Language). Les ADL permettent de décrire une architecture logicielle en se basant sur la notion de configuration. La configuration permet de décrire la structure globale d'un logiciel, c'est-à-dire l'assemblage des composants de ce logiciel sous forme de graphes de connexion formés de composants et

de connecteurs. Elle permet aussi de décrire son comportement (création, utilisation et suppression des instances).

Le problème d'assemblage des composants (ou intégration) reste un problème des plus délicats en technologie composant. La notion d'assemblage est définie soit directement pour les composants ayant des interfaces de même type, soit via des connecteurs logiciels. Les connecteurs sont des entités architecturales de communication qui modélisent de manière explicite les interactions (transfert de contrôle et de données) entre les composants. Ils contiennent des informations concernant les règles d'interactions entre les composants. Ces derniers sont proposés principalement pour faciliter l'assemblage des composants en jouant le rôle d'intermédiaires entre les composants non-compatibles. Ainsi, l'objectif des connecteurs est d'atteindre une meilleure réutilisabilité lors de l'assemblage des composants, et faciliter ainsi le développement d'applications à base de composants logiciels. Les composants s'occupent du calcul et du stockage tandis que les connecteurs s'occupent de gérer les interactions (communication/coordination) entre les composants [Had 03].

II.3 Bibliothèque de composants réutilisables

Les bibliothèques logicielles représentent des dépôts où les composants logiciels sont stockés et recherchés, en tant que tels, ils représentent une source précieuse pour l'ingénieur logiciel [Mil, al 97]. Les bibliothèques de composants sont des éléments clés dans les environnements de développement à base de composants et de réutilisation de composants. Leur efficacité croît lorsque les développeurs et les concepteurs disposent de bibliothèques riches en composants testés et validés, ce qui augmente la fiabilité et diminue le temps de développement des systèmes d'information résultants. Malheureusement, mettre simplement à disposition des développeurs des bibliothèques de composants de taille importante n'augmente pas forcément la productivité. Le concepteur a besoin de techniques performantes de recherche et de classification et de spécification de composants [Kha 05].

II.4 Conclusion

Nous avons présenté un état de l’art axé sur la réutilisation, particulièrement sur le développement de composants et le développement d'application à base de ces composants. Avec ces grandes tendances, la réutilisation présente une solution intéressante à la crise du logiciel.

Intuitivement, la réutilisation permet de réaliser des gains puisque les composants utilisés ne seront plus produits à partir de zéro. De plus, la qualité globale des logiciels peut être améliorée a conditions que les composants réutilisés soient eux mêmes de bonne qualité.

La problématique de la réutilisation met en évidence plusieurs axes de recherche qui sont en pleine effervescence, allant de la représentation et la spécification des composants jusqu'à leur assemblage, passant par leur recherche et leur adaptation. Le chapitre suivant abordera à la fois la spécification des composants logiciels, leur recherche et leur sélection.

Chapitre III

La recherche des composants logiciels

III.1 Introduction

Dans le but d'assurer la réutilisation et l'assemblage des composants logiciels pour le développement des systèmes, retrouver ces composants qui vont représenter les briques de base devient une condition sin qua non. Pour y parvenir, une recherche pertinente s'avère indispensable. La recherche des composants logiciels s'interfère avec la recherche d'information, puisque dans la plupart des situations, les composants sont décrits par des représentations externes, et qui peuvent être considérées comme des documents, sur lesquels on peut appliquer les techniques de recherche d'information.

III.2 La recherche des composants logiciels

La recherche de composants logiciels se fait en plusieurs étapes, celles ci sont présentées dans le point suivant.

III.2.1 Le processus de recherche de composants

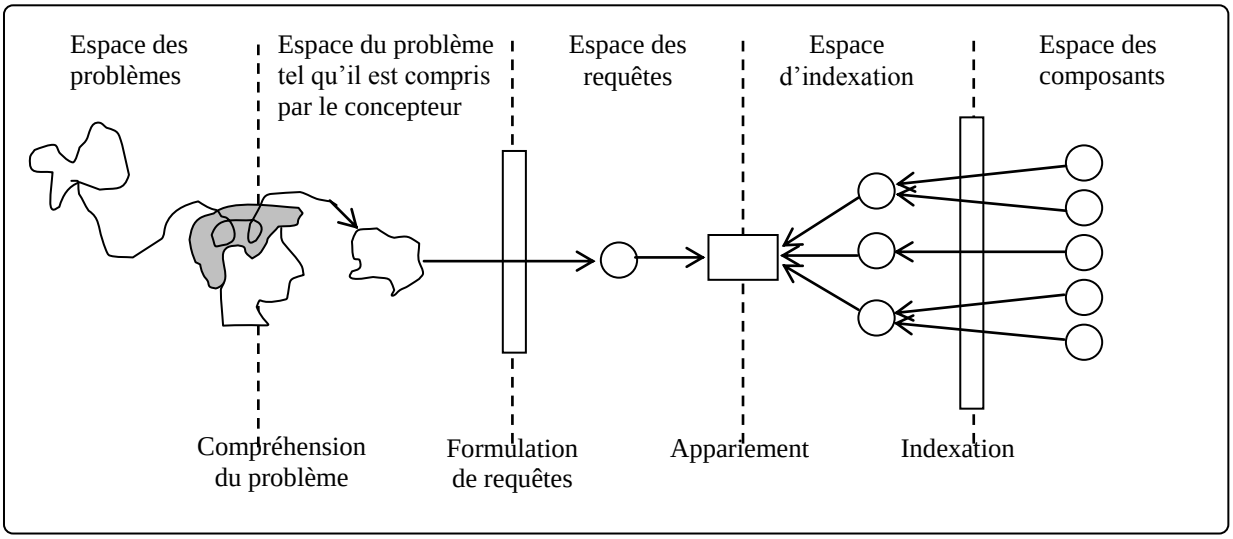

Figure 3.1 : Processus de recherche de composants [Luc, al 04]

La figure ci-dessus illustre une formalisation du processus de recherche de composants logiciels [Luc, al 04]. Cinq espaces peuvent être distingués : «l'espace des problèmes », «l'espace du problème tel qu'il est compris par le concepteur», «l'espace des requêtes », «l'espace d'indexation », et «l'espace des composants ».

Lorsque le concepteur est confronté à un problème, il le comprend à sa manière, puis il le formule par une requête, qui peut être simple tel un ensemble de mots clés ou aussi complexe qu'une spécification en langage formel. Une perte de l'information est souvent ressentie à ce niveau, puisque le concepteur ne réussit pas toujours à comprendre le problème

tel qu'il est, ou ne réussi pas à formuler sa requête correctement. Pour retrouver le composant, les informations sur les composants doivent être encodées. Ce processus de classification (aussi connu sous le nom d'indexation) peut être manuel ou automatique. Une perte d'information est également ressentie à ce niveau. La recherche elle-même consiste à comparer la requête avec l'index du composant et retourne des composants qui correspondent à la requête.

III.2.2 Techniques de recherche des composants logiciels

Plusieurs travaux ont étés proposés pour la recherche de composants logiciels réutilisables. Ceux-ci visent à faciliter la réutilisation des composants au moyen d'approches plus ou moins naturelles dont le but est de fournir des composants appropriés aux besoins des concepteurs. Ces approches peuvent être classées en trois catégories basées sur la façon dont les composants sont représentés : la classification externe, l'appariement structurel et la recherche comportementale [Kha, al 02], [Puj, Ram 04], [Kha 05].

III.2.2.1 Classification externe

La catégorie de classification externe comprend les approches qui tentent de représenter le composant par une description externe. Le processus de description, peut être automatique ou manuel. Nous distinguons [Alm, al 07]:

III.2.2.1.1 Approche par mots-clés

Vise à représenter le composant par un ensemble de mots clés, qui représente son index. Cette approche a vite montré ses limites par le fait que : lorsque la taille de la base de composants devient importante, il est difficile d'assurer la consistance de l'indexation des composants. Mais aussi, lorsque les composants sont issus de plusieurs domaines, la précision de la recherche chute rapidement.

III.2.2.1.2 Approche par langage naturel

Chaque composant est représenté par une description en langage naturel. Les informations d'indexation sont extraites en effectuant une analyse lexicale, syntaxique et sémantique sur le document (sur la description en langage naturel du composant). L'interrogation de la base de composants se fait également à travers des requêtes en langage naturel. Le mécanisme d'interprétation automatique utilisé pour l'analyse des composants utilise des techniques linguistiques pour rechercher la description la plus adéquate au besoin

exprimé par l'ingénieur. Cette méthode peut donner de bons résultats en terme de rappel mais moins bons pour la précision de la recherche, puisque les mots utilisés dans la requête de l'ingénieur peuvent avoir une diversité sémantique.

III.2.2.1.3 Approche par facettes

Proposée par P.Diaz dans [Dia 91], cette approche se compose d'une collection de facettes, qui représente les informations pertinentes pour l'identification des composants réutilisables. Chaque facette possède un nom et un espace de noms associé, appelé vocabulaire, cet espace de noms représente une collection de termes utilisés pour décrire les aspects de la facette, ceci est illustré dans la figure ci-dessous suivant le langage UML[5]. Par exemple : une facette du composant logiciel peut être le langage de programmation utilisé pour le mettre en œuvre. Cette facette peut être nommé *langage_implementation* et peut avoir l'une des valeurs suivantes : *Java, C, C++*, ...etc.

La classification à base de facettes offre des fonctionnalités qui non seulement permettent d'améliorer la recherche, mais fournit également un potentiel pour le processus de sélection et contribue à l'élaboration d'un vocabulaire standard pour les attributs des composants logiciels. Cette technique est puissante et donne de bons résultats à condition de bien choisir les facettes, et de bien définir l'espace des termes.

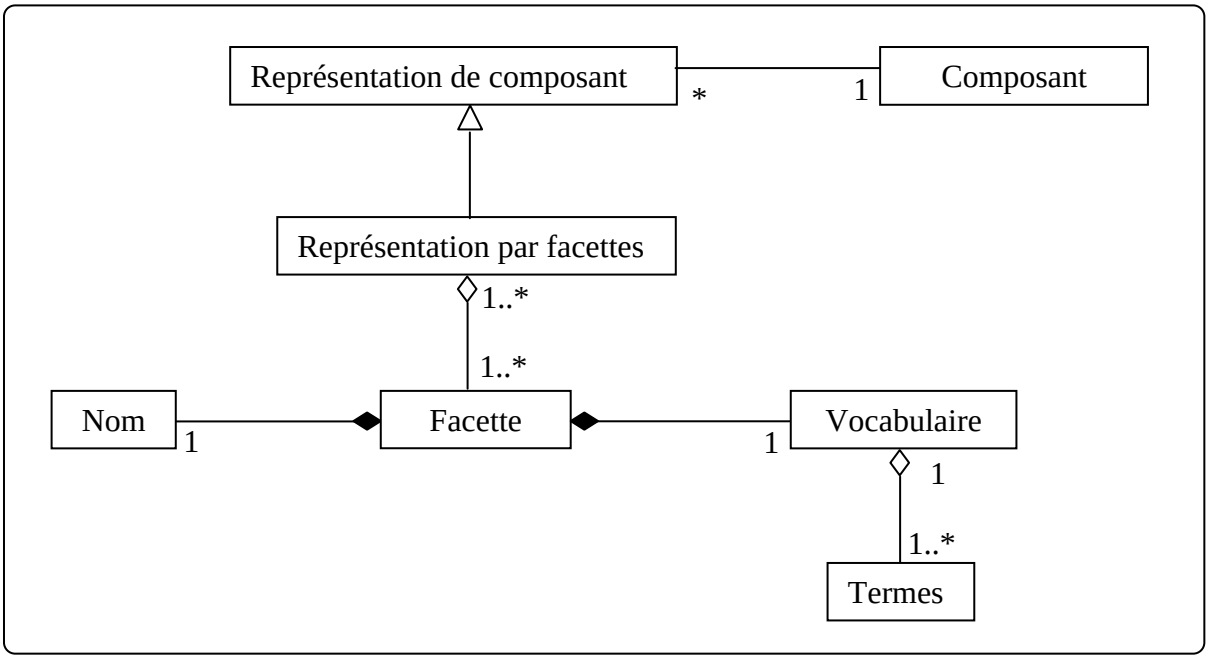

***Figure 3.2* :** Modèle de représentation par facettes [Kha 05]

[5] Unified Modeling Language

III.2.2.2 L'appariement structurel

L'appariement structurel tente de représenter le composant par l'extraction de sa structure. Deux types de descriptions sont utilisés pour déterminer l'appariement : les signatures (information de type) et les spécifications (information de comportement) [Hem, Lin 01].

III.2.2.2.1 L'appariement de signature

Représente les fonctions et les modules (ensemble de fonctions) par un ensemble de caractéristiques de signature, comme par exemple le type de variables utilisées, ceci est illustré dans la figure ci-dessous suivant le langage UML. Les approches d'appariement de signature sont en réalité un sous-ensemble des approches d'appariement de spécifications. Cette technique est très intéressante si l'ingénieur connaît a priori la forme ou le nom d'une méthode appartenant au composant qu'il cherche. Pour un acteur qui intervient dans une phase tardive du cycle de développement des SI et qui manipule des centaines de composants logiciels, cette technique peut améliorer sensiblement sa productivité.

Exemple de signature d'opération :

$$Nom\_OP = (param_1, param_2, \ldots\ldots\ldots\ldots param_n) \rightarrow Résultat$$

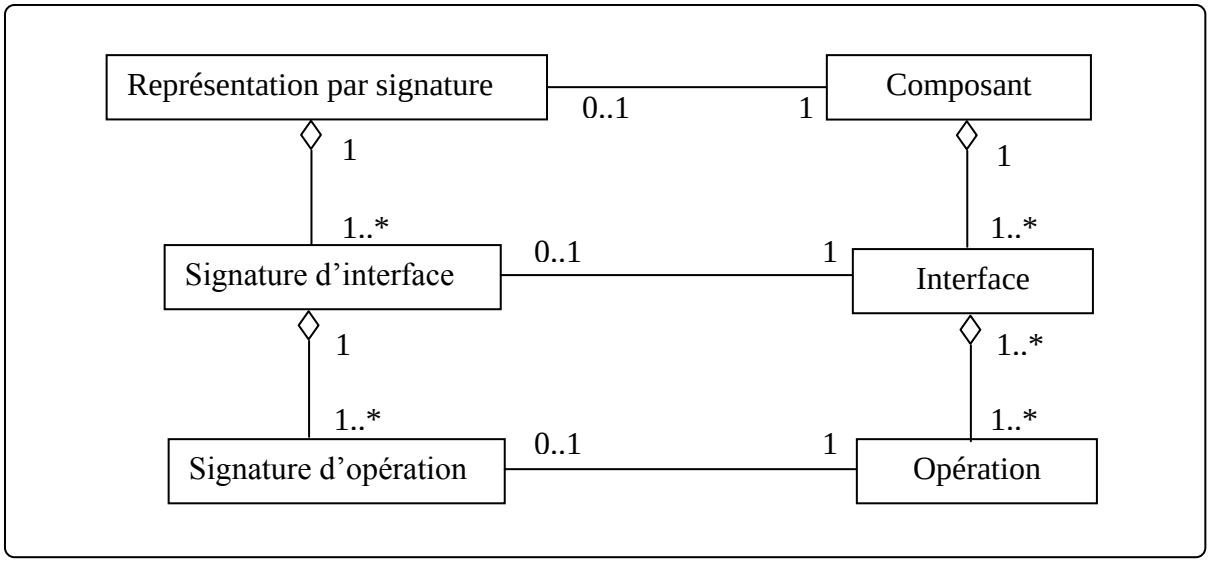

Figure 3.3 : Modèle de représentation par signatures [Kha 05]

III.2.2.2.2 L'appariement de spécification

S'appuie sur des prédicats logiques d'appariement. Les prouveurs de théorèmes sont utilisés pour apparier la requête avec les spécifications contenues dans la base de composants. La spécification d'un composant correspond à l'union de toutes les spécifications des

opérations de toutes les classes qui forment le composant. La spécification d'une opération est une paire de prédicats, une post-condition et une pré-condition qui expriment des contraintes sur les opérations. Ceci est représenté par la figure ci-dessous suivant le langage UML. Les prédicats sont exprimés dans des langages de spécifications formelles.

