République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique

UNIVERSITE Dr. TAHAR MOULAY SAIDA

FACULTE : TECHNOLOGIE

DEPARTEMENT : INFORMATIQUE

MEMOIRE DE MASTER

OPTION : RÉSEAUX INFORMATIQUES ET SYSTÈMES RÉPARTIS

Thème

Développement d'un système de recherche de composants logiciels

Présenté par :
BOUTALEB Yamina
ZELLAG Saliha

Encadré par :
Dr. Mebarka YAHLALI

Promotion : 2017-2018

Remerciements

Avant tout nous remercions "Allah" tout puissant qui nous a donné la force, la capacité et la patience d'effectuer ce projet de fin d'étude.

Nous tenons à exprimer notre profonde gratitude à Dr.Yahlali Mebarka, pour son encadrement, sa disponibilité, son soutien, ses orientations lors de la réalisation ce mémoire.

Nous remercions tous les enseignants de l'université « Dr. moulay tahar –saida » et surtout les enseignants du département informatique qui ont participé à notre formation pendant toutes ces années universitaires.

Nous remercions vivement les honorables membres du jury pour leur aimable acceptation de juger notre travail.

Enfin nous adressons nos remerciements les plus sincères à tous ceux qui ont contribués de près ou de loin à la concrétisation de ce travail.

Merci à tous et à toutes.

Dédicace

Je dédie ce modeste travail :

A mes très chers Parents qui ont toujours été là pour moi, que «ALLAH » les protège, pour leur soutien, leurs encouragements et leur sacrifice durant toute ma vie.

A mes très chère sœur et frères qui mont soutien durant cette mémoire.

A Ma chère binôme « Saliha » et à toute sa famille.

A Mes chers amis pour tous ce qu'on a partagés ensemble.

A tous mes Professeurs Enseignants, je leurs exprime mes vifs remercîments.

Boutaleb Yamina

Dédicace

Je dédie ce modeste travail :

Un grand remerciement à mon mari pour sa compréhension, sans oubliant mon fils « Mohammed Assem »et ma fille « Ranim Assaouir »

A mes très chers Parents qui ont toujours été là pour moi, que «ALLAH » les protège, pour leur soutien, leurs encouragements et leur sacrifice durant toute ma vie.

A mes très chère sœurs et frères qui mont soutien durant cette mémoire.

A Ma chère binôme « Yamina » et à toute sa famille.

A Mes chers amis pour tous ce qu'on a partagés ensemble.

A tous mes Professeurs Enseignants, je leurs exprime mes vifs remercîments.

Zellag Saliha

Abstract

Component engineering is a very promising and growing discipline in the academic and industrial fields. The component-oriented development approach consists of designing and developing applications by assembling pre-existing components. This approach attempts to reduce development costs and deadlines, facilitate the evolution and maintenance of applications by promoting adaptability to produce new features.
Our thesis presents the fundamental models, concepts and mechanisms on which it relies the development of software applications based on reusable components. It proposes a strategy for the search of the software components to obtain all the relevant components corresponding to the needs of a user (functional aspect).

Keywords :

software components, software architecture, component search

Résumé

L'ingénierie des composants est une discipline très prometteuse en pleine croissance dans les domaines académique et industriel. L'approche de développement orientée composants consiste à concevoir et développer des applications par assemblage de composants préexistants. Cette approche tente de réduire les couts et délais de de développement, de faciliter l'évolution et la maintenance des applications en favorisant l'adaptabilité pour produire de nouvelles fonctionnalités.
Notre mémoire présente les modèles, concepts et mécanismes fondamentaux sur lesquels elle repose le développement d'applications logicielles basées sur des composants réutilisables.et propose une stratégie pour la recherche des composants logiciels pour obtenir tous les composants pertinents correspondant aux besoins d'un utilisateur (aspect fonctionnel).

Mots-clés :

composant logiciel , architecture logiciel , reherche d'un composant .

ملخص

تعد هندسة المكونات من الانضباط الواعد والمتنامي في المجالات الأكاديمية والصناعية. يتكون منهج التطوير الموجه للمكونٍ من تصميم وتطوير التطبيقات من خلال تجميع المكونات الموجودة مسبقاًا. يحاول هذا النهج تقليل تكاليف التطوير والمواعيد النهائية ، وتسهيل تطور التطبيقات وصيانتها من خلال تعزيز القدرة على التكيف لإنتاج ميزات جديدة.
تقدم أطروحاتنا النماذج والمفاهيم والآليات الأساسية التي تعتمد عليها في تطوير تطبيقات برمجية مبنية على مكونات قابلة لإعادة الاستخدام ، وتقترح استراتيجية للبحث عن مكونات البرنامج للحصول على جميع المكونات ذات الصلة المقابلة لاحتياجات المستخدم (الجانب الوظيفي).

الكلمات المفتاحية :

مكونات البرامج ، هندسية البرمجيات ، البحث عن المكونات

Table des matières

Introduction Générale

Les logiciels envahissent toute l'activité mondiale (télécommunications, transports, énergie, gestion, transactions financières et administratives, jeux, santé,...) et leur part devient prépondérante dans l'économie. C'est pourquoi le coût d'un logiciel doit être maîtrisé (productivité) et son temps de développement minimisé (réactivité). D'où la nécessite d'une ingénierie du logiciel.

Actuellement ,les systèmes logiciels deviennent de plus en plus complexes et fournissant plus de fonctionnalités. Pour pouvoir produire de tels systèmes rentables, les fournisseurs utilisent souvent des technologies basées sur les composants au lieu de développer tous les parties du système à partir de zéro .

Dans l'ingénierie des logiciels basée composant, le développement d'un logiciel se réduit à un assemblage de composants prédéfinis, chacun d'entre eux joue un rôle spécifique dans le système et définit clairement les services qu'il offre et les services qu'il requiert pour accomplir sa fonctionnalité..

La motivation derrière l'utilisation des composants était initialement de réduire le coût du développement, mais plus tard, il est devenu plus important de réduire les délais de mise sur le marché et la qualité.

Les développeurs et les testeurs éviteraient d'effectuer les mêmes activités encore et encore, c'est-à-dire qu'un ensemble d'actifs réutilisables serait utilisé pour résoudre des problèmes récurrents. De cette façon, les coûts seraient réduits, car le temps qui aurait été nécessaire pour répéter une activité pourrait être investi dans d'autres tâches pertinentes. Tout ça à travers les composants logiciels.

l'utilisation de ces composants est plus souvent motivé par des réductions possibles coûts de développement. En utilisant des composants, il est possible de produire plus de fonctionnalités avec le même investissement de temps et argent. Lorsque les composants sont introduits dans un système, de nouvelles questions doivent être traitées par ex. dynamique configurations ...etc. Une partie de ces questions sont traitées avec l'ingénierie de logiciel à base de

composants(CBSE :component based software engineering).CBSE fournit des méthodes, des modèles et des lignes directrices les développeurs de systèmes à base de composants. Dans ce mémoire on propose un mécanisme qui permet de sélectionner, parmi une vaste bibliothèque de composants, le candidat qui répond le mieux à un besoin spécifique.

Problématique :

Un des problèmes majeurs qui peuvent faire face au utilisateur et le problème de recherche de composant logiciel .Ainsi trouver la réponse à une requête d'un utilisateur, suppose avoir un ensemble de composants puis a partir de cet ensemble il s'agit de sélectionner le composant le plus adaptable pour le proposer à l'utilisateur. Plusieurs facteurs peuvent venir compliquer cette recherche comme par exemple :
-le choix du format de description du besoin : c'est le premier problème qui se pose lors d'une automatisation de la recherche et de la sélection.

objectifs :

L'objectifs de ce travail est de :
1-Développer un système de recherche de composant logiciel.
2-Proposer une structure (meta donnée) pour une déscription fonctionnelle de composants logiciels.

Organisation du mémoire :

Notre mémoire est structuré comme suit :

Le premier chapitre :

Est consacré aux notions fondamentales de composants logiciels.

Le deuxième chapitre :

dans ce chapitre les approches et méthodes proposées dans la littérature pour la recherche d'un composant logiciel sont détaillées.

Le troisième chapitre :

présente les langages de déscription d'architecture

Le quatrième chapitre :

présente la conception et l'implémentation de notre système. En fin, nous terminerons notre mémoire par une conclusion générale et nous énoncerons quelques perspectives de recherche.

CHAPITRE 1

CBSE :Component Based Software Engineering

1.1 Introduction

La notion de composant logiciel occupe aujourd'hui une partie importante du paysage informatique, notamment dans les domaines du génie logiciel, des systèmes distribués ou encore des réseaux. Comme tout concept émergent, l'ambiguïté prévaut lorsqu'il s'agit de donner une définition claire.
Issu de la recherche, le domaine des architectures logicielles propose de décrire la structure d'un logiciel comme un assemblage, ou plutôt une interconnexion, de composants. Dans cette optique, le critère de neutralité associé à la notion de composant est très fort : il n'est fait aucune supposition sur le rôle d'un composant.

1.2 Historique de composant logiciel

Le développement basé sur les composants est proche de réutiliser. L'idée de réutiliser des morceaux de logiciel provient du début des années soixante lorsque le terme Software. Les crises ont été mentionnées la première fois.
L'idée de base est simple : Lors du développement de nouveaux systèmes, utilise des composants qui sont déjà développé. Lorsque vous développez des fonctions spécifiques vous avez besoin dans votre système, développez-le de cette façon. La fonction peut être utilisée par d'autres systèmes à l'avenir. Bien que le principe soit simple, il a été démontré que la mise en œuvre est assez difficile. La réutilisation a été longue et laborieuse. L'un des premiers cas de réutilisation réussie est le développement de différentes bibliothèques, par exemple bibliothèques mathématiques. Ces bibliothèques incluent des fonctions (par exemple, des fonctions mathématiques telles que sinus, cosinus, opérations matricielles, etc.), auxquels il est fait référence dans le code source, puis lié avec le propriétaire code. Le succès de ce type d'entités réutilisables réside dans plusieurs faits [9]

- Il existe une théorie bien définie sur ces types des fonctions.
- La communication entre l'application et ces fonctions sont simples. C'est de type procédural. L'application appelle les fonctions qui lui envoient paramètres d'entrée, et la bibliothèque répond par le l'exécution de la fonction et renvoyer la sortie paramètres
- Les entrées et les sorties sont strictement définies

— Bonne gestion relative des erreurs :si les entrées sont erronée, la sortie retournera généralement une valeur indiquant une erreur.

Un inconvénient et une limitation de ces types de composants est l'inflexibilité. Pour une nouvelle version de la bibliothèque, l'application doit être reconstruite. Ce problème a partiellement résolu par l'introduction de dynamique ou de fragment bibliothèque qui peut être chargée séparément de l'application. Un autre type d'inflexibilité est les limitations des types de paramètres d'entrée et de sortie. En changeant types de paramètres (par exemple en utilisant des éléments de texte au lieu de nombres dans une fonction de tri), une nouvelle bibliothèque La fonction doit être utilisée [9].

1.3 Composant logiciel

1.3.1 Définition :

Définition1

Un composant logiciel est une unité de composition qui a, par contrat, spécifié uniquement ses interfaces et ses dépendances explicites de contextes. Un composant logiciel peut être déployé indépendamment et est sujet à composition par des tierces entités. Une autre définition de composant est due à Meyer : « un composant est un élément logiciel (unité modulaire) satisfaisant aux conditions [8]

— il peut être utilisé par d'autres éléments logiciels ,ses « clients ».

— il possède une description d'usage officielle, suffisante pour un client pour l'utiliser.

— Il n'est pas lié à un ensemble fixe de clients » [8].

Définition2

Un composant est une entité logicielle (d'implémentation), élaborée et vendue (sous forme binaire) par une société et utilisée par un (ou des) client(s) pour composer un logiciel [10].

1.4 Architecture d'un composant logiciel :

L'architecture d'un composant logiciel dans la figure suivante spécifie ses entrées et ses sorties afin de faciliter la description de son comportement (services offerts) quels que soient

les langages de programmation utilisés.Comme le présente la figure ci-dessous, un composant logiciel possède, principalement, les trois éléments suivants [4]

— 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 [4].

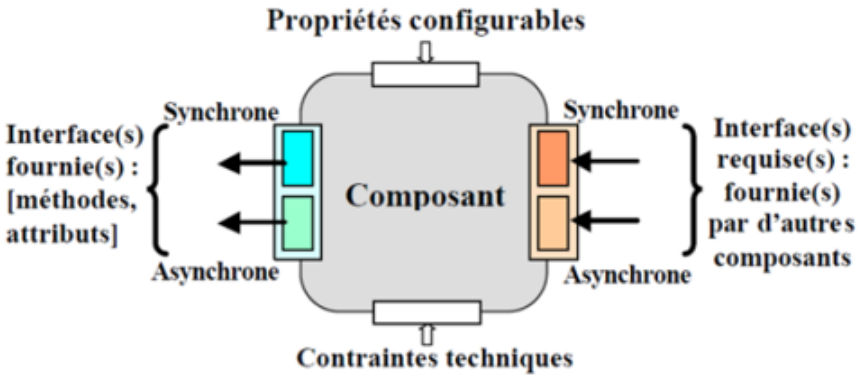


FIGURE 1.1 – architecture de composant logiciel.[4]

1.5 Les différentes abstractions d'un composant

L'abstraction d'un composant correspond à la visibilité de son implantation. On trouve trois sortes d'abstractions [27]

Boite noire (Blackbox) :

Le client ne connaît aucun détail au-delà des interfaces et de leurs spécifications.