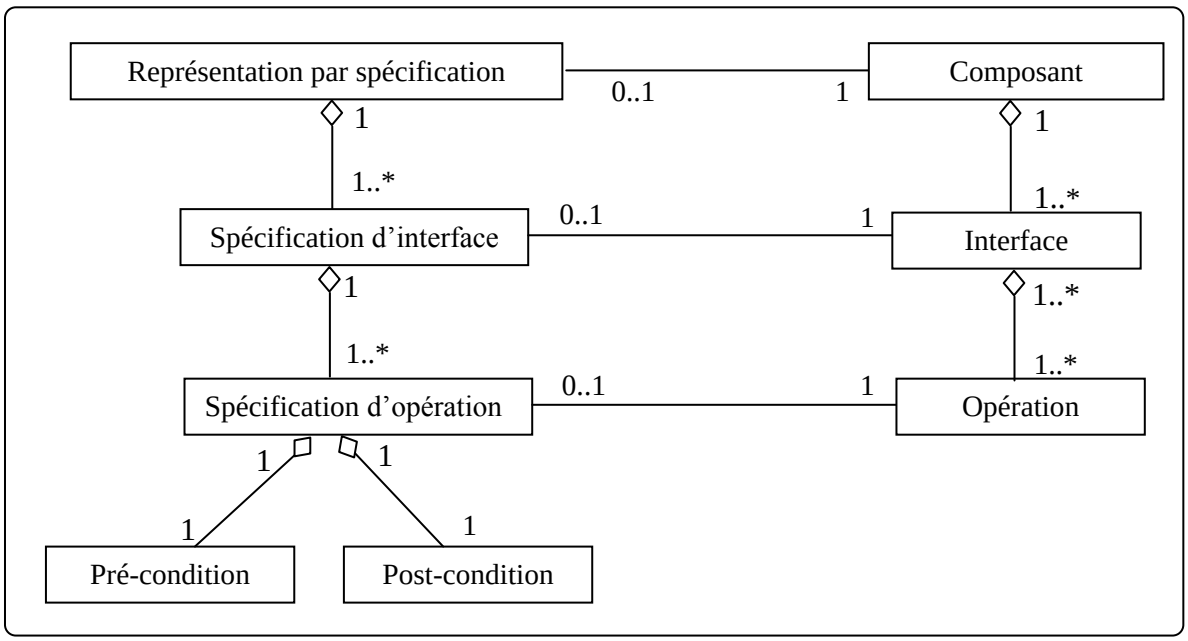

Figure 3.4: Modèle de représentation par spécification [Kha 05]

La requête de l'ingénieur se présente sous la forme d'une spécification formelle de ses besoins. Le système d'évaluation des requêtes est un moteur de preuve de théorèmes utilisant des techniques d'intelligence artificielle. Son rôle est d'essayer de prouver que deux spécifications sont équivalentes ou quasi équivalentes. Il estime une distance qui sera proportionnelle à l'effort d'adaptation nécessaire pour qu'un composant corresponde à la requête utilisateur.

Les approches de recherche de composants à base d'appariement de spécifications sont des méthodes très puissantes qui donnent de bons résultats car elles héritent de la rigueur mathématique des formalismes qu'elles utilisent. Cependant, ces approches nécessitent de très bonnes connaissances des langages de spécifications formelles comme B, Z…etc. Une deuxième difficulté vient du fait que le système d'évaluation des requêtes s'appuie sur un moteur de preuve de théorèmes. Les algorithmes de démonstration de théorèmes ont une complexité élevée. Et enfin dans la plupart des situations, la représentation interne du composant n'est pas accessible, les composants sont généralement fournis en black box, de ce fait, cette méthode n'est pas toujours réalisable.

III.2.2.3 Recherche comportementale

Les approches de recherche de composants comportementales exploitent la propriété d'exécutabilité des composants logiciels. Le test du composant logiciel, avec l'activation de ses opérations avec différentes valeurs de paramètres, renvoie des réponses qui sont collectées. Ces collections de réponses décrivent le comportement dynamique du composant selon les paramètres qu'il a reçu. Ces collections de réponses sont appelées comportement du composant. Pour pouvoir rechercher un composant possédant un comportement spécifique ou qui s'en approche le plus, il faut définir une relation d'ordre sur les comportements des composants. La requête de l'ingénieur se présente sous la forme d'un programme qui appelle systématiquement un sous-ensemble des opérations de chaque composant de la bibliothèque et qui récupère leurs comportements pour les comparer au comportement recherché. Seuls les composants qui vérifient le comportement indiqué dans la requête sont fournis à l'ingénieur. Cette approche se traduit par une haute précision, mais elle est limitée à de simples éléments, tels que les routines ou les morceaux de code. La performance du système est également affectée, puisque tous les composants doivent être exécutés [Alm, al 07].

Nous terminons par une étude comparative de ces différents modèles de recherche de composants, présentée dans le tableau suivant :

	Classification externe	Appariement structurel	Recherche comportementale
Nature du composant	Aucune restriction	Boite blanche	Exécutable
Description de la requête	Langage naturel / Mots-clés	Spécification formelle	Trace d'exécution
Description du composant	Langage naturel / Mots-clés	Spécification formelle	Trace d'exécution
Critère de pertinence	Informel	Formel	Formel
Coût	Coûteuse	Très coûteuse	Abordable
Difficulté d'utilisation	Facile	Très difficile	Difficile

Figure 3.5 : Tableau récapitulatif des différents modèles de recherche de composants

III.2.3 L'indexation des composants

L'indexation représente la première étape de recherche, qui consiste à établir les techniques permettant de passer d'une description textuelle à une représentation exploitable par un modèle de RI. L'objectif de l'indexation est de trouver les concepts les plus importants dans la description. La liste de ces concepts forme ce que l'on appelle le descripteur du composant. Les techniques d'indexation peuvent être classées selon deux grandes catégories [Sin, al 07]:

- Celles basées sur un vocabulaire contrôlé : les termes utilisés pour décrire un élément sont sélectionnés à partir d'un ensemble prédéfini, tels que les facettes, les mots clés, et l'énumération ;
- Celles basées sur un vocabulaire non contrôlé : aucune restriction sur le vocabulaire pour décrire le composant, tel que le texte libre.

Cette indexation peut être manuelle, réalisée par un expert humain, ou automatique. Le choix et l'intérêt de l'une par rapport à l'autre dépendent d'un certain nombre de paramètres, dont le plus déterminant est le volume des collections. Il est en effet difficile d'indexer manuellement des collections comportant des milliers de composants. Une étude comparative de ces deux approches a été réalisée par Anderson et Carballo dans [And, Car 01]. Le résultat de l'étude montre que les avantages et les inconvénients de chacune des deux approches ont une tendance à s'équilibrer. Autrement dit, le choix de l'une ou de l'autre est en fonction du domaine, de la collection et de l'application considéré.

III.2.3.1 Pondération des termes d'indexation

Le résultat du processus d'indexation est une liste de termes. Ces termes peuvent être de formes différentes : des mots simples ou des groupes de mots. Généralement, un mot est assorti d'un poids représentant le degré d'importance du terme dans le document (la pondération du terme), on distingue [Teb 04]:

III.2.3.1.1 Pondération locale

Elle permet de mesurer la représentativité locale d'un terme, elle indique l'importance du terme dans ce document. Les fonctions de pondération locales les plus utilisées sont les suivantes :

Fonction brut de tf_{ij} *:* (term frequency) : correspond au nombre d'occurrences du terme t_i dans le document D_j.

Fonction binaire : elle vaut 1 si la fréquence d'occurrence du terme dans le document est supérieure ou égale à 1, et 0 sinon.

Fonction logarithmique : combine tf_{ij} avec un logarithme et est donnée par :

$$\alpha + log\ (tf_{ij}) \quad [1]$$

Où **α** est une constante

Cette fonction permet d'atténuer les effets de larges différences entre les fréquences d'occurrence des termes dans le document.

Fonction normalisée : permet de réduire les différences entre les valeurs associées aux termes du document, et est donnée comme suit :

$$0.5 + \left(0.5 \times \frac{tf_{ij}}{max_{\ t_i \in D_j}(tf_{ij})}\right) \quad [2]$$

Où $max_{t_i \in D_j}(tf_{ij})$ est la plus grande valeur de tf_{ij} des termes du document D_j .

III.2.3.1.2 Pondération globale

La pondération globale prend en compte des informations concernant un terme par rapport à la collection de documents. Elle indique la représentativité globale du terme dans l'ensemble des documents de la collection. Un poids plus important doit être donné aux termes qui apparaissent moins fréquemment dans la collection : les termes qui sont utilisés dans de nombreux documents sont moins utiles pour la discrimination que ceux qui apparaissent dans peu de documents. Le facteur de pondération globale qui dépend de la fréquence inverse dans le document a été introduit, sous le nom d' IDF (Inverted Document Frequency) donné par plusieurs formules :

$$IDF\ =\ log\ (N/n_i) \quad [3]$$

Ou

$$IDF\ =\ log\ (N - n_i/N) \quad [4]$$

Où, n_i est le nombre de documents où le terme t_i apparaît dans une collection de documents de taille N.

De ce fait, par cette double pondération locale et globale, les fonctions de pondérations sont souvent référencées sous le nom de $TF - IDF$.

III.2.3.2 Pondération des concepts

La pondération des concepts se base sur l'hypothèse qui dit « les termes composés de plusieurs mots sont plus riches sémantiquement que les mots qui les composent ». Elle permet d'introduire de la sémantique dans la recherche, la méthode qui permet d'effectuer cette pondération est le CF *(Concept Frequency)* qui relie la fréquence d'un terme T composé de n mots par le nombre d'apparition du terme lui-même et de celui de tous ses sous termes dérivés. La formule mathématique est donnée par [Baz 06] :

$$CF(T) = Count(T) + \sum_{ST \in sub\_terme\ (T)} \left(\frac{Lengt\,h(ST)}{Lengt\,h(T)} \times Count(ST) \right) \quad [5]$$

Où :

· $Count(T)$ *:* renvoie la fréquence du terme T.

· $Count(ST)$*:* renvoie la fréquence du sous-terme ST.

· $Length(T)$ *:* renvoie la longueur de T.

· $Length(ST)$ *:* renvoie la longueur de ST.

III.2.4 Les modèles de recherche d'information

Une fois la description des composants est établie, et la requête d'ingénieur (qui représente l'utilisateur du système de recherche) est formulée, une correspondance entre les deux doit être effectuée pour retrouver le composant qui répond le mieux à la requête de l'utilisateur (à partir de sa description). Pour ce faire, l'appel aux systèmes de recherche d'information (SRI) s'avère indispensable. Un modèle de RI a pour rôle de fournir une formalisation du processus de recherche d'information. Il doit accomplir plusieurs rôles dont le plus important est de fournir un cadre théorique pour la modélisation de la mesure de pertinence. La figure 3.5 présente une classification des principaux modèles utilisés en recherche d'information. Les trois principales classes ou modèles de recherche d'information sont [Teb 04] :

i. Les modèles booléens

ii. Les modèles vectoriels

iii. Les modèles probabilistes

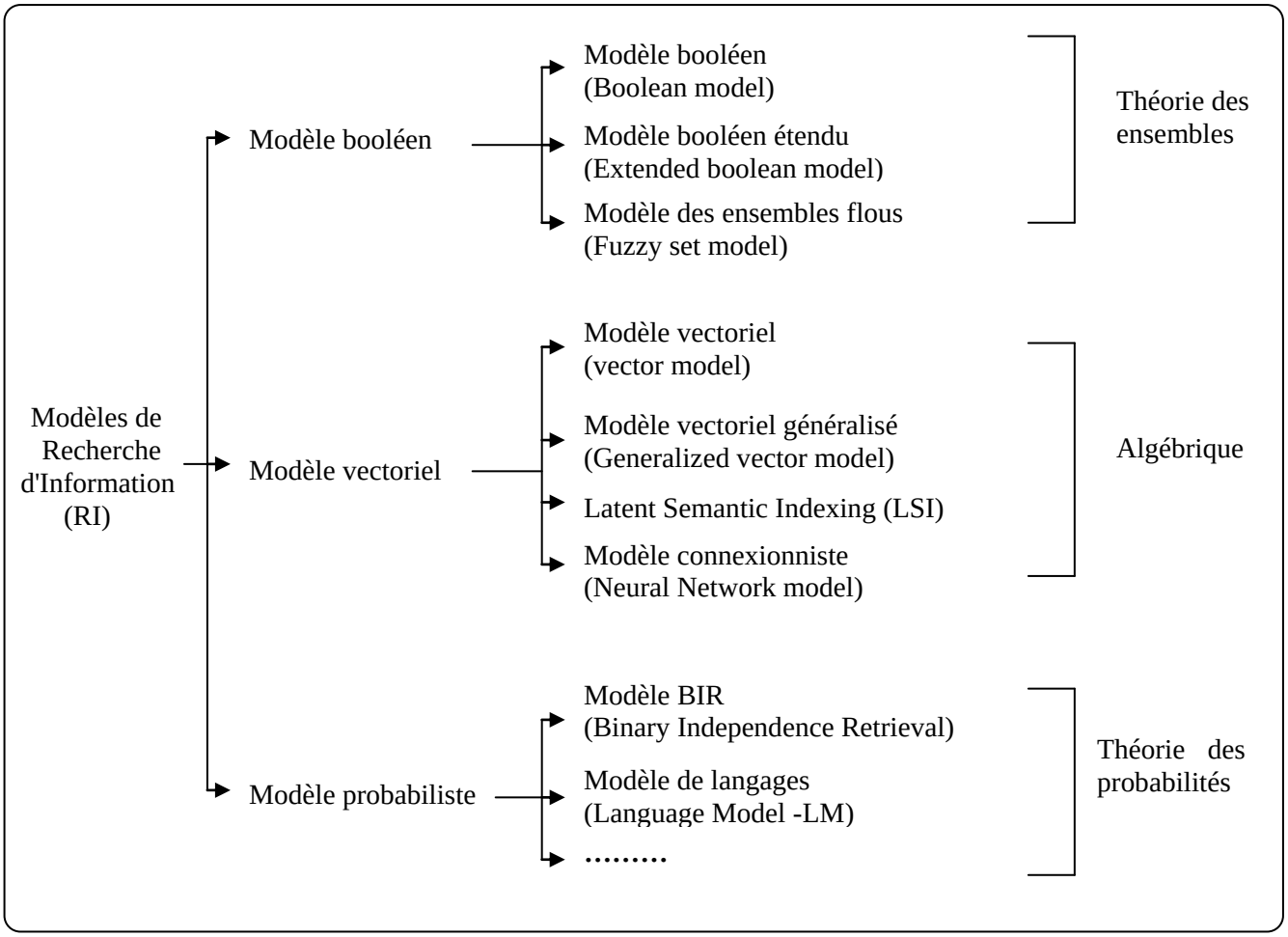

Figure 3.6 : Les modèles de recherche d'information [Teb 04]

III.2.4.1 Les modèles booléens

III.2.4.1.1 Modèle booléen de base

Les modèles booléens se basent sur la théorie des ensembles. En général, la requête est exprimée par une liste de termes et des opérateurs logiques : conjonction (ET), disjonction (OU) et négation (NON). A chaque terme est associé un ensemble de documents où il apparaît. L'opérateur "ET" restreint le résultat de la requête à l'intersection entre deux ensembles, l'opérateur "OU" fournit l'union et l'opérateur "NON" la différence entre les ensembles. Les modèles booléens utilisent le mode d'appariement exact, il ne restitue que les documents répondants exactement aux termes de la requête.

Exemple de requête :

$$Q_k = (tq_{k1} \wedge tq_{k2}) \vee (tq_{k3} \wedge \neg tq_{k4}) \quad [6]$$

Où : Q_k est la $k^{ème}$ requête, et

tq_{ki} est le $i^{ème}$ terme de la requête Q_k.

Soumettre ce type de requête à un SRI basé sur un modèle booléen, implique que ce dernier doit retourner un ensemble de documents contenant simultanément les termes tq_{k1} et tq_{k2} ou un ensemble de documents contenant le terme tq_{k3} et non pas le terme tq_{k4}.

Ce modèle est simple à mettre en œuvre. Tout de même, il présente des inconvénients, par exemple : le système retourne un ensemble de documents non ordonnés comme réponse à une requête. Cependant, il n'est pas possible de dire quel document est plus pertinent qu'un autre. Les termes dans un document sont pondérés de la même façon (soit 0 ou 1), il est ainsi difficile de distinguer les termes les plus importants.

III.2.4.1.2 Modèle booléen étendu

Le modèle booléen étendu, appelé aussi modèle P_Norm, a été introduit en 1983 par Salton. Il représente une alternative du modèle booléen de base, afin de supporter un appariement approché, en assignant des poids aux termes de la requête et des documents et en mesurant un score de pertinence. Le principe de base du modèle booléen étendu est de conférer aux termes de recherche des poids et d'interpréter les opérateurs de l'équation de la requête comme des distances entre requêtes et documents.

III.2.4.1.3 Modèle booléen basé sur des ensembles flous

L'extension du modèle booléen basée sur les ensembles flous a été proposée par Salton en 1989. Cette extension vise à tenir compte de la pondération des termes dans les documents. Un poids d'un terme exprime le degré d'appartenance de ce terme à un ensemble. Ainsi, un document peut être représenté par un ensemble de termes pondérés comme suit :

$$D_j = \{(td_{1j}, a_{1j}), \ldots, (td_{nj}, a_{nj})\}$$

Où :

td_{ij} est le i ème terme du document , et

a_{ij} est le degré d'appartenance (une valeur comprise entre 0 et 1) du i ème terme du document D_j.

III.2.4.2 Les modèles vectoriels

III.2.4.2.1 Modèle vectoriel de base

Les modèles vectoriels reposent sur la théorie algébrique. La pertinence d'un document vis-à-vis d'une requête est définie par des mesures de distance (ou similarité) dans un espace vectoriel.

Dans le modèle vectoriel, les documents et les requêtes sont représentés sous forme de vecteurs dans l'espace vectoriel engendré par les termes du langage d'indexation. Chaque poids d'un terme dans un vecteur document (resp. requête) désigne l'importance de ce terme dans ce document (resp. requête). Ces termes pondérés sont utilisés pour calculer le degré de similarité entre chaque document et la requête de l'utilisateur. Les documents retrouvés sont présentés dans un ordre décroissant de leur degré de similarité correspondant.

Formellement, dans un modèle vectoriel, on suppose que le poids wd_{ij} $(resp. wq_{ik})$ associé au terme t_i dans le document D_j (resp. la requête Q_k) est positif. Les documents et les requêtes sont les vecteurs dans un espace vectoriel de dimension N et sont définis comme suit :

$$D_j = (wd_{1j}, \ldots, wd_{Nj})$$
$$Q_k = (wq_{1k}, \ldots, wq_{Nk})$$

Le modèle vectoriel estime le degré de pertinence entre un document et la requête par un degré de corrélation entre leurs vecteurs associés. Cette corrélation peut être spécifiée par le calcul de similarité entre vecteurs, et qui peut être exprimée par le produit scalaire suivant :

$$Sim(D_j, Q_k) = \sum_{i=1}^{N} (wd_{ij} \times wq_{ik}) \quad [7]$$

Plusieurs fonctions de similarité ont été proposées dans la littérature. Les fonctions les plus répandues : les mesures de Cosinus, Jaccard, Dice…etc.

Le modèle vectoriel permet de pallier l'un des inconvénients majeur du modèle booléen. Il permet effectivement de trier les documents répondant à une requête. Les documents sont en effet restitués dans un ordre décroissant de leur degré de similarité avec la requête. Plus le degré de similarité d'un document est élevé, plus le document ressemble à la requête et donc pertinent pour l'utilisateur.

III.2.4.2.2 Modèle vectoriel généralisé

Théoriquement, le modèle vectoriel de base présente un inconvénient principal lié à l'indépendance mutuelle des termes d'indexation. Wong et son équipe, en 1985, ont proposé un modèle vectoriel généralisé (Generalized Vector Space Model) qui lève l'hypothèse d'indépendance des termes. Dans ce modèle, chaque terme est représenté par un vecteur dans un espace vectoriel et dont les axes sont orthogonaux par construction : les axes sont en fait les produits logiques des termes d'indexation. Un document est représenté par la moyenne des vecteurs représentant les termes qu'il contient. Le but de ce modèle est de transformer une représentation par des mots-clés en une autre représentation. Les documents et les requêtes, sémantiquement similaires, seront plus proches avec la représentation transformée qu'avec les mots-clés. Néanmoins la mise en œuvre de ces modèles est plus lente que celle du modèle vectoriel.

III.2.4.2.3 Modèle connexionniste

Ce type de modèle se base sur les fondements des réseaux de neurones biologiques, tant pour la modélisation des documents et des informations descriptives associées que pour la mise en œuvre du processus de recherche d'information. Ce modèle peut être vu comme un modèle vectoriel récurrent non linéaire, les neurones formels représentent des objets de la recherche d'information. Un réseau de neurone formel est construit à partir des représentations initiales des documents et de la requête. Le mécanisme de recherche d'information est fondé sur le principe de propagation de valeurs depuis les neurones descriptifs de la requête vers ceux des documents, à travers les connexions du réseau. Les résultats sont présentés à l'utilisateur selon le niveau d'activation des neurones documents. Les modèles RI basés sur les réseaux de neurones sont une solution pour combler les lacunes des modèles vectoriels. La notion de réseau est convenable pour représenter les relations et les associations qui existent entre les termes, entre les documents et enfin entre les termes et les documents.

III.2.4.2.4 Latent Semantic Indexing (LSI)

Les documents sont représentés dans un espace de dimension réduite issu de l'espace initial des termes d'indexation. Le but de ce modèle est d'aboutir à une représentation conceptuelle des documents où les effets dus à la variation d'usage des termes dans la collection sont nettement atténués. Ainsi, les documents qui partagent des termes co-

occurrents ont des représentations proches dans l'espace défini par le modèle. Par conséquent, le système permet de sélectionner des documents même s'ils ne contiennent aucun terme de la requête. Ce modèle se base essentiellement sur la décomposition en valeurs singulières, désignée par SVD (Singular Value Decomposition), de la matrice représentant en ligne les termes et en colonne les documents. Un élément de la matrice représente le poids d'un terme dans un document.

III.2.4.3 Les modèles probabilistes

Les modèles probabilistes se basent sur la théorie des probabilités, et essaie de modéliser la notion de *pertinence*. Le principe de base du modèle probabiliste consiste à présenter les résultats de recherche d'un SRI dans un ordre basé sur la probabilité de pertinence d'un document vis-à-vis d'une requête. Ce critère d'ordre est résumé par le "principe de classement probabiliste", désigné aussi par PRP (Probability Ranking Principle). Les modèles probabilistes les plus répandus, sont le modèle BIR et le modèle du langage.

III.2.4.3.1 Modèle BIR

Le modèle BIR (Binary Independence Retrieval) cherche à estimer la probabilité qu'un document D_j soit pertinent pour une requête Q_k , notée $P(R|Q_k, D_j)$. Une hypothèse sur la distribution des termes dans les documents est considérée. L'hypothèse consiste à considérer que la distribution des termes dans les documents pertinents et non pertinents est différente. Cette hypothèse est appelée aussi "clustering hypothesis". L'idée de base du modèle BIR est de représenter les termes des documents par des valeurs binaires (0 ou 1), qui précisent si un terme apparaît dans un document ou non.

III.2.4.3.2 Modèle du langage

L'objectif d'un modèle de langue est de capter les régularités linguistiques d'une langue en observant la distribution et la succession de mots dans la langue donnée. Le modèle de langue désigne une fonction de probabilité qui assigne à chaque séquence de mots une probabilité. L'idée de base des modèles de langues est de déterminer la probabilité (notée $P(Q_j|D_k)$) que la requête puisse être générée par le modèle de langue du document.

Nous terminons par la présentation d'un tableau regroupant les avantages et les inconvénients des ces différents modèles.

	Avantages	Inconvénients
Modèle booléen	-Simplicité de la conception du modèle ; -Possibilité de structurer une requête avec des opérateurs logiques.	-Nombre d'occurrence des mots n'est pas pris en compte ; -La sélection d'un document est basée sur une décision binaire ; -Pas d'ordre pour les documents sélectionnés ; -Formulation de la requête difficile, pas toujours évidente pour beaucoup d'utilisateurs ; - Le nombre de documents retournés peut être considérable.
Modèle vectoriel	-La pondération améliore les résultats de recherche ; -La mesure de similarité permet d'ordonner les documents selon leur pertinence vis à vis de la requête.	-Fondements purement empiriques : peu de fondements théoriques ; -La représentation vectorielle suppose l'indépendance entre termes.
Modèle probabiliste	-Fondements théoriques plus sophistiqués ; -Utilisation facile des modèles.	-Tendance à pousser l'utilisateur à évaluer des probabilités inconnues ou à construire des modèles.

Figure 3.7 : Tableau récapitulatif des différents modèles de recherche d'information

III.3 La sélection des composants logiciels

Un mécanisme de sélection permet de sélectionner, parmi une vaste bibliothèque de composants, le candidat qui répond le mieux à un besoin spécifique, la sélection se repose sur le degré de satisfaction des critères de recherche. Dans la plupart des situations, ces critères de recherche ne possèdent pas la même importance. Une prise de décision pour la sélection des composants logiciels qui prend en considération tous les critères de recherche ainsi que leurs importances respectives s'avèrent nécessaire.

Autre fois ceci s'effectuait d'une manière ad hoc. À présent, des processus de sélection de composants ont été proposés pour améliorer l'efficacité des méthodes informelles [Lio 05].

III.3.1 Les étapes de sélection

Şen et Baraçli dans [Şen, Bar 06] ont recensé un ensemble de 34 travaux portant sur les processus de sélection de composants logiciels, tels que : OTSO (Off-The-Shelf-Option), CISD (*COTS*-based integrated system development), PORE (Procurement-oriented requirements engineering), STACE (social-technical approach to *COTS* evaluation), CAP (*COTS* acquisition process), CRE (*COTS*-based requirements engineering)...etc. Après une étude détaillée de ces méthodes, les auteurs ont remarqué que:

- Toutes les approches ont une idée commune qui est l'utilisation des techniques de prise de décision ;
- Les approches se distinguent par les étapes proposées pour le processus de sélection, mais ils ont noté que tout processus proposé comporte au moins les trois phases suivantes :

  -La description du besoin sous la forme de critères d'évaluation ;

  -La hiérarchisation de ces critères ;

  -L'évaluation des candidats sur la base de ces mêmes critères.
- Certains de ces travaux proposent des méthodes spécifiques pour déterminer les critères d'évaluation, tels que l'analyse des besoins détaillés, des entretiens, de questionnaires et de brainstorming (le remue-méninge) et tous les autres travaux ont utilisé des critères d'évaluation prédéfinis. L'utilisation inappropriée des critères dans le processus de sélection a une influence directe sur la décision définitive de sélection ;

- Pour leur réalisation, ces processus préconisent en général l'usage de techniques dites de prise de décision multicritères. Les plus utilisées dans ces travaux sont WSM (*Weighted Scoring Method)* et AHP (*Analytic Hierarchy Process).*

III.3.2 Les techniques de sélection

Les techniques de sélection des composants logiciels proposées par les auteurs sont généralement des techniques de décision multicritères.

III.3.2.1 La technique WSM

La méthode WSM (*Weighted Scoring Method)* est une technique de prise de décision multicritères et s'applique de la manière suivante [Kon 96] :

Définir les critères, et pour chaque critère attribuer un poids ou un score.

Dans le cas de l'utilisation des poids, ces derniers doivent être normalisés pour avoir un total de un.

Si le score est utilisé, Le score de chaque alternative est calculé par la formule suivante :

$$Score_c = \sum_{j=1}^{N} (Weight_j \times Score_{cj}) \quad [8]$$

-Le poids $Weight_j$ représente l'importance du $j^{ème}$ critère par rapport aux $n-1$ autres critères d'évaluation.

-Le score local $Score_{cj}$ évalue le degré de satisfaction du critère numéro j par le candidat c.

-Le score total $Score_c$ représente la valeur globale d'évaluation de c.

Le meilleur candidat est celui dont le score total est le plus élevé.

L'application de WSM est simple et intuitive, mais présente des limitations qui sont souvent ignorées quand elle est appliquée. Par exemple WSM produit des nombres réels comme résultats, qui peuvent facilement être interprétés comme s'ils représentaient vraiment les différences exactes entre les alternatives, ce qui n'est toujours pas le cas. Et l'attribution de poids devient difficile lorsque le nombre de critères augmente.

III.3.2.2 La technique AHP

AHP (*Analytic Hierarchy Process)* a été développée par Thomas Saaty en 1999, et a été utilisée avec succès dans divers domaines. C'est une technique de prise de décision multicritères qui été adaptée pour hiérarchiser les exigences logicielles. Elle permet de décrire le besoin d'une manière organisée, en le décomposant en un arbre hiérarchique de critères et de sous-critères d'évaluation. A l'intérieur de chaque critère-nœud, on détermine l'importance de chaque sous-critère par rapport aux autres. Cette technique suppose que les critères sont indépendants [Wan 06], [Kon 96].