Boîte blanche (Whitebox) :

L'implantation d'une boîte blanche est entièrement disponible et peut donc être étudiée afin d'augmenter sa compréhension. On peut trouver dans la littérature, le terme de Boite transparente (Glassbox). Quand la distinction est faite cela signifie que la boîte blanche

permet la manipulation de l'implantation alors que la boîte transparente permet simplement l'étude de l'implantation.

Boîte grise (Graybox) :

Seule une partie contrôlée de l'implantation est visible.

1.6 Caractéristiques d'un composant logiciel :

Les caractéristiques globales d'un composant définies par Medvidovic et Taylor sont les suivantes [13]

1.6.1 L'interface d'un composant

L'interface d'un composant est la description de l'ensemble des services offerts et requis par le composant sous la forme de signature de méthodes, de type d'objets envoyés et retournés, d'exceptions et de contexte d'exécution. L'interface est un moyen d'expression des liens du composant ainsi que ses contraintes avec l'extérieur.

1.6.2 Le type d'un composant

Le type d'un composant est un concept représentant l'implantation des fonctionnalités fournies par le composant. Il s'apparente à la notion de classe que l'on trouve dans le modèle orienté objet. Ainsi, un type de composant permet la réutilisation d'instances de même fonctionnalité soit dans un même architecture, soit dans des architectures différentes. En fournissant un moyen de décrire, de manière explicite, les propriétés communes à un ensemble d'instances d'un même composant, la notion de type de composant introduit un classificateur qui favorise la compréhension d'une architecture et sa conception.

1.6.3 La sémantique d'un composant

La sémantique du composant est exprimée en partie par son interface. Cependant, l'interface telle que décrite ci-dessus ne permet pas de préciser complètement le comportement du composant. La sémantique doit être enrichie par un modèle plus complet et plus abstrait permettant de spécifier les aspects dynamiques ainsi que les contraintes liées à l'architecture.

Ce modèle doit garantir une projection cohérente de la spécification abstraite de l'architecture vers la description de son implantation avec différents niveaux de raffinements. La sémantique d'un composant s'apparente à la notion de type dans le modèle orienté objet.

1.6.4 Les contraintes d'un composant

Les contraintes définissent les limites d'utilisation d'un composant et ses dépendances intra composants. Une contrainte est une propriété devant être obligatoirement vérifiée sur un système ou une de ces parties. Si celle-ci est violée, le système est considéré comme un système incohérent et inacceptable. Elles permettent ainsi de décrire de manière explicite les dépendances des parties internes d'un composant comme la spécification de la synchronisation entre composants d'une même application (dépendance intra composant).

1.6.5 L'évolution d'un composant

Les composants sont conçus pour être des éléments de conception qui évoluent. Un ADL (langage de description d'architecture) doit favoriser la modification de ses propriétés (interface, comportement, implantation) sans perturber son intégration dans les applications déjà existantes. L'évolution doit donc être simple et s'effectuer par le biais de techniques comme le sous typage ou le raffinement.

1.6.6 Les propriétés non fonctionnelles d'un composant

Les propriétés non fonctionnelles (propriétés liées à la sécurité, la performance, la portabilité) doivent être exprimées à part, permettant ainsi une séparation dans la spécification du composant des aspects fonctionnels (aspects métiers de l'application) et des aspects non fonctionnels ou techniques (aspects transactionnel, de cryptographie, de qualité de service). Cette séparation permet la simulation du comportement d'un composant à l'exécution dès la phase de conception, et de la vérification de la validité de l'architecture logicielle par rapport à l'architecture matérielle et l'environnement d'exécution.

1.7 Spécification d'un composant logiciel

La spécification d'un composant est donnée par trois éléments [1] :

— Type : c'est la définition abstraite du composant.

— Implémentation : l'implémentation d'un composant est la mise en œuvre des aspects fonctionnels et non-fonctionnels de son type.
— Instance : est une entité exécutable du composant dans un système.

La spécification d'un composant repose essentiellement sur son type qui est caractérisé par deux éléments : ses interfaces et ses propriétés.

Interfaces :

Une interface d'un composant peut être définie comme étant une spécification de ses points d'accès. Le client accède aux services fournis par le composant via ces points d'accès. Si un composant en possède plusieurs, chacun représenté un service offert par ce composant. Il est important de souligner qu'une interface n'implémente aucune de ses opérations. En effet, une interface décrit uniquement un ensemble d'opérations ainsi que leurs contrats. Il existe deux types d'interfaces : les interfaces fournies et les interfaces requises. Les interfaces fournies décrivent les services offerts par le composant a son environnement tandis que les interfaces requises spécifient les services requis par le composant [1].

1.7.1 Propriétés d'un composant :

On peut distinguer deux types de propriétés pour un composant, les propriétés dites fonctionnelles et les propriétés dites non-fonctionnelles [1] :

Les propriétés fonctionnelles :

sont celles qui décrivent la structure, le comportement et les fonctionnalités du composant [1].

les propriétés non-fonctionnelles :

concernent les services utilises par le composant qui ne font pas partie des services applicatifs (sécurité, performance,. . .)[1].

1.8 Développement à base de composant :

L'objectif principal de l'approche orientée composant est de construire les applications par assemblage de composants logiciels préexistants. De ce fait, le CBSE distingue deux cycles de vie : un pour le développement du composant (Design for reuse) et un autre pour

le développement d'un système à base de composants (Design by reuse). Ces deux démarches Sont intrinsèquement différentes [3] :

1.8.1 Design by reuse :

Ce cycle de vie comprend un nouveau processus qui vient s'ajouter à la fois à la phase de conception du système et à la phase de maintenance. Nous montrons dans la figure comment le cycle en V est adapté afin de répondre aux besoins du CBSE [3].

Le but de cette section n'est pas de détailler toutes les phases du cycle de vie mais de mettre l'accent sur les activités spécifiques au cycle « Design by reuse » du CBSE. Ces activités sont décrites comme suit [3] :

Inspection de composants.

Le processus d'analyse et de conception doit prendre en compte les besoins du système qui doivent être compatibles avec les besoins des composants disponibles. Par exemple, la sélection d'une technologie particulière doit entreprise en considération : une technologie à composant peut requérir une implémentation particulière et inclure des services spécifiques tels que la communication inter composants.

Sélection de composants.

Pour réussir cette étape, il faut un nombre significatif de candidats dans les référentiels ainsi que des outils pour les identifier et les évaluer. La sélection est effectuée en se basant soit uniquement sur les caractéristiques fonctionnelles, soit sur les caractéristiques fonctionnelles et non fonctionnelles.

Adaptation de composants.

Certains composants peuvent être intégrés directement dans le système, d'autres ont besoin d'adaptation, soit par un processus de paramétrage ou par ajout du code qui permet une meilleure composition. Dans certains cas, il n'est pas possible de réutiliser le composant lui-même, mais seulement son interface qui doit être implémentée à nouveau.

Test de composants.

Il s'agit d'associer les besoins du composant avec les besoins du système et vérifier les propriétés du système à partir des caractéristiques fonctionnelles et/ou non fonctionnelles des composants.

1.8.2 Design for reuse :

Ce cycle est similaire au cycle « Design by reuse » : les besoins doivent être analysés et définis, et le composant doit être conçu, implémenté, vérifié et livré. Lors de la construction d'un nouveau composant, les développeurs peuvent réutiliser d'autres composants et exploiter les procédures d'évaluation semblables à celles du développement par réutilisation [3].
Cependant, un composant construit pour être réutilisé doit être générique et flexible. Ainsi, la tâche du développement se complexifie et doit répondre aux besoins suivants : Plus d'effort dans la conception et l'implémentation du composant. Plus d'effort dans le test du composant (le test s'effectue en isolation ou dans différentes configurations) [3].
Plus de qualification de la part des développeurs.
Plus de documentation pour la spécification du composant.
Une fois le composant testé, spécifié, stocké dans un référentiel de composants, la prochaine étape du cycle de vie du composant est la phase de déploiement dans le système. Ce déploiement doit s'effectuer de manière automatique et sans effort d'adaptation. Notons qu'une technologie à composant fournit un ensemble de services techniques qui permettent de connecter un composant dans un système [3].

1.9 Modèles de composant logiciel :

Les auteurs du livre « Component-Based Software Engineering Putting the Pièces Together », ont donné la définition suivante (Heineman et Councill, 2001) : « Un composant est un élément logiciel conforme à un modèle appelé modèle de composants qui définit des standards de composition et d'interactions ».
Cette définition focalise sur l'existence de plusieurs modèles de FRcomposants et que chaque modèle définit des règles pour l'assemblage de composants. Cependant, la réalité révèle que les modèles de composants les plus industrialisés sont des modèles plats qui ne proposent qu'une unique dimension de configuration pour une application.

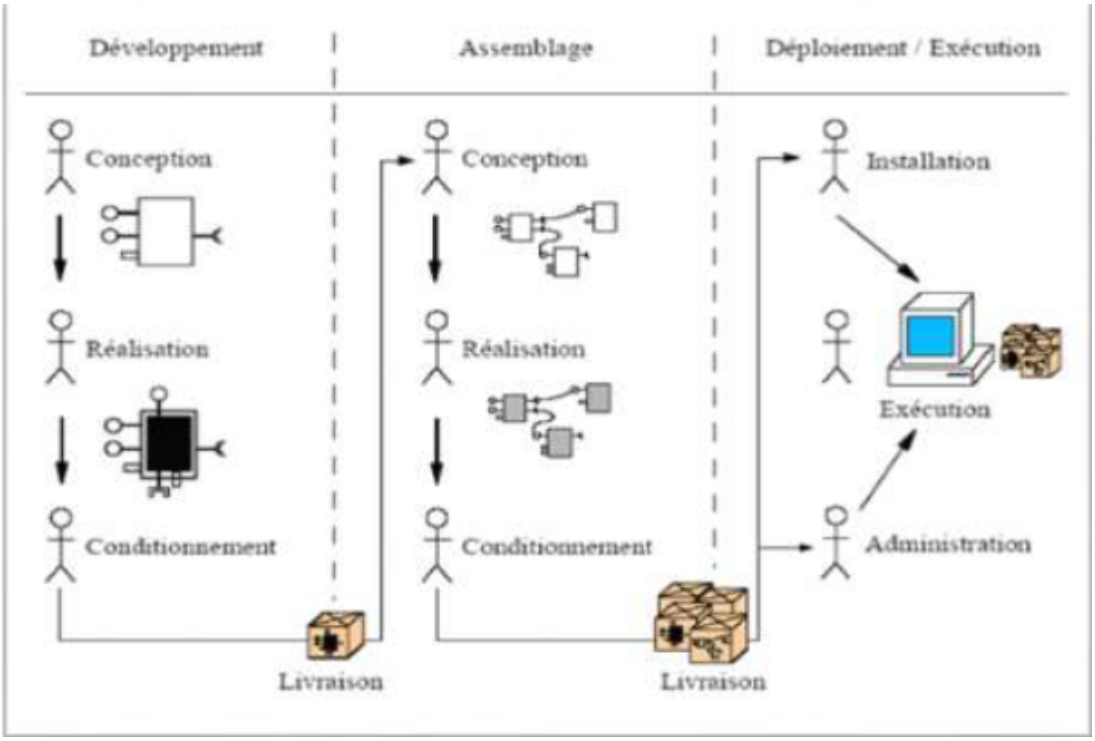


FIGURE 1.2 – Développement à base de composants [25]

Nous présentons dans ce qui suit quelques modèles de composants appartenant à deux catégories : les modèles plats et les modèles hiérarchiques [3].

1.9.1 Modèles plats :

La plupart des modèles industriels appartiennent à cette catégorie (CCM, EJB, NET). La composition des composants (boites noires) s'effectue sur un unique niveau. Nous présentons, dans ce qui suit, un exemple de ces modèles [3].

1.9.2 Modèle CCM :

Le modèle abstrait de composant permet de définir la structure intrinsèque du composant le modèle abstrait de composant décrit les ports et les interface de la home à l'aide du langage IDL actualisé par rapport au CCM [3].

Un composant CCM est constitué de [3] :

— Une référence de base.

— Des interfaces : 4 types de port (la facette, le réceptacle, la source et le puits

— d'événements).