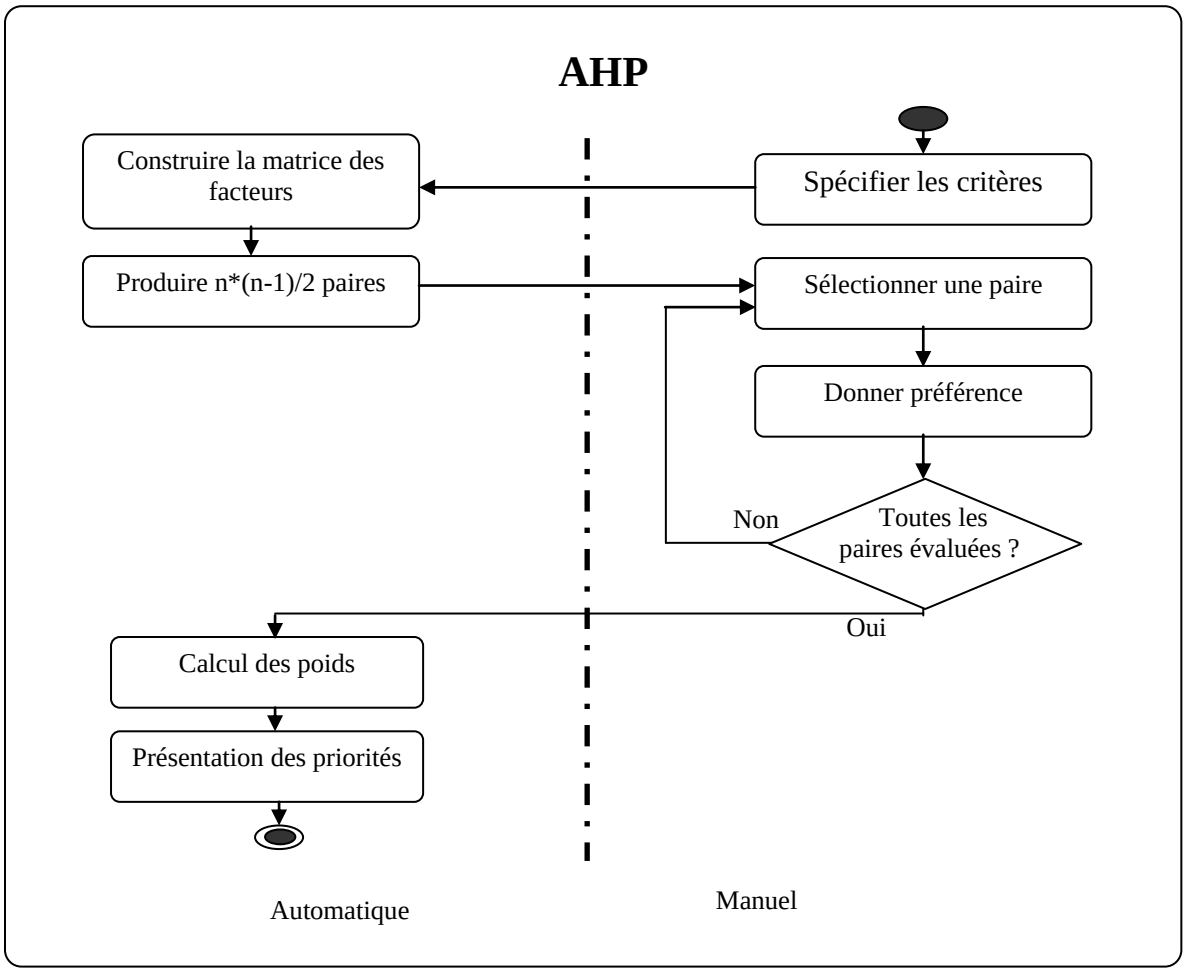

Figure 3.8 : Les étapes de la méthode AHP [Per, al 09]

Les principales étapes dans l'application de AHP sont représentées dans l'organigramme de la figure 3.8 et procède comme suit [Per, al 09]:

-Définir des critères qui influencent la décision.

-Définir une matrice dont les lignes et les colonnes représentent ces critères. Chaque élément de cette matrice est défini par une paire de ces critères. La valeur attribuée à cet élément représente la préférence des utilisateurs sur la paire correspondante.

-L'utilisateur effectue une série d'activités pour achever l'évaluation de toutes les paires de l'ensemble des critères (ce qui correspond à la boucle dans la figure).

-La valeur attribuée à chaque paire est un nombre entier appartenant à l'échelle [1,9], qui représente une mesure qualitative de la relation de préférence entre les exigences (par exemple si le critère A est "tout aussi important "que le critère B, la valeur 1 est donnée. Si le critère A est "principalement plus important "que le critère B, la valeur 9 est donnée).

-Lorsque toutes les paires ont été évaluées, le calcul s'effectue par l'algorithme AHP qui repose sur le calcul des valeurs propres de la matrice des exigences.

Le résultat est représenté par un vecteur de poids qui spécifie la priorité de chaque critère.

Cette technique donne une bonne précision, mais elle est consommatrice en termes de temps et demeure toute fois difficile à utiliser.

III.4 Conclusion

Nous avons vu dans ce chapitre, les différents aspects liés à la recherche et la sélection de composants logiciels. Nous avons cités comme techniques de recherche celles basées sur la description externe des composants, celles basées sur les approches formelles (l'appariement de signatures et de spécifications) et celles basées sur l'analyse du comportement des composants. Nous avons également présenté les aspects liés à la recherche d'information et les divers modèles (les modèles booléens, les modèles vectoriels et les modèles probabilistes).

L'ensemble des points cités dans ce chapitre, nous donne l'opportunité pour faire un bon choix de techniques à adopter, pour la mise en œuvre de notre système de recherche.

La section suivante mettra en exergue notre problématique de recherche, elle sera suivie par notre contribution.

Deuxième partie

Contribution

Chapitre IV

Contribution et validation

IV.1 Introduction

Après avoir établi un état de l'art sur le contexte de notre travail, nous étalerons notre problématique ainsi que l'idée générale de notre approche de recherche, puis nous passerons aux détails de chaque point.

IV.2 Contexte et problématique

Le développement à base de la réutilisation de composants logiciels vise à réduire le coût, le délai et la complexité du développement [Li, al 09]. La réutilisation de composants logiciels distingue deux types de problématiques [Gac, Ser 03], [Puj, al 05]. Celle qui relève de l'ingénierie pour la réutilisation « Design for reuse » qui consiste en la production de composants réutilisables, et celle qui relève de l'ingénierie par réutilisation « Design by reuse » qui consiste à exploiter les composants réutilisables pour le développement du logiciel. Ces deux démarches de développement sont complémentaires et les bibliothèques de composants sont leur point central [Kha 05]. Pour développer un produit logiciel, le développeur recherche dans la bibliothèque les composants correspondants à ses besoins. Une fois les composants acquis, il doit les adapter puis les assembler afin de construire le système final [Mos 01].

Un problème essentiel dans ce contexte est comment retrouver les composants adaptés aux spécificités du système en cours de développement et aux besoins des ingénieurs. Les difficultés sont liées le plus souvent à l'absence d'une description concise des composants. Parce qu'une description incomplète ou inconsistante rend difficile sa recherche, puisqu'elle peut inclure dans les résultats les composants qui ne répondent pas aux exigences et en exclure les pertinents. Ainsi le développeur sera mené à développer les composants de nouveaux, alors qu'ils sont présents dans la bibliothèque mais mal référencés.

Pour mener à bien le processus de réutilisation, avant de stocker les composants dans la bibliothèque, ces derniers doivent être décris d'une manière cohérente et sans équivoque. La manière de représenter et d'organiser les composants devient ainsi indispensable pour faciliter la réutilisation de composants logiciels [Gae, Tur 00], [Li, al 09].

La recherche dans la littérature sur la manière dont ces composants peuvent être décris, nous a permit à distinguer plusieurs formalismes, allant d'une description en langage naturel à la classification à base de facettes, jusqu'à la représentation formelle [Alm, al 07].

Après l'étude de ces formalismes, notre choix s'est porté sur la combinaison de la représentation à base de facettes avec la description en langage naturel. Nous pensons que c'est la meilleure approche, si nous parvenons à rassembler les points forts des deux formalismes à la fois. Ainsi en faisant un bon choix de facettes et en utilisant les techniques de recherche d'information basées sur la sémantique, notre approche sera plus intéressante. Par voie de conséquence, l'aspect initial de la classification à base de facette sera modifié pour l'adapter à nos besoins. Il sera scindé en deux parties : une partie contenant les facettes à vocabulaire contrôlé et une autre partie contenant les facettes à vocabulaire non contrôlé, i.e, une description en langage naturel. Nous pensons qu'avec cette fusion, la description du composant sera entière. Ces facettes doivent permettre de décrire la majorité des aspects du composant. Elles vont servir à la fois comme facettes pour la représentation du composant dans la bibliothèque et comme critères à base desquels l'utilisateur effectuera sa recherche. Pour bien déterminer nos facettes, nous avons effectué une étude sur une douzaine de travaux traitant ce problème.

Afin d'effectuer la recherche proprement dite et récupérer les composants qui sont pertinents à une requête donnée, la mesure de similarité basée sur la combinaison du modèle booléen et du modèle vectoriel sera utilisée. Ce sont les deux modèles qui s'adaptent le plus à la description que nous avons proposé pour les composants, le modèle booléen sera employé pour les facettes dont le vocabulaire est contrôlé, et le modèle vectoriel pour les facettes du vocabulaire non contrôlé.

Pour présenter les composants pour les utilisateurs, par ordre de pertinence, nous avons fait appel à la fois aux techniques de prise de décision multi critères WSM et AHP, qui vont nous aider à bien gérer la diversité des critères, en attribuant à chacun d'eux un poids symbolisant son importance dans la recherche.

Afin de tester notre système, nous avons rassemblé une collection de composants, pris de divers sites de composants (Les marchés des composants, ou market places) tels que : Component Source [Com 09], OSGI [Osg 09], Source Forge [Sou 09], VLC Component [Vcl 09]... etc, puis transformé selon notre modèle de description. L'évaluation du système se fera par un calcul du taux de précision, de rappel et de temps d'exécution pour un ensemble de requêtes donné. Nous avons implémenté deux approches de description, celle basée sur les facettes et celle basée sur le langage naturel. Le but est de pouvoir comparer leurs résultats avec ceux obtenus par notre système qui représente la fusion des deux approches. Le même

ensemble de requêtes de test a été soumis aux deux approches sur la même collection de composants.

À présent, l'approche générale de notre processus de recherche est définie, les détails de chaque point seront présentés dans le point suivant.

IV.3 L'architecture du système de recherche

La figure ci-dessous illustre l'architecture générale du processus adopté pour effectuer la recherche.

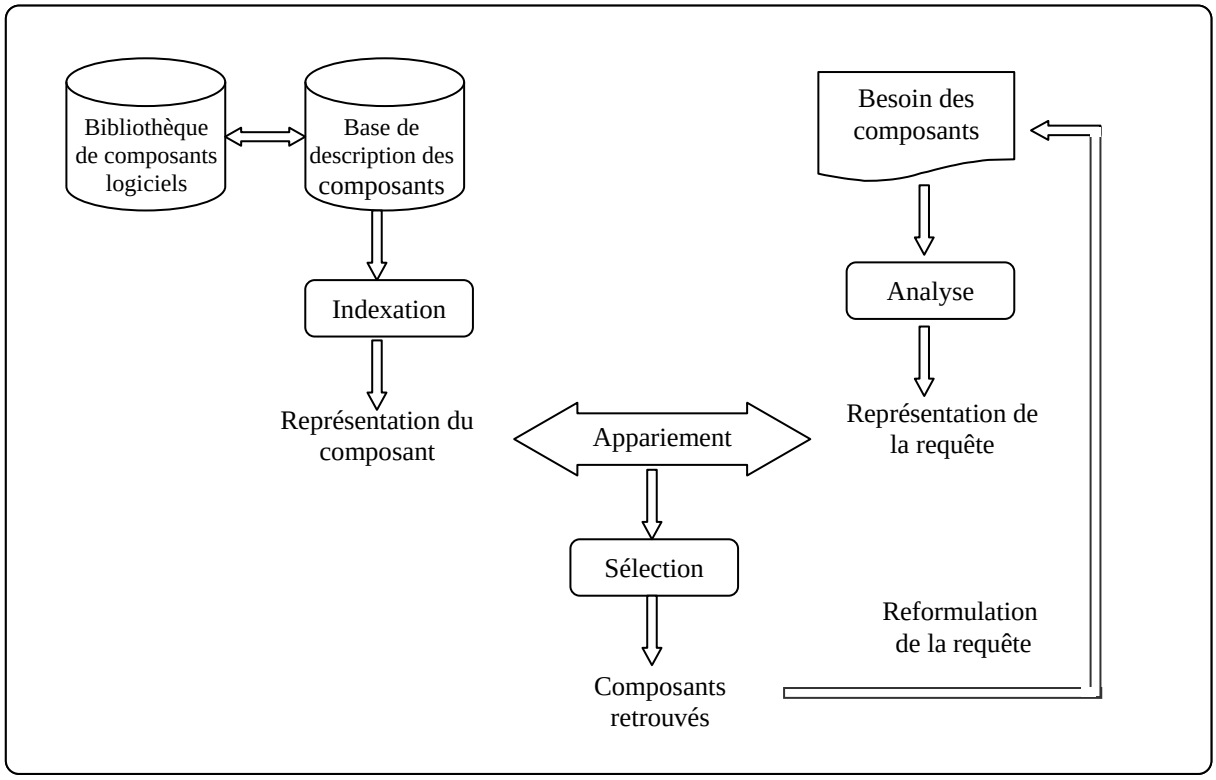

Figure 5.1 : L'architecture du système de recherche

À chaque composant logiciel est associée une description externe, ainsi, on ne manipule plus les composants mais plutôt leurs descriptions. Pour faciliter le processus de recherche, une indexation de l'ensemble des descriptions des composants est nécessaire. C'est sur cette nouvelle représentation qu'on effectue la recherche et l'appariement par rapport à la requête formulée par l'utilisateur qui consiste à attribuer des mots clés pour un ensemble de facettes du composant. Une fois fait, le mécanisme de sélection est appelé, il consiste en l'application des poids à chaque facette, ainsi qu'un calcul du Cf (Concept frequency) pour certaines facettes. La présentation du résultat de la recherche à l'utilisateur se fera par ordre

décroissant du score calculé pour chaque composant. Un mécanisme de reformulation de requête est appelé, il consiste à réafficher la requête de l'utilisateur si celui-ci veut changer, supprimer ou ajouter de nouveaux mots clés.

IV.4 La description des composants

La description des composants représente un aspect important dans la conception des systèmes de recherche. Elle influe sur la forme des requêtes auxquelles le système peut répondre et sur la façon de les évaluer. Elle influe également sur la structure de stockage et la technique de recherche à utiliser. C'est d'ailleurs pour cette raison que nous accordons une attention particulière sur la manière de caractériser les composants logiciels.