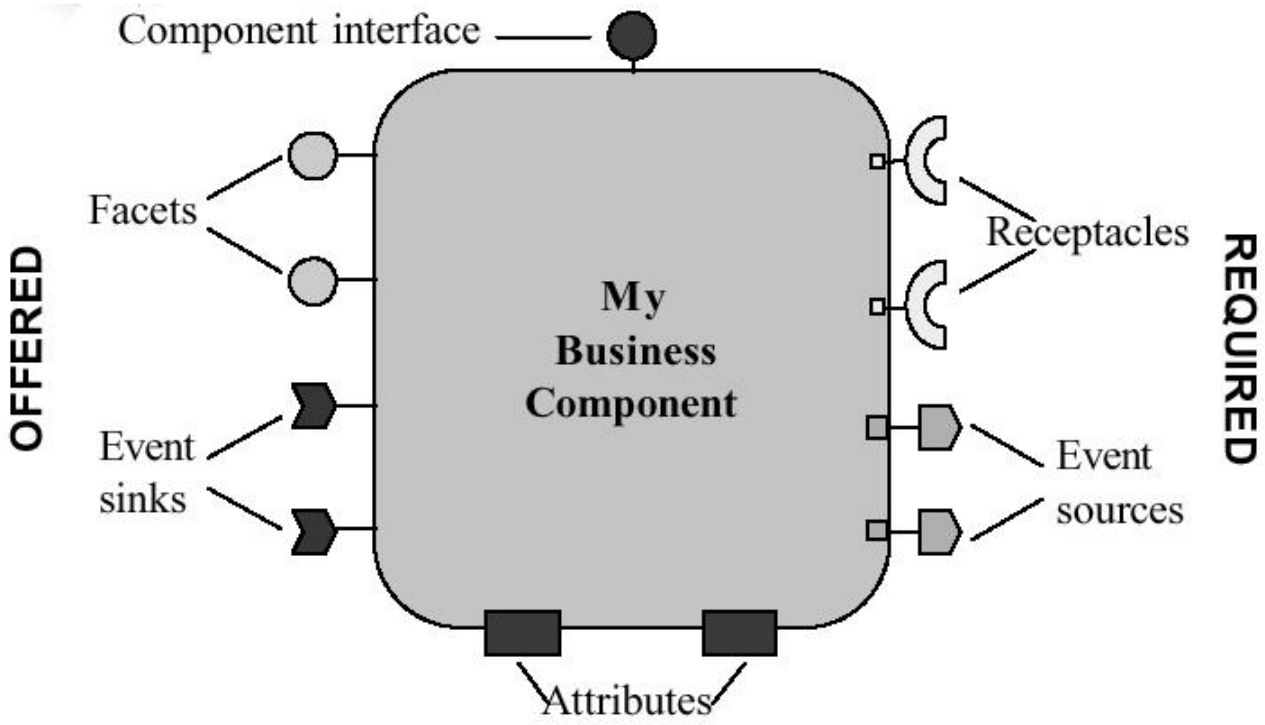


FIGURE 1.3 – Modèle de composant CCM .[16]

— Des attributs.

La facette

c 'est une interface fournie par le composant elle donne une vue sur le composant Il peut y avoir plusieurs facettes [16]

Le réceptacle

cette une interface est un point d'entrée nommé Il exprime la capacité du composant à pouvoir exploiter des références d'objets typées qui seraient fournies par l 'extérieur lien dynamique entre composants [16].

La source et le puit d événements

La source d événements elle diffuse un événement un type d événement donné vers : soit un canal d événement soit des consommateurs le puit d événements il reçoit des événements d 'un type donné, après s 'être abonné [16] .

Fabrique

la fabrique : home est une nouvelle notion introduite dans le méta-modèle de corba. C'est l'entité qui gère le composant elle permet de contrôler le comportement du composant en fonction de son cycle de vie, elle permet la gestion des instances du composant (création, gestion de clé , recherche, exploration) [16].

1.9.3 Modèles hiérarchiques :

Un composant composite est vu comme étant un graphe hiérarchique d'autres composants (appelés sous-composants) avec des composants primitifs comme feuilles. Contrairement au composant composite, un composant primitif contient le code métier. Les sous-composants ne peuvent être reliés que dans un même niveau d'imbrication et leurs interfaces peuvent être déléguées respectivement à leurs composants composites. Les modèles hiérarchiques permettent la création de composants composites (appelés boites grises). Nous présentons dans ce qui suit un exemple de ces modèles [3].

Modèle Fractal :

Le modèle de composants Fractal est un modèle général dédie a la construction, au déploiement et a l'administration (ex. observation, contrôle, reconfiguration dynamique) de systèmes logiciels complexes, tels les intergiciels ou les systèmes d'exploitation. Cela motive les principales caractéristiques du modèle [15]

composants composites :

(i.e. composants qui contiennent des sous-composants) pour permettre d'avoir une vue uniforme des applications à différents niveaux d'abstraction.

composants partagés :

(i.e. sous-composants de plusieurs composites englobants) pour permettre de modéliser les ressources et leur partage, tout en préservant l'encapsulation des composants.

capacités d'introspection :

permettre d'observer l'exécution d'un système.

capacités de (re)configuration :

pour permettre de déployer et de configurer dynamiquement un système.

Par ailleurs, Fractal est un modèle extensible du fait qu'il permet au développeur de personnaliser les capacités de contrôle de chacun des composants de l'application. Il est ainsi possible d'obtenir un continuum dans les capacités réflexives d'un composant allant de l'absence totale de contrôle a des capacités élaborées d'introspection et d'intercession (ex. accès et manipulation du contenu d'un composant, contrôle de son cycle de vie).

Composants et liaisons :

Un composant Fractal est une entité d'exécution qui possède une ou plusieurs interfaces. Une interface est un point d'accès au composant. Une interface implante un type d'interface qui spécifie les opérations supportées par l'interface. Il existe deux catégories d'interfaces : les interfaces serveurs — qui correspondent aux services fournis par le composant —, et les interfaces clients qui correspondent aux services requis par le composant.

Un composant Fractal est généralement compose de deux parties : une membrane — qui possède des interfaces fonctionnelles et des interfaces permettant l'introspection et la configuration (dynamique) du composant —, et un contenu qui est constitue d'un ensemble ni de sous-composants.
Les interfaces d'une membrane sont soit externes, soit internes. Les interfaces externes sont accessibles de l'extérieur du composant, alors que les interfaces internes sont accessibles par les sous-composants du composant. La membrane d'un composant est constituée d'un ensemble de contrôleurs. Les contrôleurs peuvent être considères comme des méta-objets. Chaque contrôleur a un rôle particulier : par exemple, certains contrôleurs sont charges de fournir une représentation causalement connectée de la structure d'un composant (en termes de sous-composants). D'autres contrôleurs permettent de contrôler le comportement d'un composant et/ou de ses sous-composants. Un contrôleur peut, par exemple, permettre de suspendre/reprendre l'exécution d'un composant. Les contrôleurs peuvent également jouer le rôle d'intercepteurs. Les intercepteurs permettent d'intercepter les appels de méthodes entrant et sortant des interfaces d'un composant. Un exemple classique d'interception est l'ajout de pré- et post-traitements a l'appel.
Le modèle Fractal fournit deux mécanismes permettant de de nir l'architecture d'une application : l'imbrication (a l'aide des composants composites) et la liaison. La liaison est

ce qui permet aux composants Fractal de communiquer. Fractal de nit deux types de liaisons : primitive et composite. Les liaisons primitives sont établies entre une interface client et une interface serveur de deux composants résidant dans le même espace d'adressage. Par exemple, une liaison primitive dans le langage C (resp. Java) est implantée a l'aide d'un pointeur (resp. référence). Les liaisons composites sont des chemins de communication arbitrairement complexes entre deux interfaces de composants. Les liaisons composites sont constituées d'un ensemble de composants de liaison (ex. stub, Skelton) relies par des liaisons primitives.

Une caractéristique originale du modèle Fractal est qu'il permet de construire des composants partages. Un composant partage est un composant qui est inclus dans plusieurs composites. De façon paradoxale, les composants partages sont utiles pour préserver l'encapsulation. En e et, il n'est pas nécessaire a un composant de bas niveau d'exporter une interface au niveau du composite qui l'encapsule pour accéder a une interface d'un composant partage. De fait, les composants partages sont particulièrement adaptes a la modélisation des ressources [15].

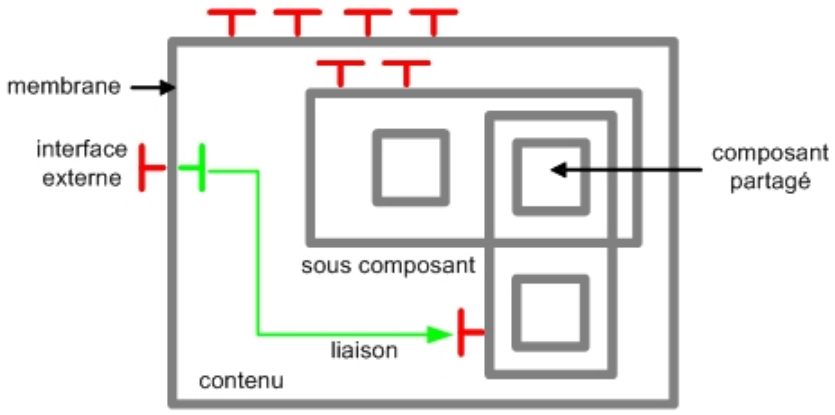


FIGURE 1.4 – Exemple de composant Fractal .[15]

La figure au-dessus représente un exemple de composant Fractal. Les composants sont représentes par des rectangles. Le tour gris du carre correspond à la membrane du composant. L'intérieur du carre correspond au contenu du composant du composant. Les interfaces sont représentées par des (gris clair pour les interfaces clients ; gris fonce pour les interfaces serveurs). Notons que les interfaces internes permettent a un composite de contrôler l'exposition de ses interfaces externes à ses sous-composants. Les interfaces externes apparaissant au sommet des composants sont les interfaces de contrôle du composant. Les flèches

représentent les liaisons entre composants. Enfin, nous avons représente un composant partage entre deux composites [15].

1.10 Conclusion

Une approche par composants ne peut pas être entièrement utilisée si les processus de développement et même les organisations de développement ne sont pas adoptés selon les principes de base de CBSE. Depuis cela approche vise à augmenter la réutilisabilité de composants existants, les efforts pour les implémentations diminuent, et les efforts de la vérification du système augmente. Cela nécessite ajustements des processus de développement. Par une De cas industrielle, nous avons souligné sortir les difficultés pour atteindre une complète séparation des processus de développement de systèmes à partir des composants, ainsi que besoin d'une organisation de projet qui met un rôle plus important sur les questions architecturales, et la vérification du système et des composants.

CHAPITRE 2

Les approches de recherche d'un composant logiciel.

2.1 Introduction

Le génie logiciel orienté composant met l'accent sur la réutilisation de composants logiciels appelés aussi « composants sur étagère » (ou COTS, pour Components Off The Shelf en anglais).
Les composants sur étagère sont des composants logiciels développés par des tiers et stockés dans des référentiels (repositories) pour être utilisés ultérieurement dans la construction d'applications. A cet effet, il existe un besoin grandissant pour la sélection de composants pertinents répondant aux exigences des développeurs.
La recherche de composants en bibliothèque (Component Search and Retrieval) peut être Considérée comme un cas particulier de recherche d'information. On appelle bibliothèque ou dépôt de composants (component library ou repository) une collection de composants, organisée d'une telle manière qu'on peut retrouver certains de ces composants à partir d'une requête particulière. Une requête (query) est une expression contenant des termes. Ces termes décrivent les besoins de l'utilisateur. Un terme peut être un mot, un diagramme, une signature, etc.

2.2 Technique de recherche composant logiciel :

La recherche de composants logiciels est un problème non trivial. Après avoir déterminé les critères d'évaluation, il faut être capable de retrouver le composant recherché, s'il existe ,parmi un ensemble grand de composants logiciels. Dans le cas échéant, il faut trouver le composant le plus proche de celui recherché. Dans la littérature, il existe plusieurs techniques de recherche d'information dont le but est d'avoir un résultat performant [3].

2.2.1 Techniques classiques de recherche d'information :

Un composant peut être considéré comme un type particulier de documents. Ainsi, plusieurs recherches ont tenté d'appliquer les techniques classiques de recherche d'information pour rechercher des composants [3].
Dans (Frakes and Nejmeh, 1987)[3], on applique la technique de recherche d'information textuelle sur un référentiel d'artefacts logiciels (des codes sources en langage C). Les artefacts sont indexés par des termes extraits des entêtes et des commentaires se trouvant dans les codes sources. L'opération d'indexation est entièrement automatisée pour garantir une uniformité des termes d'indexation. Cette approche d'indexation dépend fortement

des habitudes individuelles des programmeurs pour commenter le code source. De plus, la méthode n'utilise pas un vocabulaire contrôlé, ce qui demande un effort supplémentaire des programmeurs pour trouver les bons termes et des utilisateurs du système pour trouver ces mêmes termes.
Les techniques textuelles de recherche d'information sont appliquées sur les codes sources dans le domaine de la réingénierie des systèmes d'information. Elles aident l'ingénieur à comprendre la structure du code source et lui permettent de retrouver des constructions spécifiques [3].

2.2.2 Techniques de classification externe :

Le principe de ces techniques est d'indexer les composants à partir de leurs représentations.

Approche par mots-clés :

C'est la technique de classification externe la plus simple. Le principe est d'associer un ensemble de mots clés avec (ou sans) poids à un composant [25]. La recherche de composants par mots-clés a montré rapidement ses limites pour différentes raisons : lorsque la taille de la bibliothèque : quand elle est élevée, on risque donc de retenir trop de composants avec les mêmes mots-clés. La deuxième limite concerne le domaine des composants.
En effet, un même mot-clé peut être utilisé dans deux domaines différents avec deux significations différentes [11]

Approche par langage naturel :

Dans [12] Maarek, Berry et Kaiser ont proposé une approche basée sur l'application de techniques textuelles de recherche d'information sur des descriptions des composants en langage naturel. L'interrogation de la base de composants se fait à travers des requêtes en langage naturel. Chaque description textuelle d'un composant est analysée pour l'extraction d'un ensemble de termes d'indexation. Ces termes constituent le descripteur qui sera utilisé pour la correspondance avec les requêtes utilisateurs [25].
L'indexation est basée sur une analyse lexicale, syntaxique et sémantique des descriptions des composants. Cela revient à supposer que la description en langage naturel décrit avec précision le composant, d'où la faiblesse de cette approche. Le mécanisme d'interprétation automatique utilisé pour l'analyse des composants utilise des techniques linguistiques pour rechercher les descriptions qui correspondent le mieux aux besoins exprimés par la requête.