IV.4.1 Choix de la description à utiliser

Notre avons choisi d'établir une description des composants basée sur l'association de la description à base de facettes et de la description en langage naturel. Ainsi l'aspect global reste le même, basé sur les facettes, toutefois ces facettes seront scindées en deux parties, celles basées sur un vocabulaire contrôlé et celles basées sur un vocabulaire non contrôlé ie, en langage naturel. La nouvelle description générée en adoptant le langage UML, sera comme suit :

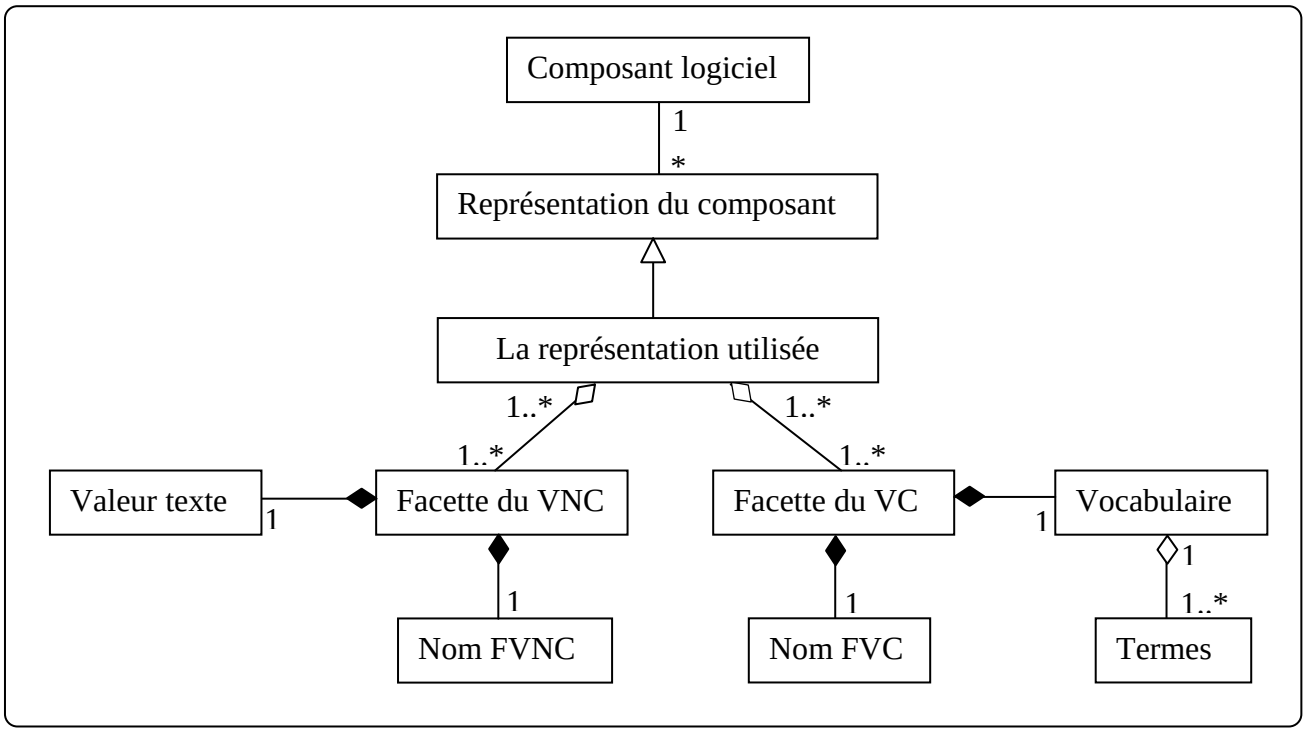

Figure 5.2 : La description des composants utilisée

Où : VC : Vocabulaire Contrôlé
VNC : Vocabulaire Non Contrôlé
FVC : Facettes Vocabulaire Contrôlé
FVNC : Facettes Vocabulaire Non Contrôlé

IV.4.2 L'étude réalisée

Afin de définir les critères les plus pertinents pour la recherche des composants logiciels, plusieurs travaux ont été rapportés dans la littérature sur ce point. Les auteurs tentent de proposer l'ensemble qui leur semble plus pertinent. Tous les ensembles diffèrent les uns des autres, et à l'heure actuelle, il n'y a pas eu d'adhésion sur les critères à utiliser pour aboutir à une meilleure recherche.

Nous avons rassemblé une douzaine de travaux qui s'inscrivent dans ce domaine. Après les avoir étudiés, nous avons essayé de faire ressortir l'ensemble des attributs les plus fréquemment cités. Nous pensons qu'avec cette démarche, nous allons pouvoir utiliser les meilleurs critères pour aboutir à une recherche plus pertinente.

Plusieurs contraintes ont été rencontrées sur ce point, nous pouvons citer:

- Les auteurs utilisent des attributs portant des noms similaires mais ne possédant pas la même signification. Dans ce cas, on prend en considération le sens, et on les comptabilise dans les attributs qui leurs sont associés ;
- Il existe également des cas où les auteurs utilisent des noms d'attributs différents, mais leur signification reste la même. Dans ce cas, on garde l'attribut le plus significatif ;
- Il arrive que certains auteurs utilisent un attribut qui est exprimé par d'autres auteurs au moyen de deux attributs distincts ou plus. Dans ce cas, on garde l'attribut (ou l'ensemble d'attributs) le (s) plus fréquemment cité (s) ;
- La classification des critères est aussi différente que l'ensemble de critères choisis par chaque auteur, après avoir déterminé les attributs à inclure, on essaye de les organiser dans des rubriques dont le nom est indicateur du contenu.

Les travaux recensés sont les suivants :

IV.4.2.1 Le travail de J. Sametinger, dans [Sam 97]. L'auteur propose une taxonomie pour les composants réutilisables, il fournit un cadre (Framework) pour la création et la recherche de composants logiciels. Les catégories des composants permettent de déterminer le potentiel de réutilisation de composants spécifiques et d'évaluer l'état actuel de

la réutilisation des composants. Cette taxonomie est basée sur les interfaces du composant, les techniques de composition et les attributs de plateforme, sans détails sur le domaine.

IV.4.2.2 Le travail de J. Dong, P. S. C. Allencar, D. D. Cowan, dans [Don, al 99]. Les auteurs proposent un descripteur pour la spécification des composants logiciels commercialisés. Celui-ci peut être utilisé comme une base pour guider les développeurs à base de composants et les fournisseurs dans la mise à niveau de leurs composants ainsi que pour les intégrateurs à sélectionner et à assembler les composants. Les auteurs jugent que la spécification des propriétés fonctionnelles et non fonctionnelles permet aux intégrateurs de vérifier la cohérence des systèmes composés, mais n'inclut pas les informations concernant le fabriquant du composant, son coût, son domaine…etc.

IV.4.2.3 Le travail de S. Yacoub, H. Ammar, A. Mili, dans [Yac, al 99]. Les auteurs proposent la classification des différentes caractéristiques de composants selon trois catégories, à savoir: description informelle**,** propriétés externes et propriétés internes, dérivant pour chaque catégorie un ensemble de caractéristiques pour le composant logiciel. Ces caractéristiques permettent de comprendre ce qu'est un composant logiciel et comment il peut être classé. Selon les auteurs, cette caractérisation permet un meilleur catalogage, une meilleure utilisation et une meilleure recherche dans les bibliothèques de composants, mais n'inclut pas les informations concernant le fabriquant du composant et le coût.

IV.4.2.4 Le travail de Morisio et Torchiano, dans [Mor, Tor 02]. Les auteurs étendent le travail de David Carney et Fred Long [Car, Lon 00] et proposent un framework pour la caractérisation de COTS. Ce framework contient un certain nombre d'attributs et des valeurs possibles pour chaque attribut. Un composant est décrit par une valeur unique pour chaque attribut. Des composants différents ayant le même ensemble de valeurs appartiennent à la même classe. Le travail est basé sur les attributs techniques, constitués de dix attributs regroupés dans quatre groupes soient : l'origine du composant, sa personnalisation, son paquetage compilé et son rôle. Cette classification ne prend pas en considération l'aspect non fonctionnel des composants.

IV.4.2.5 Le travail de S. Ben Sassi, L. Labed Jilani, H. Hajjami Ben Ghezala dans [Sas, al 03]. Les auteurs pensent qu'aucun des travaux retrouvés dans la littérature n'a pu pleinement préciser les caractéristiques des composants logiciels. Afin d'en savoir

suffisamment sur les composants le long de leur processus d'utilisation, les auteurs proposent un modèle déduit de l'homogénéisation des attributs définis dans d'autres travaux, et le complètent avec d'autres. Ce modèle élargi est structuré en sept catégories en fonction de leur sémantique d'utilisation. Chaque catégorie d'attributs peut être utilisée dans une ou plusieurs phases du processus de développement.

IV.4.2.6 Le travail de I. Rebaï, J. M Labat, **N. Maisonneuve** dans [Reb, al 05] [Reb 06] [Reb, al 04]. Les auteurs s'intéressent particulièrement à l'indexation, le stockage et la recherche des composants logiciels pédagogiques spécifiques au domaine des EIAH (Environnements Informatiques d'Apprentissage Humain). Les auteurs proposent des descripteurs des métadonnées pour les composants pédagogiques. Ces descripteurs sont partagés en deux sections, une section spécifique aux composants logiciels pédagogiques et une autre, à laquelle nous nous intéressons, appelée section commune. Cette section permet de décrire n'importe quelle catégorie de composants logiciels, et ne contient que des caractéristiques du composant, indépendamment de son domaine et de sa spécialité.

IV.4.2.7 Le travail ASSET selon [Kha 05], ASSET (Asset Source of Software Engineering Technology) est une base de composants conçue en 1993 par l'agence ARPA (Advanced Research Projects Agency) sous le programme STARS. Un composant ASSET est défini comme un composant logiciel réutilisable. Le composant est décrit par un ensemble d'informations telles que : nom, autres noms alternatifs, date de création, taille, type de composant, domaine, …etc. La base de composants ASSET permet d'effectuer une recherche à base de facettes.

IV.4.2.8 Le travail de G. Redolfi, L. Spagnoli, P. M. Cristal, A. Espindola, dans [Red, al 05]**.** Les auteurs proposent un modèle de référence pour la description des composants logiciels, qui représente les informations nécessaires sur les composants afin de faciliter leur réutilisation. L'objectif principal de ce modèle de référence est de définir un référentiel capable de stocker, de cataloguer, de rechercher et de sélectionner les composants. Les auteurs proposent de classer les informations relatives au composant, selon les sept groupes suivants: identification, utilisation, maturité, documentation, technologie, modification et assurance qualité.

IV.4.2.9 Le travail de M. Keil, et A. Tiwana, dans [Kei, Tiw 05]. Les auteurs ont effectué une étude étendue, portant sur 126 organismes, pour analyser comment les gestionnaires des systèmes d'information évaluent t'ils les principaux attributs des composants logiciels. Les résultats donnent un aperçu sur les caractéristiques des composants logiciels, que les acheteurs jugent plus importants. Les résultats de leurs recherches recensent une longue liste des critères possibles d'évaluation pour les composants logiciels. Toutefois, les auteurs ont identifié uniquement sept caractéristiques, que tous les responsables mentionnent comme étant les plus importantes, et qui représentent un ensemble de critères d'évaluation de composants.

IV.4.2.10 Le travail de N. S. Gill, dans [Gil 06]. L'auteur reprend le travail présenté dans [Yac, al 99], sans apporter le moindre changement apparent dans son modèle. Des attributs influençant le choix des utilisateurs ne sont pas cités dans ce descripteur. Par exemple, le constructeur du composant, le coût…etc. L'attribut *age* est intéressant, mais pour éviter le changement fréquent de la valeur de l'attribut, mieux vaut préciser la date de production du composant.

IV.4.2.11 Le travail de H. M. Kienle et H. A. Muller, dans [Kie, Mul 07]. Les auteurs introduisent une taxonomie pour caractériser les composants. Celle-ci dessine une image globale du composant et dont les valeurs sont souvent tirées de la documentation des développeurs. La taxonomie permet d'effectuer une comparaison des composants qui peuvent être utilisés comme entrée pour la sélection des composants, elle peut aussi identifier les caractéristiques qui ont un impact sur le système et ses exigences. La taxonomie a six niveaux de critères, entre autres: origine, la forme de distribution, les mécanismes d'adaptation, le package,…etc.

IV.4.2.12 Le travail de A. Masmoudi, R. Champagne, G. Paquette, dans [Mas, al 08]. Les auteurs présentent leur structure de représentation de composant intitulée SOCOM (SOftware COmponent Metadata), pour décrire les principales caractéristiques d'un composant logiciel. Cette structure étend le travail présenté dans [Mas, al 06], lui apportant un ensemble de changements dans le but de décrire de façon complète plusieurs types de composants. La nouvelle structure couvre six aspects, dont les suivants : statique, services, qualité, plateformes,…etc. L'approche permet le référencement et la recherche des

composants dans le système TELOS (TELelerning Operating System), en cours de construction au sein du projet LORNET (Learning Object Repositories' NETworks).

IV.4.3 La détermination des facettes

Nous pensons que le travail effectué dans la section précédente possède un double intérêt. En plus de la détermination des attributs, il nous permet d'associer une pondération à ce même ensemble, en fonction du nombre d'apparition des attributs dans l'ensemble des travaux traités.

Gorman dans [Gor 99] indique que l'inclusion des attributs supplémentaires permet d'améliorer l'efficacité de la recherche de composant. Poulin, dans [Pou 99] suggère que la collecte de grandes quantités d'information, pour aider à retrouver les composants, entraîne une perte de temps et d'argent et rend difficile la recherche dans la bibliothèque. Il est donc nécessaire de limiter la sélection des attributs à ceux qui ont un impact important par rapport à un contexte particulier [Boo, al 03]. C'est pour cette raison, que nous avons gardé uniquement les attributs qui ont été cités dans au moins trois travaux, et nous avons éliminé ceux qui sont apparus une seule fois ou deux. Les attributs mis en relief après cette étude sont énumérés dans le tableau de la figure 5.3 :

Rubrique	Attributs
Générale	• Nom du composant • Intention • Coût / Licence • Auteur • Date de création • Version • Taille • Encapsulation • Contact • Domaine
Technologie	• Standard • Langages de Développement • Systèmes d'exploitation
Spécificités	• Aspect comportemental • Aspect Structurel • Interfaces Requises • Interfaces Fournies • Types d'adaptation
Composants en rapport	• Composants Similaires • Composants Collaborateurs
Evaluation	• Fiabilité • Performance • Portabilité • Sécurité • Facilité d'utilisation • Interopérabilité

Figure 5.3 : Le modèle de caractérisation proposé

La description détaillée de chaque rubrique est présentée aprés:

IV.4.3.1 La rubrique « Générale »

Regroupe les informations, permettant de décrire le composant d'une manière assez générique, sans entrer dans les détails techniques ou d'implémentation. Cette section convient aux développeurs ne possédant pas beaucoup d'informations sur le composant. Elle renferme les attributs suivants.

1. **Nom du composant:** Permet de déterminer un nom identifiant le composant.
2. **Intention :** Précise le rôle essentiel du composant. Ceci inclut le problème ou l'ensemble des problèmes que le composant résout (ou participe à résoudre).
3. **Coût / Licence** : Présente la redevance pour acquérir le composant ainsi que sa licence, si celui-ci est payant, ou préciser s'il est libre.
4. **Auteur :** Représente l'origine du composant à travers l'identification du développeur, l'équipe, l'entreprise… etc. responsable de son développement.
5. **Date de création:** Cette caractéristique reflète la stabilité et la maturité du composant.
6. **Version :** Représente un identificateur numérique qui permet au mécanisme de stockage de composants d'enregistrer l'historique d'un composant. Les nouvelles versions du composant sont établies sur la base des modifications faites au composant, afin de permettre sa réutilisation ou son évolution / la correction de ses fonctions.
7. **Taille** : Détermine la taille du composant. Qui peut être exprimé en nombre de ligne code du composant ou en Ko.
8. **Encapsulation :** Il définit le niveau de visibilité des détails internes d'un composant réutilisable. En d'autre terme, précise l'accès au code source du composant. Ce qui a un impact direct sur la manière d'utilisation**.**
9. **Contact :** Permet de définir les informations permettant de contacter le fournisseur du composant.
10. **Domaine :** Représente les situations dans lesquelles le composant peut être réutilisé, il permet de définir le domaine et les applications possibles où le composant peut être intégré.

IV.4.3.2 La rubrique «Technologie»

Ce groupe de caractéristiques englobe les informations relatives à l'implémentation du composant et de la technologie utilisée dans sa construction

1. **Standard :** Inclut les informations sur le modèle du développement du composant, tels que CCM, COM ou EJB, OSGi...etc.

2 **Langages de développement :** Permet de préciser le langage avec lequel le composant est développé.

3 **Systèmes d'exploitation :** Précise les systèmes d'exploitation où le composant peut s'exécuter.

IV.4.3.3 La rubrique «Spécificités »

Précise les détails techniques spécifiques au composant, cette rubrique permet de raffiner la recherche du composant d'une manière significative Elle renferme les attributs suivants :

1 **Interfaces fournies:** Enumère toutes les interfaces fournies par le composant et certains de leurs détails

2 **Interfaces requises**: Précise l'ensemble des interfaces dont le composant a besoin pour fonctionner.

3 **Types d'adaptation :** Précise le degré de modification du composant, ie, les types d'adaptation pouvant être appliqués au composant.

4 **Aspect structurel** : La structure du composant décrit les participants internes du composant et leur manière de collaborer, comprend les informations syntaxiques telles que : les noms, les types, le nombre de paramètres et d'autres informations statiques.

5 **Aspect comportemental :** L'aspect comportemental du composant permet de comprendre comment le composant se comporte. Il est important de ne pas avoir des comportements non attendus lors de l'utilisation du composant.

IV.4.3.4 La rubrique «Composants en rapport»

Cette caractéristique définit les composants connus qui résolvent le même problème, ou un problème similaire. L'inclusion de ce rapport d'information entre les composants, offrant des services similaires et provenant de différents fabricants, permet aux intégrateurs de retrouver les composants cibles avec moins d'effort et moins d'erreur. Ces informations sont également utiles lorsque les intégrateurs veulent savoir quels éléments collaborent bien avec un composant spécifique. La rubrique contient les deux attributs suivants :

1 **Composants similaires***:* Composants offrant des services similaires**.**

2 **Composants collaborateurs :** Composants avec lesquels il peut collaborer.

IV.4.3.5 La rubrique «Evaluation »

Dans certains cas, il est essentiel de déterminer l'aspect non fonctionnel du composant, lorsque qu'il est destiné à des domaines critiques.

1. **Fiabilité:** Capacité du composant à remplir ses fonctions dans des conditions d'exploitation définies et sur une période de temps donnée.
2. **Performance:** Représente le temps d'exécution d'une fonction d'un composant.
3. **Portabilité**: La portabilité caractérise les interfaces de plateforme, ie, comment le composant interagit avec la plateforme (hardware, système d'exploitation, sous-système de communication…etc.), ainsi que les questions liées à la possibilité d'utiliser le composant dans d'autres plates-formes.
4. **Sécurité:** Concerne le contrôle d'accès, authentification …etc.
5. **Facilité d'utilisation:** Permet de mesurer la facilité d'utilisation du composant lors de la conception d'applications.
6. **Interopérabilité:** Cette propriété précise l'interaction entre les composants, compte tenu de l'échange d'information entre composants développés dans différentes technologies. Cette information précise et / ou limite l'interaction entre les composants de différentes technologies.

IV.4.4 La détermination du vocabulaire

Nous avons sectionné nos facettes en deux sections, celles basées sur un vocabulaire contrôlé et celles basées sur un vocabulaire non contrôlé. Nous présentons dans ce qui suit, les différentes facettes de chaque section.

IV.4.4.1 Vocabulaire contrôlé

Le vocabulaire contrôlé va contenir l'ensemble des facettes suivantes : Coût / Licence, Taille, Encapsulation, Domaine, Standard, Langages de développement, Systèmes d'exploitation, Interopérabilité, Fiabilité, Performance, Portabilité, Sécurité, Facilité d'utilisation et Types d'adaptation. Nous présentons ci-dessous quelques facettes de cet ensemble avec le vocabulaire qu'il lui est associé :

- **Coût / Licence**: Free, Shareware, Demos
    - **Coût:** Contiendra des intervalles de prix

- **Licence**: AFL, AL20, ASL, APSL, BSD, CPL, EPL10, GTPL, GPL, LGPL…etc.

- **Encapsulation:** Black box, Gray Box et White Box.
- **Domaine:** Base de données, Design, E-commerce, Internet, Mathématiques, Réseaux, Sécurité, Développement logiciel…etc.
- **Standard:** EJB, CCM, COM, DCOM, COM+, .NET, JBEAN, OSGI, …etc.
- **Langage de développement :** ActionScript, Ada, ASP, ASP.NET, Assemble, C, C#, C++, Cobol, ColdFusion, Delphi, Eiffel, Erlang, Fortran, Java, JavaScript, JSP, Lisp, Lua, Mathematica, Matlab, ObjectiveC, Perl, PHP, Prolog, Python, Ruby, Scheme, Smalltalk, SQL, Tcl, VB, VB.NET.
- **Sécurité:** Considérée et Non considérée.

IV.4.4.2 Vocabulaire non contrôlé

Le vocabulaire non contrôlé va contenir l'ensemble des facettes ne pouvant pas être cerné par un vocabulaire prédéterminé, il aura comme facette : Nom du composant, Intention, Auteur, Version, Contact, Aspect comportemental, Aspect structurel, Interfaces requises, Interfaces fournies, Composants similaires et Composants collaborateurs.

IV.5 La recherche des composants logiciels

L'étape qui précède la recherche effective est la formulation de la requête, L'utilisateur trouve souvent des difficultés pour exprimer ces besoins sous forme de requête. Nous avons pensé à développer une interface évoluée qui va lui proposer tous les critères qui peuvent influencer son choix ; son travail consiste à sélectionner une valeur pour chaque critère dans la liste s'il fait partie du vocabulaire contrôlé ou a en introduire s'il fait partie du vocabulaire non contrôlé. L'interface va permettre également à l'utilisateur de préciser une priorité pour les critères. Avec cette interface les besoins de l'utilisateur vont être bien cernés. Nous avons également pensé à développer une deuxième interface de recherche qui viendra d'utilité pour les développeurs n'ayant pas une idée fixe sur le composant. Celle-ci lui permettra de naviguer dans le contenu de la bibliothèque en fonction d'un critère particulier.

Notre représentation du composant est réalisée à l'aide de deux types de facettes, ce qui nous offre la possibilité de les traiter de façons différentes. Cette partie d'indexation est propre aux facettes issues du vocabulaire non contrôlé, présentées en texte libre.

Notre processus d'indexation s'effectue par les différentes étapes illustrées sur la figure suivante.

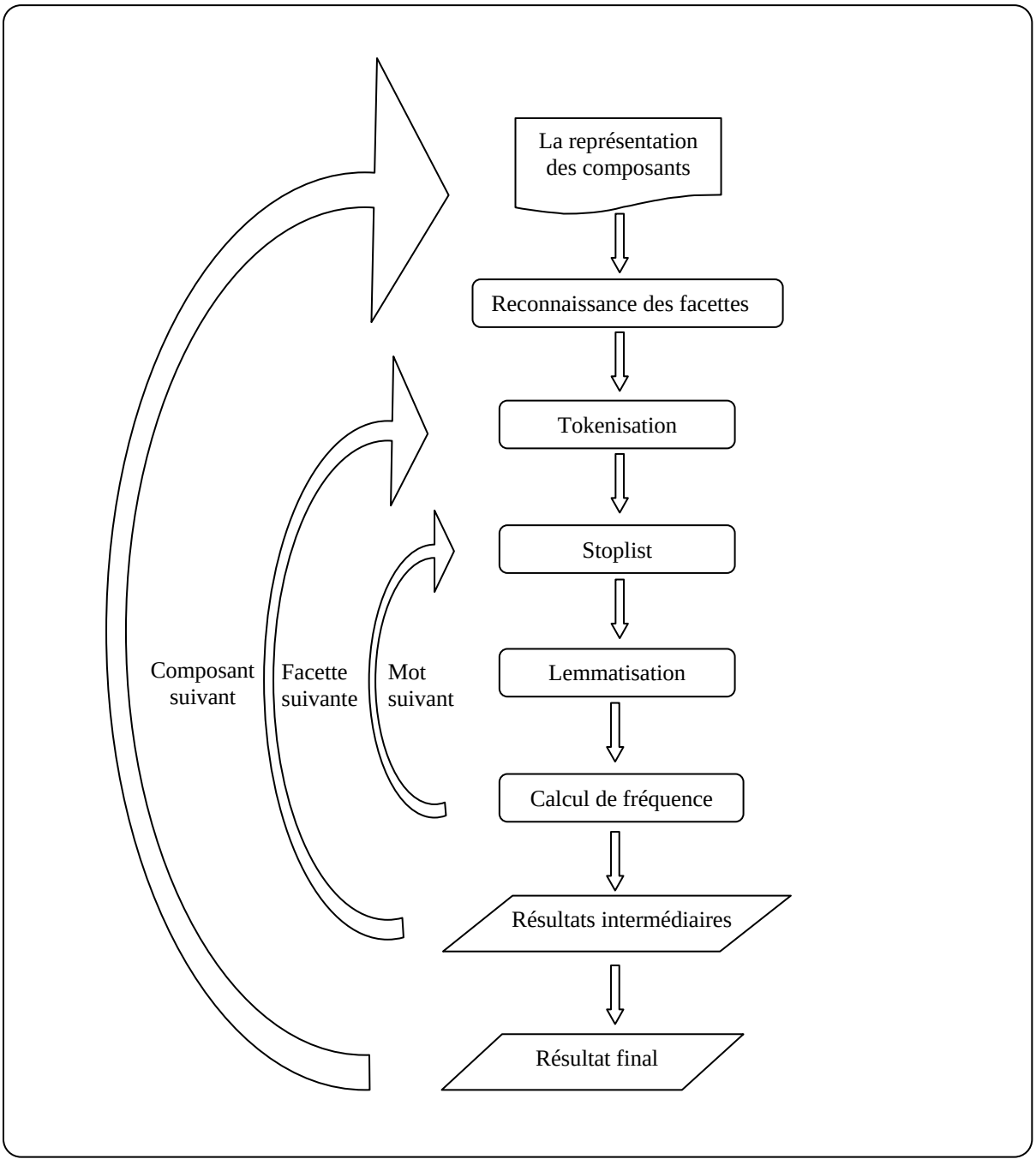

Figure 5.4 : Le processus d'indexation

L'indexation du composant et l'indexation de requête ont beaucoup de similitudes, à quelques détails près. Les étapes sont présentées dans ce qui suit :

- **Reconnaissance de facettes**: Pour traiter un composant donné, on traite séparément l'ensemble de ses facettes, telles qu'elles sont représentées dans notre base de description de composants.
- **Tokenisation**: Pour chaque facette, on doit d'abord la découper en mots. Cela consiste à reconnaître les séparateurs de mots, comme ",", ";", l'espace… etc. Le résultat de ce traitement est une suite de "tokens" ou mots (dans un sens large).
- **Stoplist**: Les mots trouvés vont être comparés avec la stoplist qui contient l'ensemble de mots qu'on ne veut pas garder dans l'index, ce processus représente un nettoyage des mots non significatifs. Cette liste de mots vides peut contenir par exemple : elle, ici, ceci, des...etc.
- **Lemmatisation**: Ce traitement consiste à transformer une forme de mot en une certaine forme standard. Nous avons fait appel à l'algorithme de Porter, qui permet d'extraire le radical des tokens générés de l'étape précédente. C'est avec cette forme standard obtenue que la comparaison avec la requête s'effectue (après lui avoir effectué le même processus)
- **Calcul de fréquence**: Pour chaque mot, on doit calculer sa fréquence d'apparition dans la description. Nous avons utilisé pour ce faire le calcul du CF basé sur la sémantique (présenté dans le point III.2.3.2), Ainsi, on calcule la fréquence d'apparition d'un concept ainsi que les mots qui le composent.
- **Itérations:** Dans la figure, on retrouve trois itérations dans l'indexation: une première itération pour traiter chaque mot, une seconde itération pour chaque facette, et une ultime itération pour chaque composant.
- **Résultat:** Le résultat d'indexation pour un composant est un ensemble d'entrées contenant les mots significatifs ainsi que leur fréquence d'apparition.

Une fois cette étape est achevée, une correspondance entre le vecteur requête (qui contient des valeurs pour chaque facette ou un ensemble de facettes introduites par l'utilisateur) et le résultat de l'indexation est effectuée.

IV.6 La sélection des composants logiciels

C'est l'étape qui suit le processus de recherche, elle consiste en une pondération de l'ensemble des facettes. Cette pondération effectué par l'association de deux techniques d'aide à la décision multi critères WSM pour *Weighted Scoring Method* et le AHP pour *Analytic Hierarchy Process.* Nous avons adopté une combinaison telle quelle est définie dans [Bar, al 08], utilisation de AHP qui aide à décrire le besoin de manière organisée, en le décomposant en un arbre hiérarchique de critères et de sous-critères d'évaluation. A l'intérieur de chaque critère on détermine l'importance de chaque sous-critère par rapport aux autres. Ensuite, on utilise la méthode WSM pour évaluer chaque candidat en additionnant les scores locaux à l'intérieur de chaque nœud et en propageant les sommes obtenues jusqu'à la racine de l'arbre pour obtenir le score total du candidat.

L'interface de recherche, que nous avons développée, offre à l'utilisateur un moyen pour lui permettre d'associer un degré d'importance pour une facette donnée. Par défaut, le système utilise la pondération générée après l'étude des différents travaux traités, sinon il peut préciser si ce poids doit être augmenté (ou diminué) s'il juge que la facette est d'une grande (ou petite) importance selon son contexte de recherche.

Une fois le calcul des scores intermédiaires et final est effectué, Les composants sont affichés à l'utilisateur, et classés par ordre de pertinence.

Si l'utilisateur juge avoir trouvé le composant recherché, il peut visualiser la description détaillée du composant pour plus d'informations, puis entame la procédure nécessaire pour l'acquisition du composant, sinon un mécanisme de reformulation de requête est appelé. Ce mécanisme permet de mémoriser et de réafficher toutes les données introduites par l'utilisateur lors de sa requête. Ainsi, il peut supprimer, ajouter ou modifier une ou plusieurs valeur (s) pour élargir, affiner ou modifier sa requête.

Nous avons également envisagé une deuxième interface de recherche, valable pour une recherche simple de composants, l'utilisateur n'a pas à saisir de mot clé, il peut naviguer dans le contenu de la base en spécifiant la facette à utiliser. Ce type de recherche convient aux utilisateurs n'ayant pas d'idée claire sur le composant à rechercher.

IV.7 Evaluation de notre proposition

Afin de tester notre système, nous avons rassemblé une collection de composants, pris de divers sites de marché de composants tels que : componentsource [Com 09], osgi [Osg 09], sourceforge [Sou 09], vclcomponent [Vcl 09]...etc, puis transformé, selon notre modèle de description.

Pour évaluer notre système, nous avons fait appel aux techniques classiques d'évaluation, qui consistent à calculer le taux de rappel, de précision et le temps d'exécution pour un ensemble formé d'une vingtaine de requêtes.