Le mécanisme d'interprétation représente donc une deuxième source de difficultés. Cette méthode peut donner de bons résultats, mais elle reste difficile à mettre en oeuvre [7].

Approche par facettes :

Une facette représente une information particulière qui permet d'identifier et de caractériser un composant. Elle est définie par son nom et son vocabulaire, c'est-à-dire l'ensemble des mots-clés qui permettent de la décrire. Pour décrire un composant, un ou plusieurs mots-clés doivent être choisis dans le vocabulaire de chaque facette [11] L'approche de classification par facettes a été utilisée dans plusieurs systèmes de recherche de composants comme [19] [20]. Cette technique a réduit les difficultés soulevées précédemment et donne de bons résultats à condition de bien choisir les facettes, de bien indexer les composants en donnant les bons termes et de bien définir l'espace des termes. La classification par facettes nécessite une indexation manuelle souvent coûteuse en temps [7]. Le tableau de la figure suivant représente les différentes facettes retenues d'un système de recherche [6].

vue	Facette
Vue générale	• Nom du composant • Rôle du composant • Visibilité • Licence • Date de publication • Editeur • Taille • Version • Domaine • Info fournisseur
Vue technologie de développement	• Langage de programmation • Model

FIGURE 2.1 – caractérisation des composants par facettes [6]

2.2.3 Techniques de recherche structurelle

les techniques de la classification externe se basent sur les descriptions de composants (documentation). Alors que les techniques de recherche structurelle concernent l'aspect structurel des composants.

Cette catégorie se divise en deux sous-catégories : les techniques d'appariement (ou matching en anglais) de signatures et les techniques d'appariement de spécifications. Les approches d'appariement de signatures sont en réalité un sous-ensemble des approches d'appariement de spécifications. En plus de l'aspect signature, une approche par spécification s'intéresse à la transformation effectuée sur les données et à la conception interne du composant. Nous faisons cette distinction entre les deux catégories car les approches de classification par signatures sont exploitables avec les composants de type boîte noire, boîte grise et boîte blanche. Les approches de classification par spécification ne sont par contre applicables qu'avec les composants de type boîte blanche et boîte verre [3]

Appariement de signatures :

La signature d'un composant est l'union des signatures de toutes les interfaces qu'il définit. De même, la signature d'une interface, c'est l'union des signatures des opérations qu'elle déclare. Ces techniques, considèrent un composant comme un prestataire de services par le biais de ses opérations. Par conséquent, les requêtes se font sur les signatures de ces opérations [25]. Par exemple dans [26] Zaremski et Wing proposent un processus matching de signature basée sur la définition de type suivante :
Un type est soit une variable de type TypeVar, soit un opérateur de type TypeOp. Cet opérateur peut être, soit un opérateur prédéfini (BuiltInOp) soit un opérateur défini par l'utilisateur (UserOp). Partant de cette définition, l'égalité de type (=T) , est définie comme suit : deux types t et t' sont égaux si ce sont des variables de type lexicalement identiques, où si ce sont des opérateurs de type qui ont les mêmes paramètres et le même résultat [25].
A partir de cette définition, Zaremski et Wing [26] ont défini les règles de matching suivantes ; le matching exact, le matching généralisé, le matching spécialisé et le matching réordonné. Ces règles s'appliquent sur deux signatures tl (est le type d'une fonction appartenant à un composant dans une bibliothèque) et tq (un type entré comme requête) [25]

1. tl « matche exactement » tq : s'ils sont égaux au renommage de variables prés. Par exemple tl (x,x) → bool et tq(y,y)→ bool . Alors tl match exactement tq . [25] Avec une substitution V[y/x] on bien V.tl=T tq.
2. tl « généralise » tq : si on peut retrouver tq à partir de tl via une séquence de substitutions de variables.[25]
3. Dans le matching spécialisé, à l'inverse, c'est tq qui généralise tl .[25]

4. Il y a un matching réordonné quand il existe une permutation des paramètres du type tl telle que celui-ci matche exactement avec tq. Par exemple, (int ;x) → int et (x ; int) → int peuvent matcher, moyennant une permutation de int et de x.[25]

Appariement de spécifications :

Les techniques de recherche de composants par appariement de spécifications tentent de retrouver les composants dont les spécifications correspondent à la spécification de la requête. Ces techniques peuvent être classées selon les types de composants auxquels elles sont applicables : les composants logiciels et les composants conceptuels. Nous nous intéressons dans ce qui suit au premier type de composants. Les techniques de recherche de composants par appariement de spécifications représentent généralement les composants logiciels selon le modèle de la figure suivante [7]

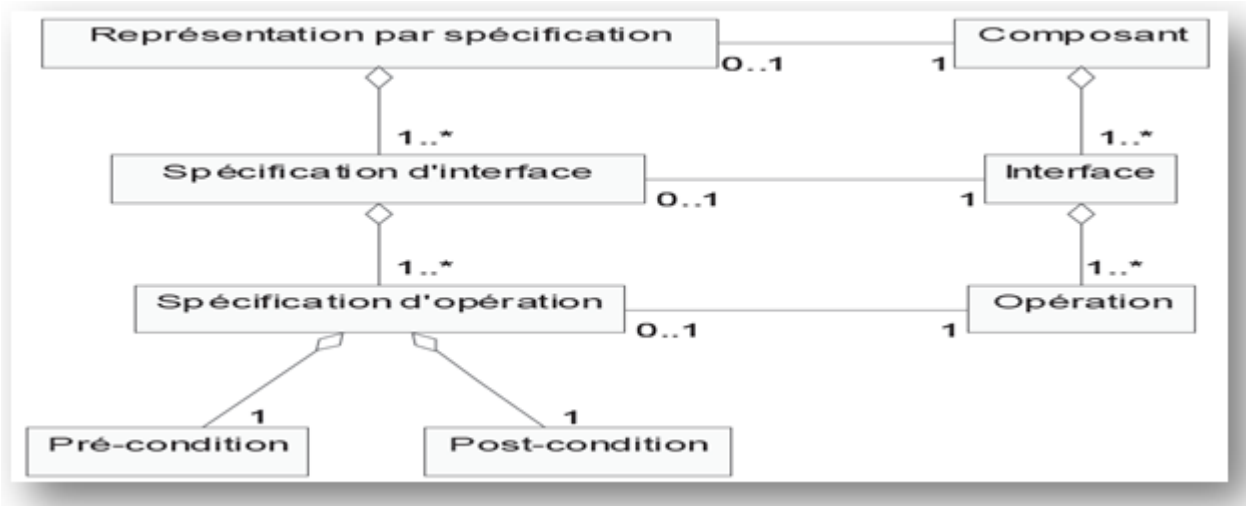


FIGURE 2.2 – Modèle de représentation par spécifications [7]

Zaremski and Wing on a étendu des travaux antérieurs sur les signatures pour proposer une approche d'appariement de spécifications. Les requêtes et les composants sont représentés par un ensemble de paires de pré et post conditions (une paire par méthode). On définit aussi un critère général d'appariement grâce à la formule [25] : En variant les relations R1, R2 et R3, on obtient une hiérarchie de critères d'appariement.[25]

$Match(S,Q) = (Q_{Pré} R_1 S_{Pré}) R_2 (S_{Post} R_3 Q_{Post})$

où $S = (S_{Pré}, S_{Post})$ est la spécification d'un composant,

$Q = (Q_{Pré}, Q_{Post})$ est la spécification d'une requête,

R_1, R_2, R_3 sont trois relations logiquement connectives.

2.2.4 Techniques de recherche comportementale

Les techniques de recherche comportementale s'intéressent à l'aspect dynamique des composants en analysant leur comportement lors de l'exécution.

Approche par analyse des traces d'exécutions :

Parmi les travaux qui se sont intéressés à cette approche, on peut citer ceux de Podgursky et Pierce [22] [23]. Après une observation statistique, les auteurs ont déduit qu'un composant appartenant à une base de composants peut être identifié en se basant uniquement sur son comportement avec des paramètres d'entrée aléatoires. En se basant sur cette observation, ils ont construit une base de composants qui a les propriétés suivantes :

— un composant est représenté par son code exécutable et la description de ses paramètres d'entrée. Un paramètre d'entrée est défini par son type ainsi qu'une distribution probabiliste des valeurs qu'il peut prendre par rapport à son domaine de définition. La distribution probabiliste permet d'estimer la pertinence du choix d'une valeur par rapport à son occurrence lors de l'utilisation du composant.

— la requête se compose de deux parties : l'espace des valeurs d'entrée désirées et une condition qui détermine si la réponse d'un composant est satisfaisante par rapport aux besoins de l'utilisateur.

— l'opération de comparaison de la requête et d'un composant de la base se fait en trois étapes : sélection aléatoire d'un certain nombre de valeurs d'entrée en se basant sur les paramètres de la requête et la description du composant dans la base, application des valeurs d'entrée sur tous les composants de la base, sélection des seuls composants qui vérifient les paramètres de la requête.

Dans [24], Hall critique certains aspects de l'approche de Podgursky et Pierce et propose des améliorations parmi lesquelles :

— la possibilité de définir des paramètres optionnels pour les fonctions.

— la possibilité que l'utilisateur choisisse lui même les données d'entrée par rapport auxquelles les composants sont testés. Hall affirme que le fait de choisir aléatoirement les paramètres d'entrée n'améliore pas forcément la précision et le rappel, mais au contraire affecte fortement les performances du système.
— la sémantique de la fonction de correspondance est modifiée. Au lieu de sélectionner uniquement les composants qui satisfont les besoins de l'utilisateur, la base renvoie aussi les composants qui pourraient être réutilisés par l'utilisateur après une légère adaptation.

Approche par spécification comportementale :

les auteurs [25] présentent un référentiel de composants orientés objet utilisant une technique comportementale de sélection de composants Un composant est représenté dans la base non plus par son code exécutable, mais par des réseaux sémantiques décrivant son comportement. La requête est comparée aux composants pour sélectionner ceux qui maximisent la fonction de similarité qui est égale au rapport de la cardinalité des comportements communs sur la cardinalité totale des comportements de la requête. L'avantage de cette technique est qu'elle permet non seulement de retrouver les composants simples (une classe), mais également les composants complexes (un graphe de classes). De plus, la technique est outillée par :

— un éditeur de spécifications,
— outil de classification de spécifications,
— gestionnaire de bases de composants
— un outil de recherche de spécifications.

2.2.5 Techniques de recherche par navigation

Les techniques de recherche de composants par navigation exploitent les relations implicites ou explicites qui peuvent exister entre les composants appartenant à une base de composants. Par exemple, certains environnements de développement adoptent les techniques de recherche de composants par navigation comme le browser de Smalltalk-80 . Nous présentons dans la suite de cette section deux exemples de sous-catégories des approches de recherche de composants par navigation : approche hypertexte et approche par clustering [7]

Approche hypertexte :

L'approche hypertexte organise les informations en un graphe de noeuds appelés unités d'informations. Ces noeuds sont interconnectés avec plusieurs types de liens appelés relations. L'utilisateur navigue à l'intérieur du graphe en utilisant les relations. La sémantique associée à ces relations guide l'utilisateur selon ses besoins pour sélectionner le bon chemin qu'il doit suivre à travers le graphe.[7]
Le problème majeur de l'approche hypertexte est que le processus de développement et de maintenance de ce genre de bases de composants demande un effort humain considérable. En effet, il n'existe pas une méthode claire et efficace permettant d'extraire et d'identifier les relations qui offrent une richesse sémantique suffisante pour aider au mieux l'utilisateur dans ses choix de navigation. De plus, l'ajout d'un nouveau composant au système peut remettre en cause la consistance des liens déjà existants. Pour remédier à ce problème, des travaux proposent de générer semi-automatiquement des liens entre les composants. En effet, le mécanisme d'extraction des relations varie suivant le modèle de représentation de composants utilisé par la base de composants.
La base de composants SEL représente la collection de composants par un graphe hypertexte de documents reliés par des hyperliens. Les documents décrivent les composants. La recherche se fait à travers un système de requêtes composées sous la forme d'expressions booléennes de termes. SEL utilise un thésaurus pour étendre les requêtes par synonymie. Les documents retrouvés sont classés selon une mesure de pertinence. La pertinence d'un composant par rapport à une requête est fonction de la fréquence d'apparition des termes de la requête dans le document et du nombre de documents appartenant à la base de composants. Une fois la requête évaluée, l'utilisateur part des documents retournés par SEL pour naviguer dans la base.
La technologie de l'hypertexte est utilisée dans les travaux de Isakowitz et Kaufmann pour construire la base de composants ORCA. Elle exploite une technique de classification des composants par facettes, implantée sous la forme d'un système hypertexte.[7] ? ? ? ?