IV.7.1 Le rappel

Le rappel est le rapport du nombre de composants pertinents retrouvés par le système sur le nombre de composants pertinents disponibles. L'équation correspondante est la suivante [Bou 07] :

$$Rappel = \frac{Le\ nombre\ de\ composants\ pertinents\ retrouvés}{Le\ nombre\ de\ composants\ pertinents\ dans\ la\ base} \qquad [9]$$

IV.7.2 La précision

La précision est la proportion de composants pertinents parmi les documents sélectionnés. L'équation correspondante est la suivante [Bou 07] :

$$Précision = \frac{Le\ nombre\ de\ composants\ pertinents\ retrouvés}{Le\ nombre\ de\ composants\ retrouvés} \qquad [10]$$

En d'autres termes :
Le rappel représente la capacité d'un système à sélectionner tous les composants pertinents de la collection. La Précision représente la capacité d'un système à sélectionner que les composants pertinents

IV.7.3 Le temps d'exécution

Le temps d'exécution représente le temps total consommé par le système pour l'achèvement de toutes les opérations nécessaires pour le traitement de la requête.

Pour estimer ce temps d'exécution, il est nécessaire de préciser les caractéristiques de la machine qui a servi à faire le test. Dans notre cas, nous avons utilisé : Un processeur Dual Core 2,20 Ghz, 2 Go de RAM avec Windows XP SP2.

Les résultats d'évaluation obtenus par notre système pour les trois critères cités précédemment, sont présentés sur la figure ci-dessous :

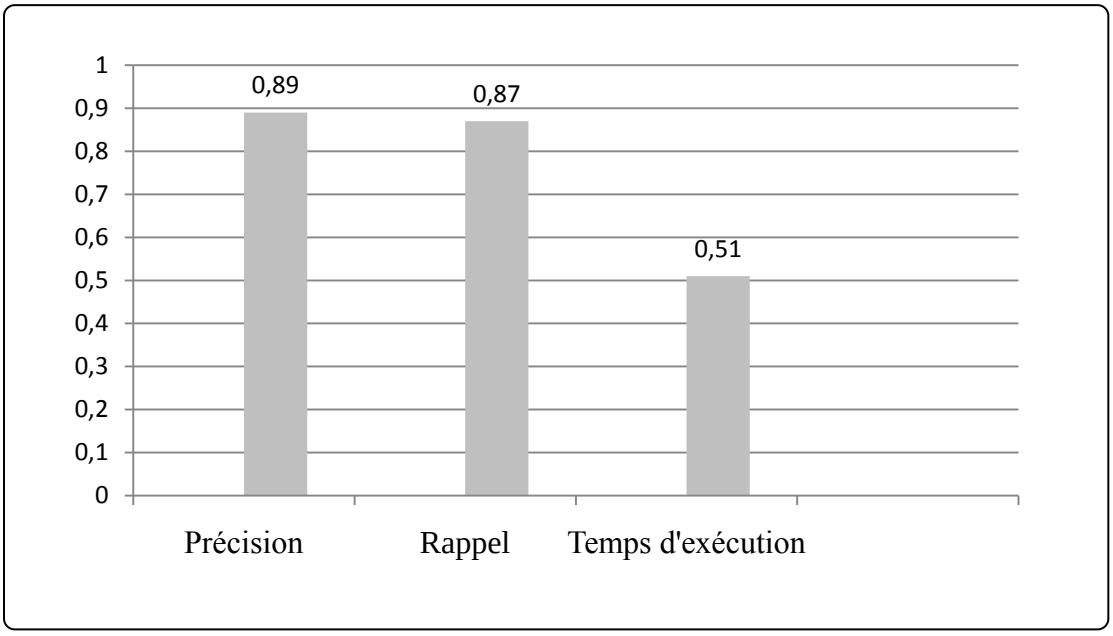

Figure 5.5 : Résultats de l'évaluation de notre travail

IV.8 Validation

Afin de valider notre travail, nous avons pensé à utiliser le Benchmark, qui représente la meilleure solution pour valider notre approche de recherche. N'ayant pas trouvé de Benchmark pour les composants logiciels, nous avons opté pour la solution suivante : qui consiste à implémenter deux approches, à savoir celle de Morisio [Mor, Tor 02] basée sur la description à base de facettes et celle de Girardi [Gir, Ibr 93] basée sur la description en langage naturel. L'idée derrière ce choix est de pouvoir évaluer notre solution qui consistait à fusionner à la fois, l'approche à base de facettes et celle en langage naturel en la comparant aux deux approches prises séparément.

Les tests ont été faits sur la même base de collections et avec les mêmes requêtes de test. Les résultats sont illustrés sur les figures (5.6 et 5.7) :

Selon les résultats obtenus par l'approche en langage naturel (Figure 5.6), nous remarquons que la recherche à base de cette approche donne de bons résultats pour le critère de rappel mais moins satisfaisants pour le critère de précision. Quant au temps d'exécution il reste très satisfaisant.

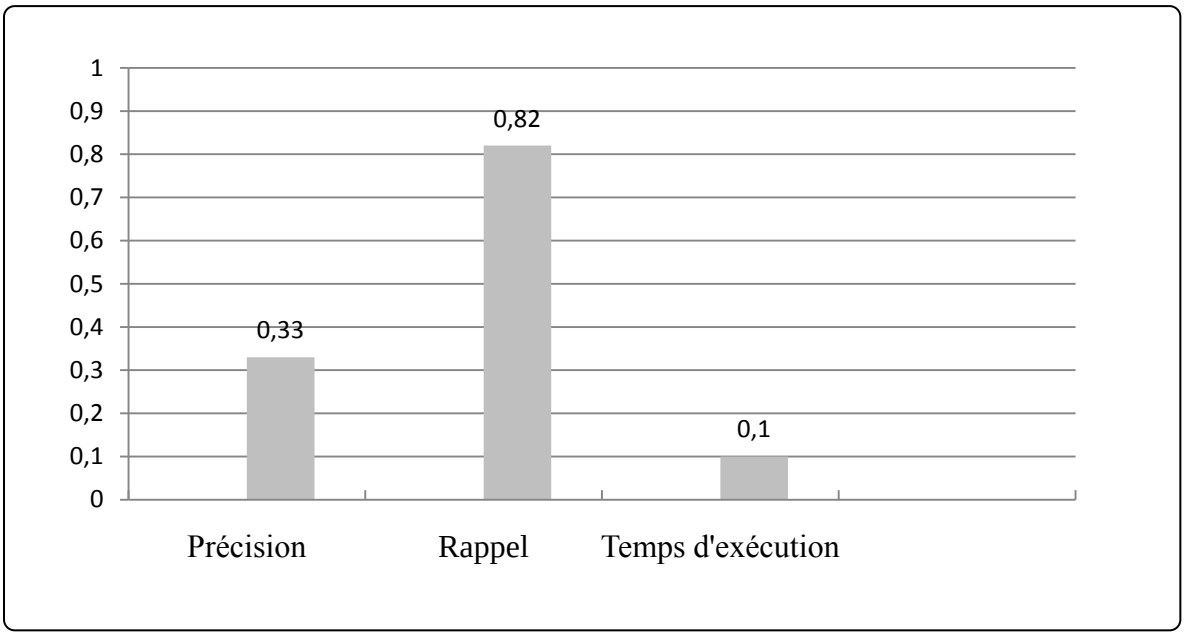

Figure 5.6 : Résultats de l'évaluation du travail de [Gir, Ibr 93]

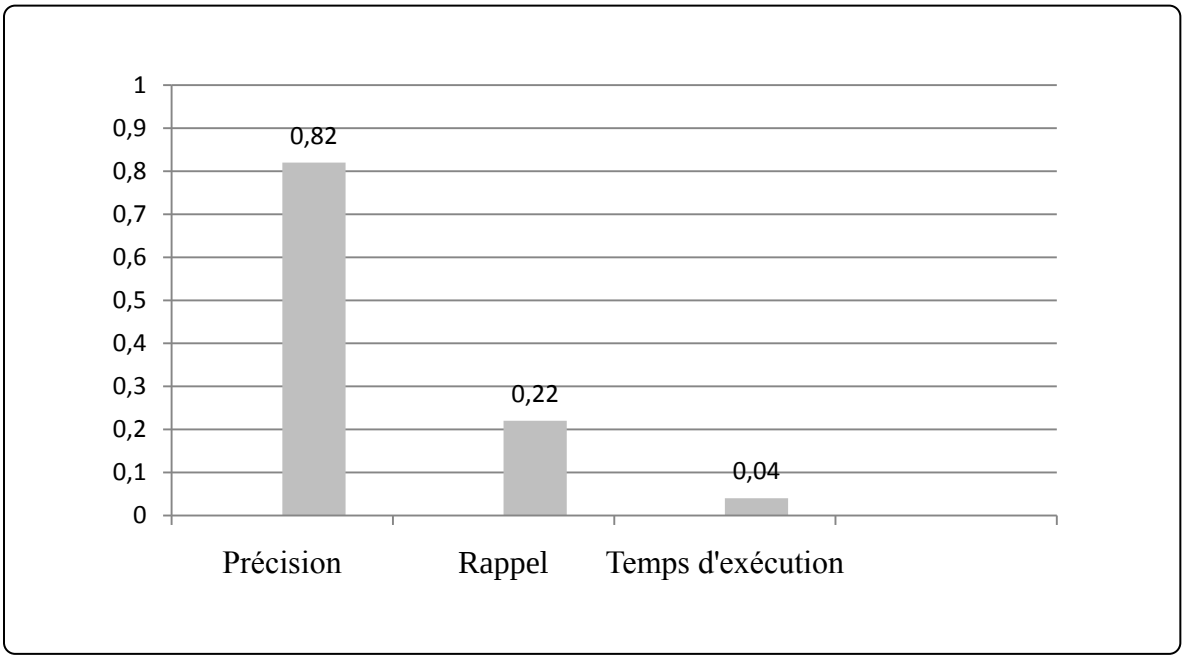

Figure 5.7 : Résultats de l'évaluation de l'approche [Mor, Tor 02]

Contrairement à l'approche en langage naturel, l'approche à base de facettes donne des résultats en terme de précision plus satisfaisants qu'en terme de rappel. Concernant le temps d'exécution, il reste très réduit.

IV.9 Discussion

La recherche en langage naturel possède un bon rappel mais avec une petite précision. En d'autres termes, l'approche réussie à extraire de la collection la majorité des composants répondants à la requête, mais laissant s'infiltrer beaucoup de composants n'ayant pas de relation avec la requête. Ceci est dû au traitement linguistique effectué à la description du composant, qui permet de retourner tous les composants ayant des termes figurant dans la requête.

Quant à l'approche à base de facettes de Morisio, elle offre une bonne précision avec un petit rappel. En d'autres termes, la plupart des composants retournés par le système sont pertinents, mais laissent beaucoup de composants répondant à la requête dans la base. Les composants retournés sont ceux qui correspondent exactement à tous les mots clés précisés dans la requête, ceci peut être expliqué par le fait que l'approche se base sur une classification en vocabulaire contrôlé et utilise des mots clés prédéfinis. Le temps d'exécution est minime par rapport aux deux approches, puisque le système ne fait pas d'étude syntaxique ou sémantique pour le traitement de la requête.

Notre approche a donné de meilleurs résultats ; nous avons pu avoir à la fois un meilleur rappel et une meilleure précision mais avec un temps de recherche relativement important. Le premier point peut être expliqué par la combinaison des deux approches citées précédemment, la description du composant est scindée en facettes, qui à leur tour sont scindées en deux ensembles, les facettes basées sur un vocabulaire contrôlé et celles basées sur un vocabulaire non contrôlé et c'est sur ces dernières que nous avons appliqué les techniques de recherche d'information basées sur la sémantique, avec une interface de recherche orientée facettes pour restreindre les mots clés de la requête à la facette correspondante. Quant au temps consommé pour la recherche des composants, il peut être expliqué par les différentes étapes de traitement que nous avons soumis à l'ensemble des facettes, à savoir : la reconnaissance de facettes, la tokenisation, l'épuration, la lemmatisation, la fréquence conceptuelle et l'application des méthodes de sélection.

IV.10 Conclusion

Nous avons présenté dans ce chapitre les divers détails de notre approche. En premier lieu, la description du composant, qui est une description générée par l'association de la description à base de facettes et la description en langage naturel. Nous avons présenté par la suite, les différents travaux qui nous ont permit de mettre en relief l'ensemble de nos facettes, son processus de recherche et de sélection. Nous avons enfin présenté les résultats d'évaluation de notre système concernant le rappel, la précision et le temps d'exécution. Ceux ci ont été comparés aux approches de Morisio et de Girardi. Les résultats obtenus ont été satisfaisants. Pour le rappel, nous avons eu un taux de 0.89 alors qu'il est de 0.82 et de 0.33 pour les approches de Girardi et Morisio (resp). Quant à la précision, nous avons eu un taux de 0.87 alors qu'il est de 0.22 et de 0.82 pour les approches de Girardi et Morisio (resp). Ces taux nous ont permit de conclure sur l'efficacité de notre système.

Conclusion générale et perspectives

Conclusion générale et perspectives

Le présent travail s'intéresse au domaine de la réutilisation et de l'assemblage des composants logiciels. Particulièrement, notre intérêt s'est penché sur la difficulté de retrouver les composants logiciels qui satisfassent nos besoins lors d'une nouvelle exploitation. Pour bien cerner cette problématique, nous avons élaboré un état de l'art sur les différents concepts et travaux qui lui sont liés, notamment :

Les composants logiciels : cités dans le premier chapitre, en présentant les différents concepts qui lui sont attachés,

La réutilisation : abordée dans le deuxième chapitre, en présentant ses deux aspects : à savoir : le design for reuse et le design by reuse.

La description, la recherche et la sélection de ces composants logiciels : présentés dans le troisième chapitre, en présentant pour chaque point les différentes tendances de recherche pour pouvoir tracer la nôtre. Dans ce chapitre nous avons cités comme techniques de recherche celles basées sur la description externe des composants, celles basées sur les approches formelles (l'appariement de signatures et de spécifications) et celle basées sur l'analyse du comportement des composants. Nous avons opté pour la description externe du composant puisqu'elle est la seule qui permet de cerner le maximum de composants logiciels.

Ainsi, nous avons proposé une nouvelle approche de caractérisation de composants logiciels. L'idée adoptée consiste à jumeler deux tendances de recherche à savoir : la description à base de facettes et la représentation en langage naturel. Les résultats de l'évaluation étaient satisfaisants. La technique utilisée a réussi à retenir les points forts des deux approches. Les résultats obtenus étaient bons en terme de rappel et de précision mais moins bons en terme de temps de recherche.

Nous pouvons considérer que cette étude représente un premier pas pour faciliter la réutilisation de composants logiciels. Néanmoins, il convient d'achever ou d'approfondir certains points, à savoir :

La manière d'optimiser notre processus de recherche afin de réduire le temps d'exécution de la requête de l'utilisateur.

Il serait également intéressant d'étendre notre travail pour toucher à la fois les différentes phases du processus de réutilisation de composants logiciels.

Il serait aussi intéressant de prévoir un processus permettant d'alimenter automatiquement les bases descriptives des composants logiciels, directement à partir de la

documentation fournie par le concepteur, ou d'en définir un modèle global universel que l'ensemble de tous les concepteurs devrait respecter.