Approche par clustering

Jeng et Cheng construisent un graphe hiérarchique de spécifications permettant la navigation pour la recherche de composants. Le graphe est construit en utilisant un algorithme de clustering et une relation de subsomption. Dans un premier temps, la relation de subsomption est utilisée pour construire des graphes hiérarchiques de spécifications. Puis,

les graphes sont reliés grâce à un algorithme de clustering sur les sommets des graphes hiérarchiques(figure8) . Ainsi lors de la navigation, l'utilisateur commence par choisir le cluster qui l'intéresse, puis continue sa navigation dans la base de spécifications en suivant les liens de subsomption.

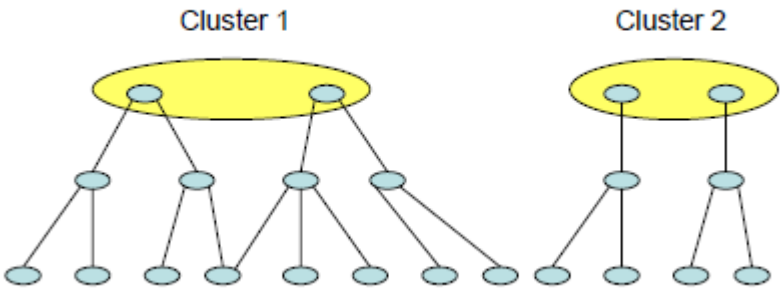


FIGURE 2.3 – de graphes hiérarchiques de spécifications de composant [7]

Les approches de recherche de composants par clustering et hypertexte demandent à l'utilisateur de piloter le processus de navigation, ce qui risque de devenir une tâche difficile si la taille de la base de composants est grande et selon le niveau d'expertise de l'utilisateur. De plus, le résultat final du processus de recherche dépend fortement du nœud de départ dans le graphe de recherche. Pour cela, les approches de recherche de composants par navigation sont souvent utilisées comme un mécanisme complémentaire de raffinement de la recherche des systèmes utilisant les approches linéaires à base de requêtes. D'un autre côté, les approches par navigation demandent un effort important pour la construction des graphes qui sont utilisés lors de la navigation. Les algorithmes de construction des graphes de navigation sont généralement semi-automatiques.

2.2.6 Techniques de recherche sémantique

Les techniques de recherche sémantique s'intéressent à l'aspect sémantique des composants en analysant les axiomes formels qui représentent des informations supplémentaires sur les concepts et leurs relations ainsi que des restrictions relatives aux valeurs de propriétés et de concepts. Par conséquent, les axiomes jouent un rôle important dans le domaine sémantique et peuvent être pertinents pour la recherche de composants [3]

Approche par extension de mots clés

Les techniques de cette catégorie étendent les techniques de recherche par mots-clés. Les mots-clés ajoutés possèdent un lien sémantique avec les mots-clés extraits de la requête. Durant les décennies 60 à 80, les techniques d'expansion de requêtes se basaient sur le critère de co-occurrence des termes d'indexation et n'ont pas eu beaucoup de succès [3].

Approche basée sur les ontologies

Les moteurs de recherche sémantiques reposent sur certaines structures d'ontologies. Les ontologies sont généralement construites à partir de concepts, de propriétés, de contraintes et éventuellement d'axiomes. Les ontologies peuvent servir à calculer la similarité entre la représentation de la requête et la représentation des documents dans le cas où les deux représentations sont construites à partir des concepts d'une même ontologie.. Cette ontologie est un graphe orienté contenant un ensemble de concepts reliés par des relations dont la relation de subsomption. L'avantage du calcul de la similarité est de classer les documents en fonction de leur similarité à la requête, cette similarité dépend de l'organisation des concepts dans l'ontologie.[3]

2.3 Conclusion

Après l'étude de ces différentes approches, on peut dire que chaque approche aide l'ingénieur d'application à résoudre un type de requête.par exemple la représentation par facette et classification en utilisant la description en langage naturel qui aidera l'ingénieur de l'application pour sélectionner les composants en utilisant des informations externes fournie par la description humain des composants, ce genre de modèle de récupération de composants fournit des outils adaptés aux utilisateurs non expérimentés. L'approche par signature et les modèles de correspondance des spécifications « matching » permettent à l'ingénieur de sélectionner les composants en spécifiant leur signature ou leur spécification. Ils sont utiles pour les ingénieurs d'applications qui ont des besoins spécifiques et sont capables de spécifier formellement leurs besoins.

CHAPITRE 3

langage de description d'architecture

3.1 Introduction

Depuis le début des années 90, la communauté de l'ingénierie du logiciel a développé des techniques centrées sur la description de l'architecture des systèmes . Celles ci ont été élaborées à de multiples fins : modélisation de la structure et du comportement améliorer la compréhension et la conception des systèmes complexes, adapté aux projets de grande taille ,favoriser leur évolution et leur réutilisation, procéder à diverses analyses, faciliter la construction et le déploiement, ou encore aider à la gestion de configuration. Ces techniques se sont concrétisées au travers de langages spécifiques, les ADL's (Architecture Description Langage), et l'outillage associé.
Plusieurs travaux actuels vont dans le sens de l'unification ou de la standardisation de ce type de langage pour les rendre plus utilisables. L'aboutissement de ces recherches permettrait de palier les difficultés de cohérence et de compréhension des modèles dues à la multiplicité et à la spécificité des langages, ou encore de surmonter les problèmes d'interopérabilité issus de la variété des formats. Un voie s'est ouverte avec le travail effectué à UCI (University of California, Irvine) qui a élaboré un ADL modulaire et hiérarchisé, xADL2.0, isolant ainsi le fondement générique des ADL's, et laissant la voie ouverte vers la spécialisation par l'extension. Nous proposons dans cet article d'étendre xADL2.0 à un domaine qu'il ne couvre pas encore, l'expression du comportement, pour en faire un langage d'analyse.

3.2 Définition d'un ADL :

Les ADL's sont une famille de langages élaborés dans les années 90 dans le domaine de l'ingénierie du logiciel. Ils ont en commun une capacité à décrire l'architecture des systèmes mais diffèrent largement quant à leurs motivations et leurs formalismes d'écriture. Ils sont en général graphiques et textuels et possèdent des outils associés [14]

3.3 Concepts de base des ADL :

3.3.1 Connecteur

Un connecteur est un bloc de construction utilisé pour exprimer les interactions entre composants ainsi que les règles qui gouvernent cette interaction. Les connecteurs sont des entités architecturales qui relient des ensembles de composants et agissent en tant que

médiateurs entre eux. Les exemples de connecteurs incluent des formes simples d'interaction, comme des pipes, des appels de procédure et l'émission d'évènements. Les connecteurs peuvent également représenter des interactions complexes, comme un protocole client/serveur ou un lien SQL entre une base de données et une application. Contrairement aux composants, les connecteurs peuvent ne pas correspondre à des unités de compilation.
Cependant, les spécifications de connecteurs dans un ADL peuvent également contenir des règles pour implémenter un type spécifique d'un connecteur [2].

3.3.2 Configuration :

Les configurations architecturales représentent les graphes de composants et de connecteurs et la façon dont ils sont reliés entre eux. Cette notion est nécessaire pour déterminer si les composants sont bien reliés, si leurs interfaces s'accordent, si les connecteurs correspondants permettent une communication correcte. La combinaison de leurs sémantiques aboutit au comportement désiré, qui vient en appui des modèles de composants et de connecteurs. Les descriptions des configurations permettent l'évaluation des aspects distribués et concurrents d'une architecture, comme par exemple, la possibilité de déterminer des verrous, de connaître le potentiel de performance, de fiabilité, de sécurité...etc. Le rôle clé des configurations est de faciliter la communication entre les différents intervenants dans le développement d'un système. Leur but est d'abstraire les détails des différents composants et connecteurs. Ainsi, elles décrivent le système à un haut niveau d'abstraction qui peut être potentiellement compris par des personnes avec différents niveaux d'expertise et de connaissances techniques [17].

3.3.3 La composition/assemblage

permet de construire des applications complexes à partir de composants simples. La composition ou assemblage permet de lier un composant demandant des services à d'autres composants offrant ces dits services [17].

3.4 Familles des ADLs

Les langages de description d'architecture sont des langages dits déclaratifs. Ils peuvent être classés en deux grandes familles. La première correspond aux langages qui privilégient la description des éléments de l'architecture et leur assemblage structurel, la seconde définit

les langages qui se centrent sur la description de la configuration d'une architecture et sur la dynamique du système [13]

— première famille : la première famille de langages correspond à une famille de langages qui est accompagnée d'outils de modélisation, d'analyseur syntaxique et de générateur de code [13].

— La deuxième famille : La deuxième famille de langages regroupe des langages accompagnés d'outils de modélisation et de génération de code mais aussi d'une plate-forme d'exécution ou de simulation d'un système, voire de modification dynamique pendant l'exécution. La particularité de ces langages est de définir un élément d'une architecture (composant ou connecteur) comme une instance. Il devient alors facile et simple de spécifier l'évolution dynamique d'une application au cours de son exécution [13].

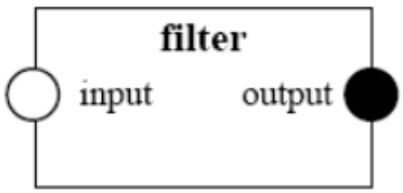
3.5 Quelques ADLs académiques

Un grand nombre de langages de description d'architecture sont récemment nés au sein de la communauté scientifique et apportent chacun leurs spécificités fonctionnelles. les ADLs les plus connus sont : UniCon, Darwin, Wright, Rapide, OLAN, ACME, etc.

3.5.1 Darwin :

Darwin est défini pour supporter l'analyse des transmissions de messages dans les systèmes parallèles distribués. Les principales abstractions gérées par Darwin sont les composants. Un composant type est décrit par une interface composée d'une collection de services qui sont fournis ou requis. Les connecteurs quant à eux ne sont pas considérés comme des entités de première classe. Chaque interaction est représentée par un lien entre un service requis et un service fourni de composants différents. Les configurations (composants composites) sont décrites par les déclarations d'instanciation des composants et par les liaisons entre les services requis et les services fournis de ces instances de composants.[18]
La figure au dessous montre un exemple de composant qui est un filtre qui fournit (provides) un service (output) et requiert (requires) un service (input). Le type de service est spécifié entre ¡¿. Dans cet exemple, le mécanisme de communication est un Stream et le type de données communiquées est char . Un composant peut fournir plusieurs services et en requérir plusieurs. Les noms des services sont locaux. En fait, un composant n'a pas à connaître les noms globaux des services externes ni leur localisation dans le système distribué. Cela per-

met d'implanter et tester un composant indépendamment du reste du systèm e et aussi de le réutiliser et le remplacer facilement [28].




```
component filter{
  provide output<stream char>;
  require  input<stream char>;
}
```

FIGURE 3.1 – exemple d'un composant [28]

3.5.2 Wright :

L'ADL Wright est construit autour de trois principaux concepts architecturaux : les composants ,les connecteurs et les configurations [18].Un composant est décrit en deux parties. Une partie qui contient la déclaration d'un ensemble de ports qui représentent son interface et une partie qui représente la description du comportement du composant appelée calcul (computation). Chaque port représente un point d'interaction entre le composant et son environnement. Une spécification formelle exprimée par le langage CSP est associée à chaque port, elle exprime les propriétés et les attentes du composant vu à travers ce port et elle constitue une partie de son comportement. La spécification du calcul (computation)fournit une description complète du comportement et des propriétés du composant en montrant comment les ports sont regroupés et utilisés. Les analyses faites sur le composant se baseront sur cette spécification. [28]
Dans une architecture Filtre et Canal (pipe and filter), considérons un exem ple où on a deux filtres (Filtre1, Filtre2), le prem ier reçoit un flux de caractères en entrée et le transm et en sortie au second filtre [28].

FIGURE 3.2 – exemple de filtre et canal [28]

Le filtre correspond à un composant qu'on peut décrire simplement avec le langage Wright comme suit :

Component Filtre

Port Entrée *[Lire les données jusqu'à ce que la fin des données soit atteinte]*

Port Sortie *[sortir les données de manière permanente]*

Computation *[Lire continuellement les données à partir du port* Entrée*, puis les envoyer sur le port* Sortie*]*

3.5.3 ACME :

ACME est un langage de description d'architecture qui a pour but principal de fournir un langage pivot, facilitant ainsi les échanges entre différents ADLs.Il fournit une structure générique extensible pour la représentation, la génération et l'analyse de descriptions d'architecture. ACME se base sur sept types d'entités pour décrire une architecture : les composants, les connecteurs, les systèmes, les ports, les rôles, les représentations et les cartes de représentations (représentation map)[17]

- Le composant représente l'unité de traitement d'une application. Il est composé d'une interface constituée de ports.
- Les ports représentent les points d'interaction entre différents composants. Il peut s'agir, par exemple, d'une simple signature d'une méthode ou d'une collection de procédures devant être appelées dans un ordre précis.
- Le connecteur représente l'interaction entre deux composants.Il correspond à un médiateur de communication qui coordonne les interactions entre les composants.
  Les connecteurs peuvent être, par exemple, un protocole d'appel de procedures à distance RPC(Remote Procedure Call)1ouunbusORB(Object Request Broker).Un connecteur est composé d'un ensemble de types de Rôles tels que le rôle client et le rôle serveur.
- Le rôle définit un participant à la connexion. Par exemple, pour un connecteur permettant a transmission de messages, les rôles "sender" et "receiver" peuvent être définis pour indiquer respectivement l'émetteur et le récepteur du message envoyé.
- Le système représente la configuration d'une application, c'est-à-dire l'assemblage structurel entre les composants et les connecteurs.
- Les composants et les connecteurs peuvent être décrits d'un niveau général à un niveau plus détaillé et peuvent donc être raffinés. Chaque nouvelle description d'un élément est appelée une représentation.

— La carte de représentation spécifie la correspondance entre un élément de l'architecture et ses représentations .La représentation et la carte de représentation permettent à ACME de supporter la description hiérarchique d'une architecture.

La figure suivante représente un diagramme correspondant à une architecture client-serveur. Ce système contient deux composants : le client et le serveur. Ces composants sont reliés par le connecteur RPC [28].

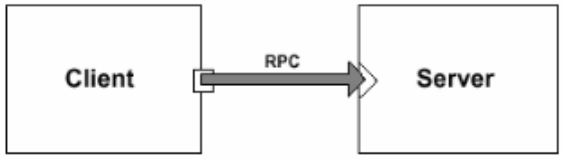


FIGURE 3.3 – Architecture client-serveur [28]

Ce système est décrit avec ACME comme suit [28] :

```
System simple_cs = {
        Component client = { Port send-request }
        Component server = { Port receive-request }
        Connector rpc = { Roles {caller, callee} }
        Attachments : {
                client.send-request to rpc.caller ;
                server.receive-request to rpc.callee }
}
```

3.6 Avantages des architectures logicielles :

La description de l'architecture logicielle s'impose de plus en plus comme une étape indispensable du développement des systèmes à base de composants en permettant au concepteur de raisonner sur les propriétés fonctionnelles et non fonctionnelles (fiabilité, sécurité.) du système à un haut niveau d'abstraction. Il est bien admis aujourd'hui qu'une bonne architecture peut amener à un produit qui répond aux besoins des utilisateurs et qui peut être modifié facilement et qu'une mauvaise architecture peut avoir des conséquences désastreuses sur le système. D'autres avantages ont été reconnus pour les architectures logicielles à base de composants tels que [2] :

— La dissociation des interactions entre les composants de leur implémentation, ainsi que la définition explicite de ces interactions dans la plupart des descriptions architecturales.
— La considération des interfaces comme des entités de première classe et leur description explicite.
— La description d'une architecture globale d'un système peut être spécifiée avant de compléter la construction de ses composants (implémentation).
— La possibilité de définir des représentations hiérarchiques très riches, notamment via la notion de la composition .

donc comme nous avons vu les architectures logicielles constituent un guide pour l'évolution des systèmes logiciels à base de composants. En effet, aborder la problématique de l'évolution au niveau de l'architecture permet au concepteur de raisonner à un haut niveau d'abstraction sur l'ajout, la suppression, la modification des composants et/ou des connecteurs ou la réorganisation de l'architecture, sans pour autant se noyer dans les détails de l'implémentation de ses éléments [2].

Pour asseoir le développement de telles architectures, il est nécessaire de disposer de notations formelles et d'outils d'analyse de spécifications architecturale .Les langages de description d'architectures constituent une bonne réponse. Nous les abordons dans la section suivante [2].

3.7 Conclusion :

L'architecture logicielle occupe une position centrale dans le processus de développement des systèmes complexes. Les systèmes logiciels complexes nécessitent des notations expressives et bien définies pour représenter leurs architectures logicielles. Dans ce chapitre, nous avons présenté les concepts de base des langages de description d'architectures et les principaux Adl.

CHAPITRE 4

Conception et Implémentation

4.1 Introduction

Dans le cycle de vie de notre projet, la conception représente une phase primordiale déterminante pour produire une application efficace. C'est dans ce stade que nous devons clarifier en premier lieu la conception générale que nous allons suivre dans la partie réalisation de notre projet. Puis, dans un deuxième lieu nous allons détailler notre choix conceptuel a travers plusieurs types de diagrammes Choix dune méthode de conception :
il existe plusieurs méthodes de conception, citons comme exemples : UML,OMT,MERISE, autant de méthodes maitrisées par Additeam ,elles permettent de diriger une transformation continue du système d'information en le simplifiant durablement pour des applications clientes indépendantes, des modules ou composants en forte cohérence, mais en couplage faible. Dans notre démarche et pour la conception du projet, nous retenons le langage de modélisations UML.

4.2 Description d'un composant logiciel :

La description des composants concerne l'aspect fonctionne : les développeurs doit donner une description de l'ensemble des services fournis et requis en précisant des mots clés. La structure générale de déscription que nous avons proposés est présente dans la figure suivante

```
<!DOCTYPE Component [
<!ELEMENT name (#PCDATA)>
<!ELEMENT KeyWord (#PCDATA)>
<!ELEMENT Keywords (KeyWord)+>
<!ELEMENT Input (#PCDATA)>
<!ATTLIST Input type CDATA #REQUIRED>
<!ELEMENT Output (#PCDATA)>
<!ATTLIST Output type CDATA #REQUIRED>
<!ELEMENT Interface_Decription ((Input)+,Output)>
<!ELEMENT Component (name,Keywords,Interface_Decription)>
]>
```