Nous pensons enfin qu'il est intéressant de pouvoir lier les différents marchés de composants logiciels. Ainsi, toutes les recherches des utilisateurs effectuées sur un marché de composants quelconque auront la possibilité de retrouver les composants situés dans d'autres marchés sur le net.

Références

[Alm, al 07] E. S. D. Almeida, A. Alvaro, V. C. Garcia, J. C. C. P. Mascena, V. A. D. A. Burégio, L. M. D. Nascimento, D. Lucrédio, S. L. Meira; "C.R.U.I.S.E: *Component Reuse in Software Engineering*"; C.E.S.A.R e-book, Avril 2007.

[And, Car 01] J. Anderson, J. P. Carballo*; "The nature of indexing: how humans and machines analyze messages and texts for retrieval: part ii: machine indexing and the allocation of human versus machine effort*"; Information Processing and Management 37, pp 255–277. Tarrytown, NY, USA, Pergamon Press, Inc. 2001.

[Acc 02] Projet ACCORD ; *"Etat de l'art sur les Langages de Description d'Architecture (ADLs)*, Livrable 1.1-2. Juin 2002.

[Bar, al 08] G. Bart, R. Fleurquin, S. Sadou; *"A Component Selection Framework for COTS Libraries";* CBSE 2008, LNCS 5282, pp. 286–301, Springer-Verlag Berlin Heidelberg, 2008.

[Bas, al 07] G. Bastide, A. Seriai, M. Oussalah; "*Restructuration de composants logiciels : Une approche d'adaptation structurelle de composants logiciels monolithiques basée sur leur refactorisation*"; RSTI - L'Objet - Special issue on Software Evolution, vol. 13, n°1, pp. 81-116, Hermès/Lavoisier Editions, 2007.

[Baz 06] M. Baziz; *"Utilisation des mesures de similarité pour la représentation / recherche d'information textuelle";* Équipe SIG Séminaire ERSS-IRIT IRIT-SIG février 2006.

[Boe 87] B. W. Boehm ; *"Improving Software Productivity";* Computer, Vol. 20, No. 9, pp. 43-57, Septembre. 1987.

[Boo, al 03] S. Boonsiri, R. C. Seacord, D. A. Mundie*; "Rule-Based COTS Integration"*; Journal of Research and Practice in Information Technology, ACM Society Vol 35, No: 3, pp. 197-204, Août 2003**.**

[Bou 07] M. Boughanem ; "*Recherche d'information* "; Université Paul Sabatier de Toulouse, Laboratoire IRIT. Support de cours présenté à l'INI, Avril 2007.

[Buc, Wec97] M. Bünchi, W. Weck; *"A Plea for Grey-Box Components*"; Workshop on Foundations of Component-Based Systems, Zürich, Septembre 1997.

[Car, Lon 00] D. Carney, F. Long; *"What do you mean by COTS"; IEEE Software*, Vol 17, Issue: 2, pp. 83-86, Avril 2000.

[Chi, Cro 90] E. Chikofsky, J. Cross; *"Reverse Engineering and Design Recovery: A Taxonomy";* IEEE Software, Vol. *7*, No.1, pp. 13-17; Janvier 1990.

[Chi 98] Y. Chikhi; *"Réutilisation des structures de données dans le domaine des réseaux électriques*"; Thèse de doctorat, Université Paris 6, Juillet 1998.

[Col 04] *"Component Adaptation and Assembly"*, Page designed by Quadrifoglio, Component-Based Software Development; Mai 2004.
Disponible sur http://cbs.colognet.org/adaptation.php

[Com 09] www.componentsource.com. Consulté en Avril 2009.

[Cou 06] T. Coupaye, V. Quéma, L. Seinturier, J.-B. Stefani ; "*Chapitre 3:Le système de composants Fractal";* Intergiciel et Construction d'Applications Réparties, ICAR (Ed.), Juillet 2006

[Dep 05] A.M. Déplanche; *"Composants et Architectures Temps Réel";* Rapport de synthèse CNRS, Département STIC – Proposition d'Action Spécifique, AS 195 - Mars 2005.

[Dia 91] R. P. Diaz, *"Implementing Faceted Classification for Software Reuse*", Communications of the ACM, Vol. 34, No. 5, 1991.

[Dic 08] Le grand dictionnaire terminologique, disponible sur *http://www.granddictionnaire.com*

[Don, al 99] J. Dong, P. S. C. Allencar, D. D. Cowan; *"A component specification template for COTs based software development*"; 1st Workshop on Ensuring Successful COTS Development, USA, 1999.

[Gae, Tur 00] M. Gaedke, K. Turowski; *"Standardizing the Web Composition Process Model – Disciplined Development and Evolution of Component Based Web Applications";* Allemagne, Juillet 2000.

[Gac, Ser 03] A. Gacemi, D. Seriai; "*La réutilisation : concepts et techniques*"; Technical Report, Ecole des Mines de Douai, France, Avril 2003.

[Gil 06] N. S. Gill; *"Importance of Software Component Characterization for Better Software Reusability"*; ACM SIGSOFT Software Engineering Notes. Vol 31, Issue 1, Janvier 2006.

[Gir, Ibr 93] M. R. Girardi, B. Ibrahim; *"A software reuse system based on natural language specifications"*; Proceeding of International Conference on Computing and information (ICCI'93), pp 507-511, Sudbury, Ontario, Canada, 1993.

[Gor 99] M.Gorman; *"Metadata or cataloguing? a false choice";* Journal of Internet Cataloging, Vol. 2 No.1, pp.5-22. 1999.

[Ha, Tra 05] V. Ha, K. V. Tran; *"An empirical study of software components reuse in Statoil: An introduction";* Thèse de doctorat. Université IDI NTNU, Norvège; Décembre 2005.

[Had 03] S. Hadjab*;* "*Assemblage de Composants Logiciels : Les Composants Composites*"*; Mémoire* de stage, Université de Technologie de Troyes-UTT, *Juillet 2003*.

[Her 05] C. Herault; " *Adaptabilité des Services Techniques dans le Modèle à Composants*", Thèse de doctorat, Université de Valenciennes**,** Juin 2005.

[Her, Lin 01] D. Hemer, P. Lindsay; "*Specification based Retrieval Strategies for Module Reuse"*; Proceedings Australian Software Engineering Conference (ASWEC'01), pp 235-243, Canberra, Australie, IEEE Computer Society 2001.

[Kei, Tiw 05] M. Keil, A. Tiwana; *"Beyond Cost: The Drivers of COTS Application Value"*; IEEE Computer Society. Vol. 22, No. 3, pp. 64-69, Juin 2005**.**

[Kha 05] O. Khayati; " *Modèles formels et outils génériques pour la gestion et la recherche de composants* "; Thèse de doctorat, Institut National Polytechnique de Grenoble, Décembre 2005

[Kha, al 02] O. Khayati, A. Conte, J. P. Giraudin; *"Les systèmes de recherche de composants*"; Journées des systèmes à composants adaptables et extensibles Grenoble 2002.
Disponible sur http://arcad.essi.fr/2002-10-composants.html.

[Kie, Mul 07] H. M. Kienle, H. A. Müller; "*A Lightweight Taxonomy to Characterize Component-Based Systems"*; Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems (ICCBSS'07); pp 192-204, IEEE Computer Society. Mars 2007.

[Kim 05] W. Kim; "*On Issues with Component-Based Software Reuse*"; Journal of Object Technology, Vol. 4, No. 7, September-October 2005, pp. 45-50 Disponible sur http://www.jot.fm/issues/issue_2005_09/column5

[Kon 96] J. Kontio; "*A Case Study in Applying a Systematic Method for COTS Selection";* 18th International Conference on Software Engineering (ICSE'96), pp.201, IEEE, Mars 1996.

[Kru 92] C. W. Krueger*; "Software reuse*"; ACM Computing Surveys (CSUR). Vol 24, Issue 2, Juin 1992.

[Li, al 09] Y. Li, L. Zhang, B. Xie, J. Sun; *"Refining component description by leveraging user query logs"*; The Journal of Systems and Software 82 751–758. Science Direct 2009.

[Lio 05] W. J. Lioyd; *"A Common Criteria Based Approach for COTS Component Selection*"; *Journal of Object Technology*, Vol. 4, no. 4, pp. 27-34, April 2005, Special issue: 6th GPCE Young Researchers Workshop 2004, http://www.jot.fm/issues/issue_2005_04/article7.

[Luc, al 04] D. Lucredio, A. F. do Prado, E. S. de Almeida; "*A Survey on Software Components Search and Retrieval "; Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04)*, IEEE Computer Society, 2004.

[Nei 80] J. M. Neighbors; ”*Software Construction Using Components*”; Thèse de doctorat, Department of Information and Computer Science, Université de Californie, Irvine, 1980

[Nin 97] J. Q. Ning; *"CBSE Research at Andersen consulting";* U.S. National Institute of Standards and Technology’s Advanced Technology Program on Component-Based Software. Février 1997**.**

[Mac 04] K. Macedo De Amorim; “*Modélisation d’aspects qualité de service en UML : application aux composants logiciels*”*;* Thèse de doctorat, Université de Rennes 1, Mai 2004**.**

[McI 68] M. D. McIlroy; *“Mass produced software components”;* Proceedings, NATO Conference on Software Engineering, Allemagne, Octobre 1968.

[Mas, al 06] A. Masmoudi, R. Champagne, G. Paquette; *" Recherche des composants logiciels référencés par un modèle ontologique";* Actes du 24ème Congrès INFORSID, Hammamet, Tunisie, Juin 2006.

[Mas, al 08] A. Masmoudi, R. Champagne, G. Paquette; *"Ontologie et modèle relationnel pour le référencement, la recherche et l’agrégation des composants logiciels "; JFO 2008 Lyon, France, ACM publication. Décembre 2008*.

[Mil, al 97] R. Mili, A. Mili, R. T. Mittermeir*; "Storing and Retrieving Software Components: A Refinement Based System";* IEEE transactions on software engineering, Juillet 1997.

[Mil, al 03] H. Mili, E. Ah-Ki, R. Godin, H. Mcheick; " *An experiment in software component retrieval ";* Information and Software Technology, Vol 45, Issue 10, 15 pp 633-649. Science Direct, Juillet 2003.

[Mor, Tor 02] M. Morisio, M. Torchiano; "*Definition and classification of COTS: a proposal";* ICCBSS, Orlando (FL). Février 2002.

[Mos 02] M. A. Mostefai; “*Le Génie Logiciel Basé sur les composants : Méthodologies, technologies et impacts* ”; USTHB 2002.

[Ozc 07] A. E. Ozcan; “*Conception et Implantation d’un Environnement de Développement de Logiciels à Base de Composants : Applications aux Systèmes Multiprocesseurs sur Puce*”, Mars 2007.

[Ous, al 05] M. Oussalah. al; *“ Ingénierie des composants – concepts, techniques et outils* ”; Editions Vuibert Informatique, Paris, 2005.

[Osg 09] https://www.osgi.org. Consulté en Avril 2009.

[Per, al 09] A. Perini, F. Ricca, A. Susi; *"Tool-supported requirements prioritization: Comparing the AHP and CBRank methods";* Information and Software Technology, Science Direct, 2009.

[Pou 99] J. S. Poulin; *"Reuse: Been there, done that";* Communication of the ACM. Mai 1999.

[Puj, al 04] V. Pujalte, P. Ramadour, C. Cauvet; *"Recherche de composants réutilisables : une approche centrée sur l'assistance à l'utilisateur*"; Actes du 22ème congrès Inforsid, Biarritz, pp. 211-227, 25-28 Mai 2004**.**

[Puj, al 06] V. Pujalte–Busseuil, P. Ramadour, E. Tranvouez ; "*Conception par réutilisation : des composants aux agents composants* "; Actes du 24ème Congrès INFORSID, Hammamet, Tunisie, Juin 2006.

[Puj,Ram 04] V. Pujalte, P. Ramadour; "*Réutilisation de composants : un processus interactif de recherche*"; Actes Majecstic'05, Calais, 2004.

[Ram 01] P. Ramadour; "*Modèles et langage pour la conception et la manipulation de composants réutilisables de domaine*", Thèse de doctorat, Université Aix-Marseille III, 2001.

[Reb 06] I. Rebai; *"Conception et développement d'une plate-forme permettant la capitalisation de composants logiciels pour les environnements d'apprentissage humain"*; Thèse de doctorat, Université René Descartes-Paris V, *Décembre 2006*.

[Reb, al 04] I. Rebaï, J. M. Labat; *"Des métadonnées pour la description des composants logiciels pédagogiques";* Technologies de l'Information et de la Communication pour l'Education – TICE, 2004.

[Reb, al 05] I. Rebaï, N. Maisonneuve, J. M. Labat; "*Un entrepôt pour stocker et rechercher des composants logiciels utiles aux EIAH"; Revue STICEF*, Revue Sciences et Technologies de l´Information et de la Communication pour l´Éducation et la Formation, 2005.

[Red, al 05] G. Redolfi, L. Spagnoli, P. M. Cristal, A. Espindola; "*A Reference Model for Reusable Components Description*"; Proceedings of the 38th Hawaii International Conference on System Sciences, IEEE computer, Janvier 2005.

[Sam 97] J. Sametinger; "*Software Engineering with Reusable Components*"; Springer-Verlag, Mars 1997.

[San, al 07] P. S. Sandhu, H. Singh, B. Saini; "*A New Characterization Scheme of Reusable Software Components*"; IJCSNS International Journal of Computer Science and Network Security, Vol.7, No.8, Août 2007.

[Sas, al 03] S. B. Sassi, L. L. Jilani, H. H. B. Ghezala; *"COTS Characterization Model in a COTS-Based Development Environment";* Proceedings of the Tenth Asia-Pacific Software Engineering Conference, APSEC'03, IEEE Computer Society 2003**.**

[

[Şen, Bar 06] C. G. Şen, H. Baraçli; *"A Brief Literature Review of Enterprise Software Evaluation and Selection Methodologices : A Comparison in the Context of decision-Making Methods"*; Proceedings of the 5th International Symposium on Intelligent Manufacturing Systems, 2006.

[Sin, al 07] P. S. Sandhu, H. Singh, B. Saini; *"A New Characterization Scheme of Reusable Software Components ";* International Journal of Computer Science and Network Security, IJCSNS, Vol 7, No.8, Août 2007.

[Sou 09] http://www.sourceforge.net. Consulté en Avril 2009.

[Szy 02] C. Szyperski; "Component *Software: Beyond Object Oriented programming*"; Second Edition. Addison Wesley, 2002.

[Teb 04] H. Tebri; *"Formalisation et spécification d'un système de filtrage incrémental d'information";* Thèse de doctorat, Université Paul Sabatier de Toulouse, Décembre 2004.

[Vcl 09] http://www.vclcomponents.com. Consulté en Avril 2009.

[Wan 06] L. Wang; *"A Collaboration Framework of Selecting Software Components based on Behavioral Compatibility with User Requirements";* Thèse de doctorat, Université BOND, Queensland, Australie. Novembre 2006.

[Yac, al 99] S. Yacoub, H. Ammar, A. Mili; "*Characterizing a Software component"*; International workshop on component-based software engineering, Los Angeles, CA, USA, Mai 1999.

[Zha, al 06] Y. Yang, W. Zhang, X. Zhang, J. Shi; *"A Weighted Ranking Algorithm for Facet-based Component Retrieval System ";* International Association Of Science And Technology For Development Proceedings of the 2nd IASTED international conference on Advances in computer science and technology. ACM Publications, 2006.