FIGURE 4.1 – Description de composants avec DTD

par exemple le composant "connexion.xml" peut etre décrit comme suit :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Component>
<name>connexion</name>
<Keywords>
<KeyWord>login</KeyWord>
</Keywords>
<Interface_Decription>
<Input type="String">nom d'utilisateur</Input>
<Input type="String">mot de pass</Input>
<Output type="String">access compte</Output>
</Interface_Decription>
</Component>
```

FIGURE 4.2 – Un fichier xml «généré»

4.3 Comportement fonctionnel du système développé

La vision globale du comportement fonctionnel du système développé, est présentée dans le diagramme de cas d'utilisation ci-dessous

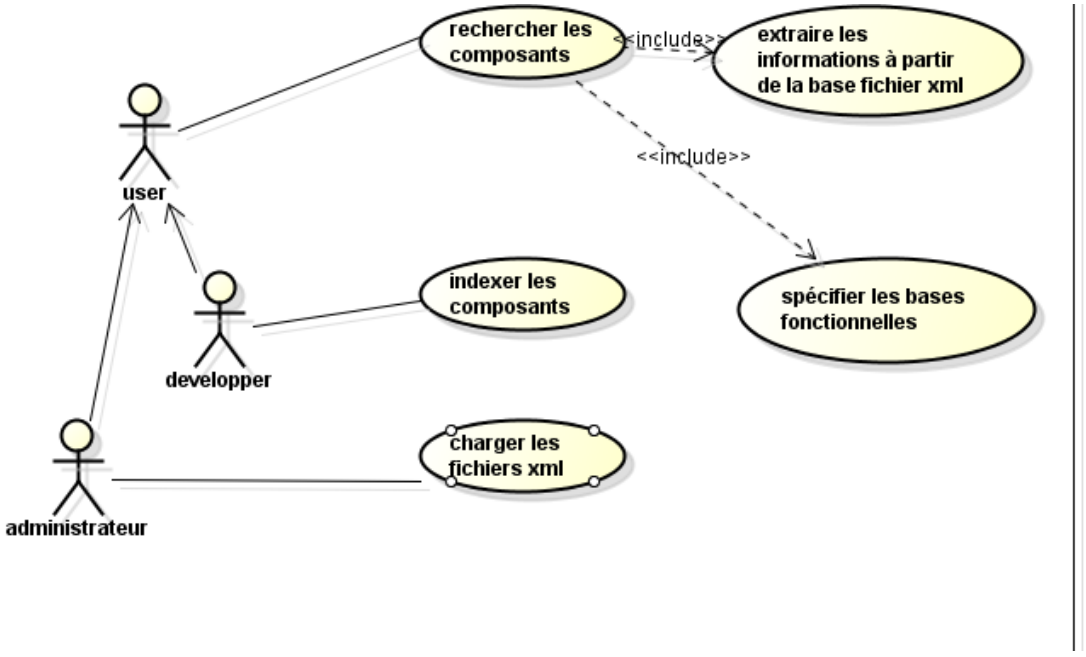


FIGURE 4.3 – diagramme de cas d'utilisation de notre systeme

Comme acteurs on va trouver deux catégories :

— Les administrateurs : à pour rôle charger les fichiers xml
— Les indexeurs : ont pour rôle de mettre en place une structure unifiée pour l'ensemble des fichiers XML(composants)à indexer, et ils peuvent être des utilisateurs.
— Les utilisateurs :Des utilisateurs "clients" cherchent des services, même aussi ils peuvent etre des développeurs cherchent des composants pour l'opération de l'assemblage

Le cas d'utilisation «Indexer» :

L'indexation des fichiers XML dans une base de données bien déterminée soit au préalable par le développeur ou après l'exécution de l'application par l'utilisateur ,ce dernier est charger de respecter notre structure pour le fichier XML .La figure suivante représente l'ordre chronologique des différentes interactions et l'implémentation du cas «Indexer»

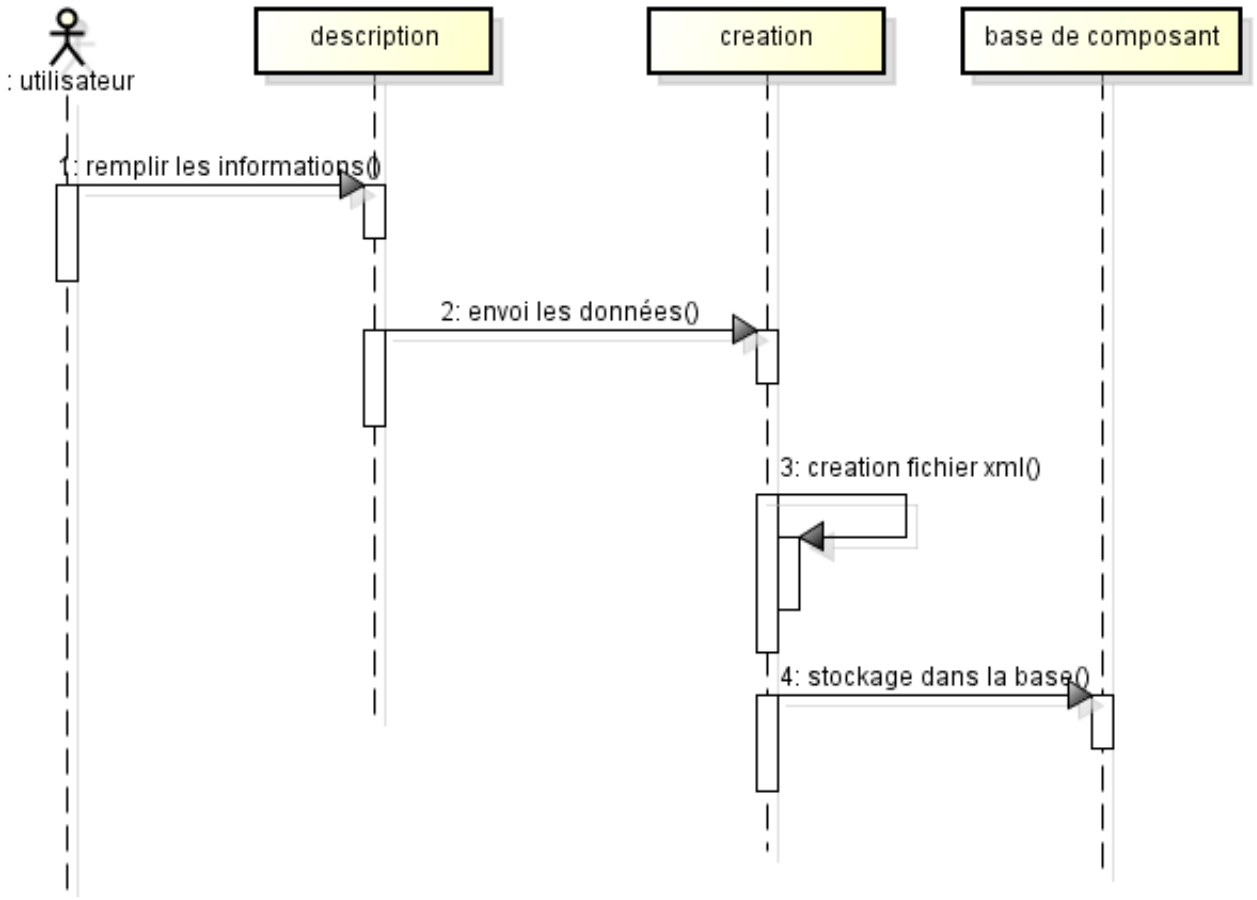


FIGURE 4.4 – Diagramme de séquence « cas indexer des composants »

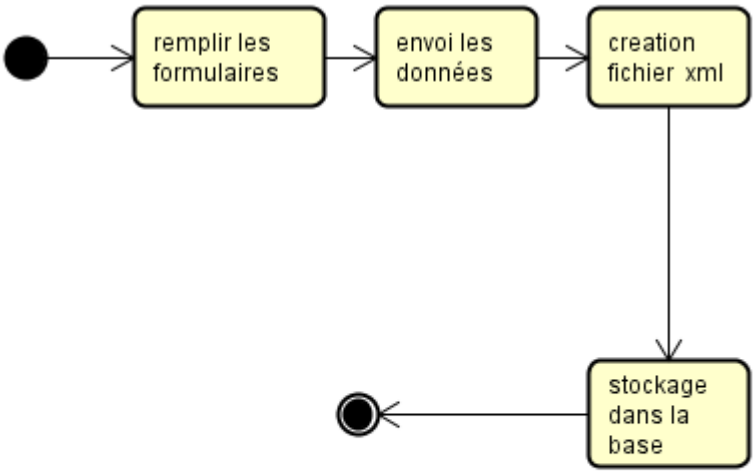


FIGURE 4.5 – Diagramme d'activité « indexer des composants »

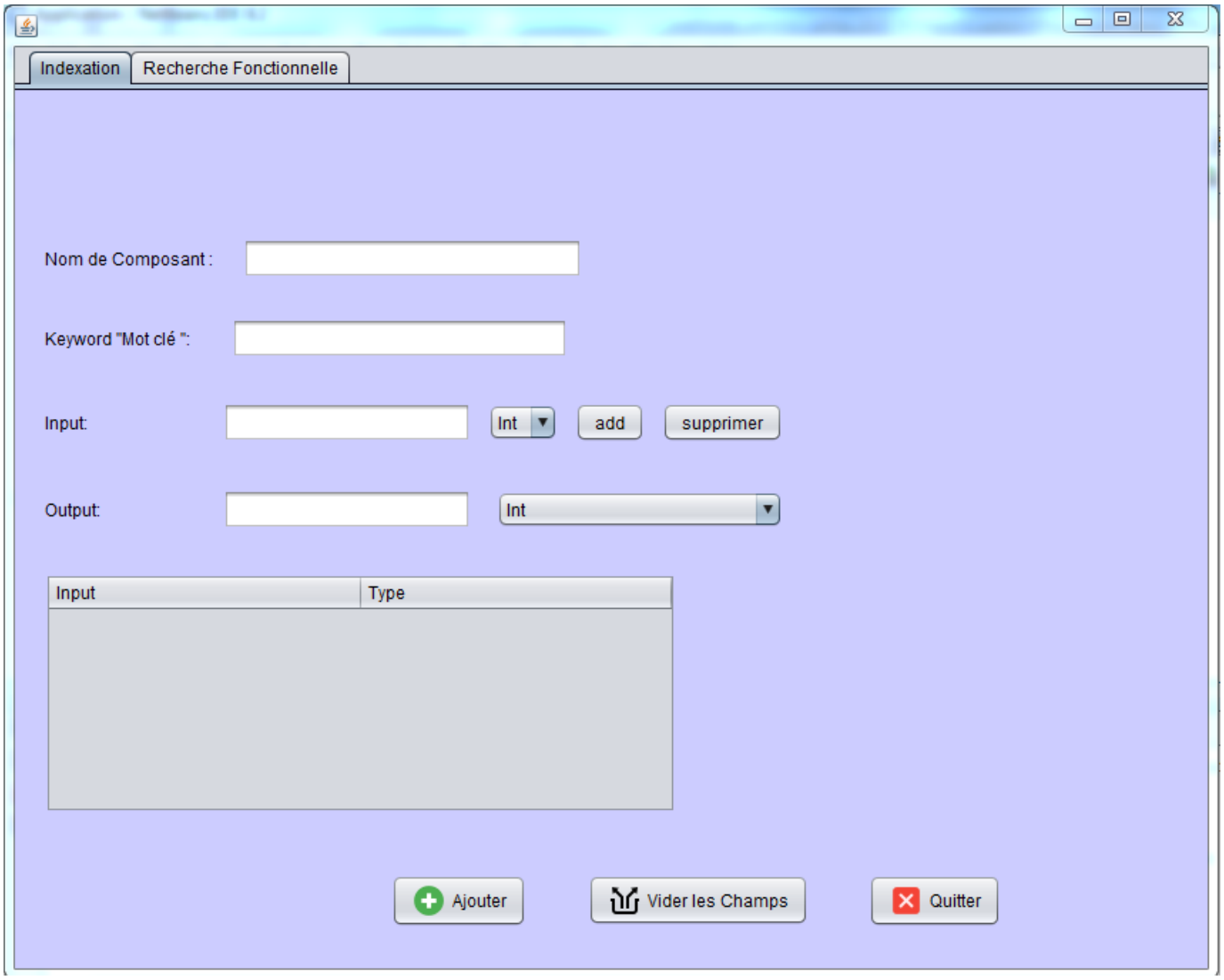


FIGURE 4.6 – Menu «Indexation».Capture d'écran

Le cas d'utilisation «charger» :

La figure suivante représente l'ordre chronologique présente les étapes de charger d'un composant logiciel.

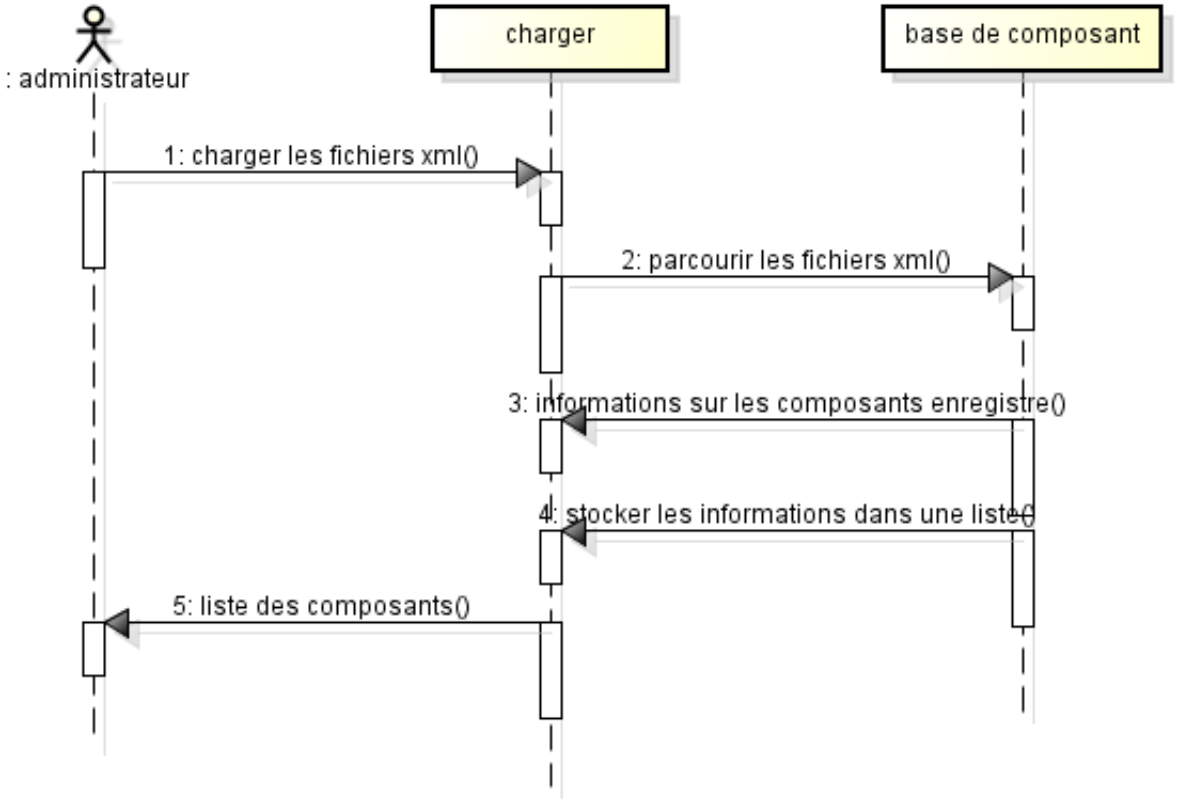


FIGURE 4.7 – Diagramme de séquence «cas charger des composants»

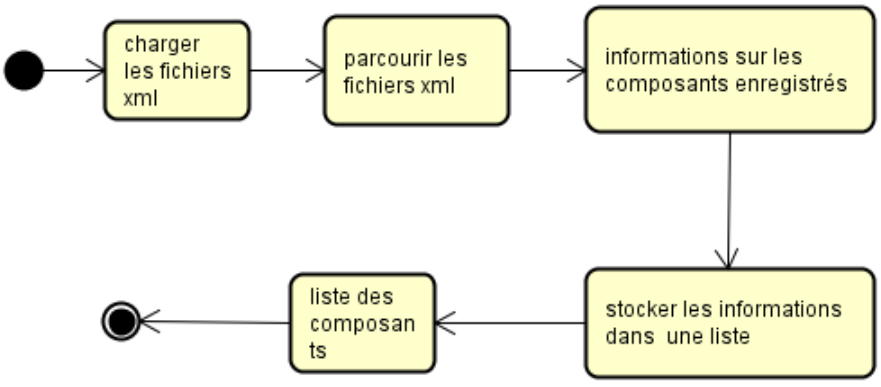


FIGURE 4.8 – Diagramme d'activité «charger des composants»

4.3.1 Le cas d'utilisation «Rechercher»

La figure suivante représente l'ordre chronologique présente les étapes de recherche d'un composant logiciel.

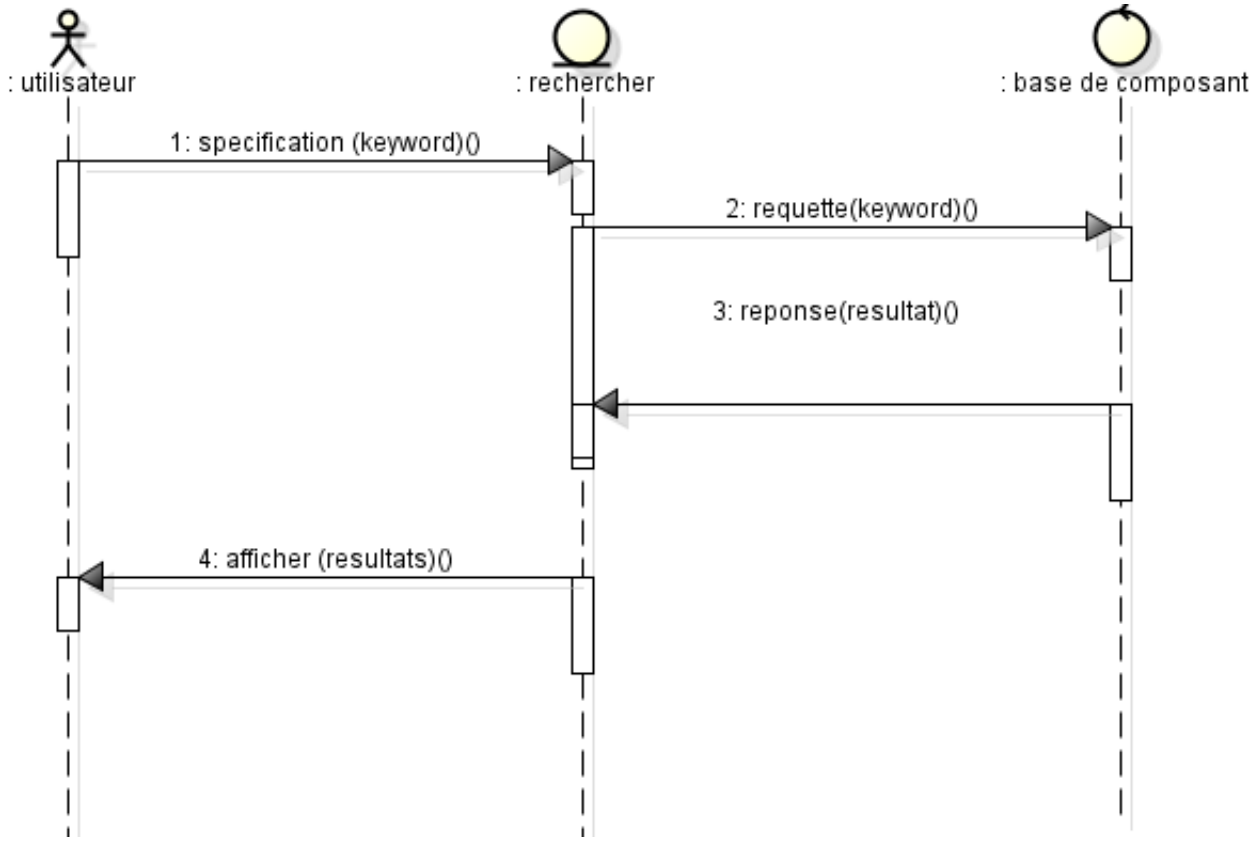


FIGURE 4.9 – Diagramme de séquence « rechercher des composants»

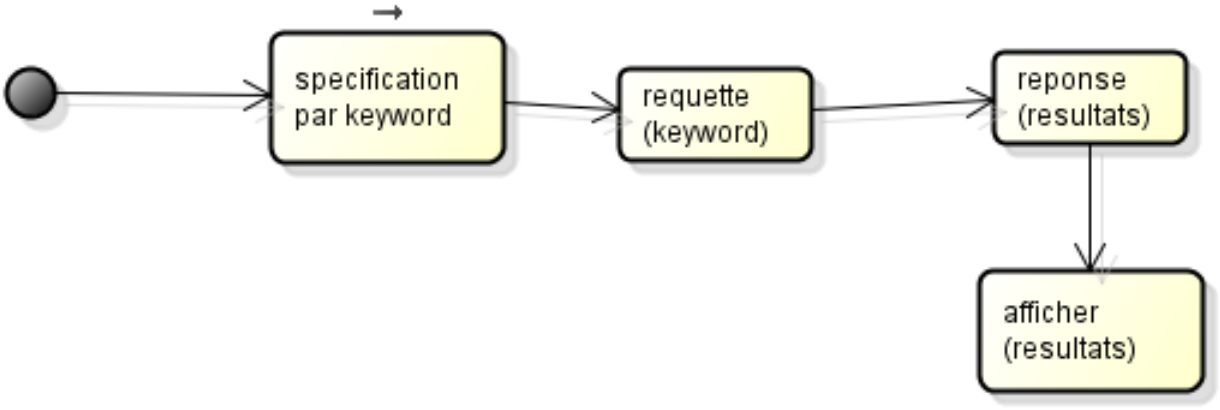


FIGURE 4.10 – Diagramme d'activité «recherche des composants»

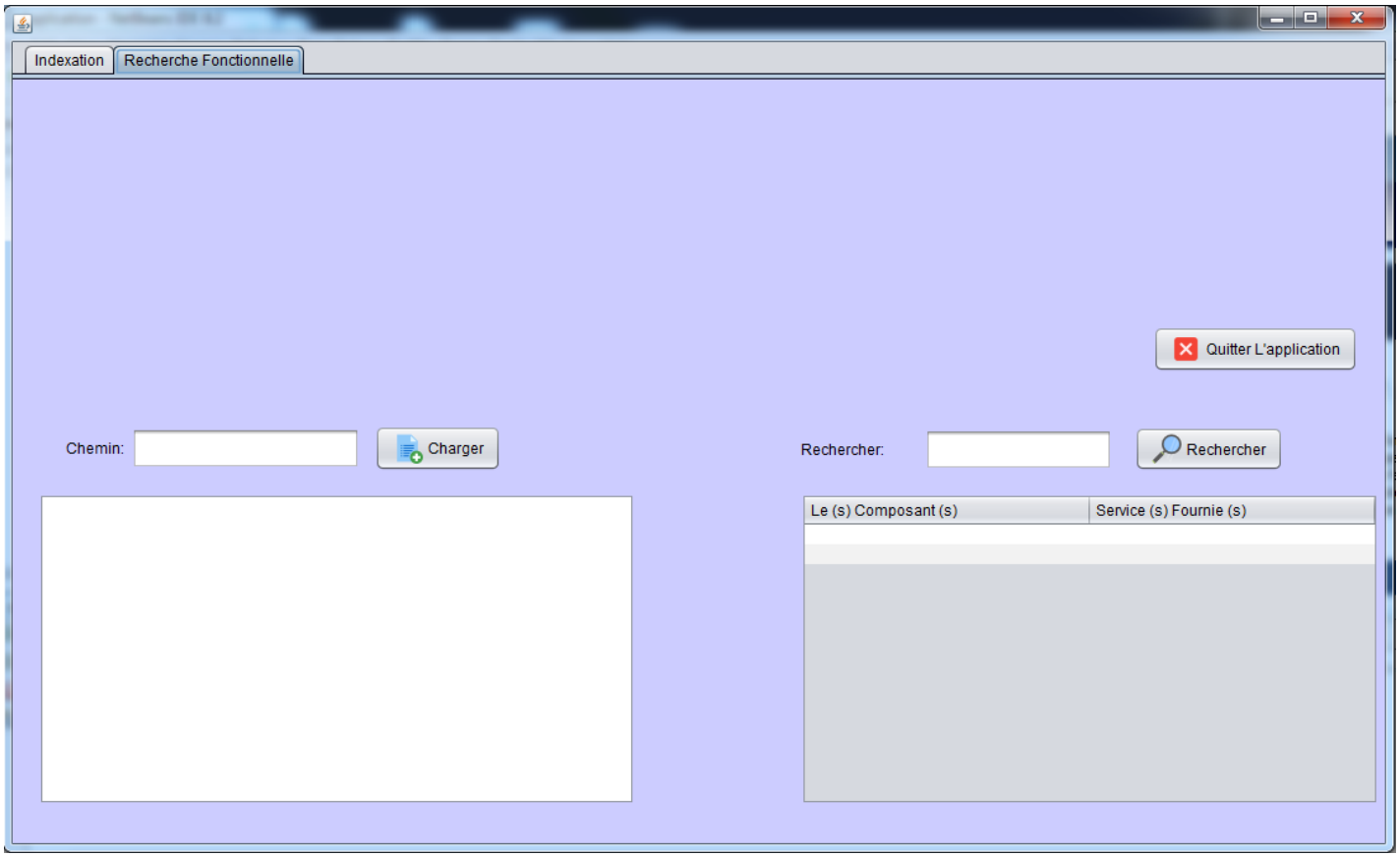


FIGURE 4.11 – Menu «Rechercher».Capture d'écran

4.4 Diagramme de classe :

Le diagramme de classes est généralement considéré comme le plus important dans un développement orienté objet. Sur la branche fonctionnelle, ce diagramme est prévu pour développer la structure des entités manipulées par les utilisateurs. En conception, le diagramme de classes représente la structure d'un code orienté objet, ou au mieux les modules du langage de développement.

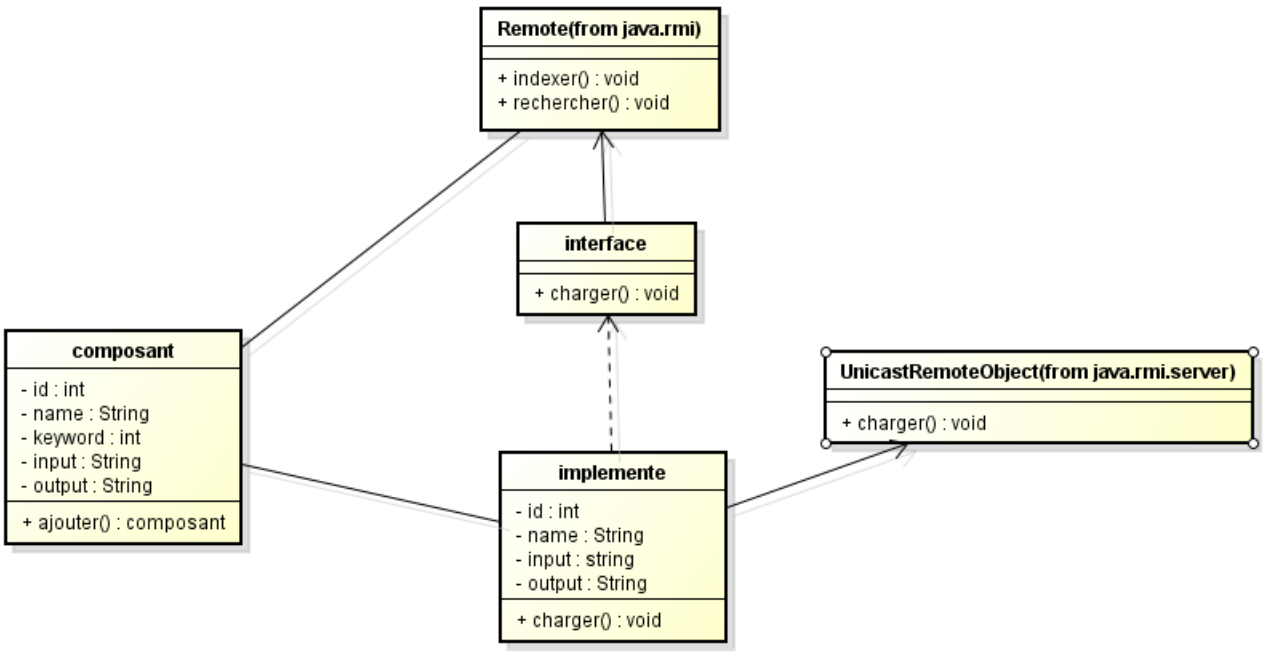


FIGURE 4.12 – Diagramme de classe de notre système

4.5 Conclsion

nous avons choisi de travailler avec UML parce qu'il exprime mieux la vue statique et dynamique du système d'information et pour notre application , il est nécessaire de faire une analyse très approfondie pour pouvoir dégager les nécessités de développement ainsi que quelques scénarios d'exécution.

Conclusion générale

Au terme de ce mémoire, nous procédons dans les lignes qui suivent à un récapitulatif du travail effectué. Rappelons que notre travail consistait à développer un système qui permet la recherche par signature d'un composant logiciel. La première partie de ce mémoire est réservée aux travaux component based software engineering . Nous nous sommes attelés, dans un premier temps, à définir les différents concepts liés aux composants logiciels puis on a cité les différents modèles de composants et les ADLs qui spécifient l'interface d'un composant et nous avons présenté quelques techniques pour la recherche de composants . Nous avons vu qu'il en existe une grande variété. La deuxième partie était consacrée à l'implémentation. A ce niveau, nous avons présenté l'architecture générale de notre application avant de détailler la structure et le fonctionnement des différents modules. Comme perspectives, nous envisageons d'integré un niveau sémantique de la recherche.

Annexe

Définition :

RMI est un ensemble de classes qui permettent la communication entre machines virtuelles Java (JVM) qui peuvent se trouvent physiquement sur la même machine ou sur deux machines distinctes : c.à.d. la manipulation des objets sur des machines distantes (objets distants) de manière similaire aux objets sur la machine locale (objet locaux). Ces manipulations sont "relativement" transparentes. Pour cela RMI propose : - un ramasse-miettes distribué. - la gestion des représentants locaux d'objets distants et leur activation. - la liaison avec les couches transport et l'ouverture de sockets appropriés. 14 - la syntaxe d'invocation, d'utilisation d'un objet distant est la même qu'un objet local. En Java, une fois que l'environnement détermine qu'un objet n'était plus utile, le ramasse-miettes (en anglais, Garbage Collector) libère la mémoire qu'il occupe. L'avantage est immense par rapport aux langages plus anciens, où le développeur devait connaître à tout instant ce qui se trouvait dans la mémoire, et gérer de manière efficace sa libération.

Application RMI

Une application RMI est composée d'une partie client et d'une partie serveur. Le rôle d'un serveur est de créer des objets qu'on qualifie de « distants », de les rendre accessibles à distance et enfin d'accepter des connexions de clients vers ces objets. Le rôle d'un client est donc d'accéder aux méthodes de ces objets, tout en considérant comme si ces objets étaient des objets locaux. RMI est très souvent utilisé en parallèle avec l'API d'annuaire JNDI (services de nommage).
afin que les clients trouvent les services distants, ceux-ci doivent au préalable être enregistrés dans un annuaire, pour cela RMI fourni un rmiregistry.

rmiregistry :

possède une table de hachage dont les clés sont des noms et les valeurs sont des objets distants. Package RMI : C'est un système d'objets distribués constitué uniquement d'objets Java, et qui permet et assure la communication entre machines virtuelles Java.

Bibliographie

[1] Nassima Saididj.Modèles cmposants,Université de Bretagne Occidentale,2009

[2] Nassima Sadou. Evolution Structurelle dans les Architectures Logicielles à base de Composants.Génie logiciel [cs.SE]. Université de Nantes ; Ecole Centrale de Nantes (ECN), 2007.

[3] Lamia Yessad ,Sélection sémantique de composants logiciels basées sur des criteres de qualité de service, Projet d'obtention de diplôme de doctorat. L'université de Mentouri-Constantine.2011-2012.

[4] Salah HADJAB,Assemblage de Composants Logiciels :LesComposants- Composites, Mémoire de stage-Diplôme d'Études Approfondies.

[5] KOFFI Kouakou Ive Arsène, Docteur BROU Konan Marcellin, Prof. OUMTAGADA Souleymane, « DEVELOPPEMENT DE METHODES AUTOMATIQUES POUR LA REUTILISATION DES COMPOSANTS LOGICIELS »

[6] Sofiane Batata ,« Moteur de recherche pour la sélection de composants logiciels ».Ecole Nationale Supérieure d'Informatique (Ex. INI), 2011.

BATATA, Sofiane. Moteur de recherche pour la sélection de composants logiciels.

[7] Oualid Khayati, « Modèles formels et outils génériques pour la gestion et la recherche de composants »,2005

[8]] Kung-Kiu Lau ,Simone di Cola « An Introduction To Component-Based Software Development »,Mars 2017.

[9] Ivica Crnkovic, Magnus Larsson, « Component-Based Software Engineering – New Paradigm of Software Development »

[10] Frédéric Peschanski, Thomas Meurisse, Jean-Pierre Briot, « Les Composants Logiciels : Evolution technologique ou nouveau paradigme»

[11] Bart GEORGE, Un processus de sélection de composants logiciels multi-niveaux. Thèse de doctorat, Université de Bretagne Sud,2007.

[12] Y.Maarek, D. Berry, and G. Kaiser. An information retrieval approach for automatically constructing software libraries. IEEE Transactions on Software Engineering, Vol 17 Issue 8, Pages 800-813, August 1991.

[13] Projet ACCORD, Etat de l'art sur les Langages de Description d'Architecture (ADLs)»Livrable 1.1-2, Juin 2002.

[14] Christophe Mareschal, « Adaptation d'un langage de description d'architecture à l'expression du comportement ; applications ».

[15] http ://www-igm.univ-mlv.fr/ dr/XPOSE2002/CCM/corba.htm

[16] http ://www-igm.univ-mlv.fr/ dr/XPOSE2002/CCM/corba.htm

[17] Adel Alti , Coexistence de la modélisation à base d'objets et de la modélisation à base de composants architecturaux pour la description de l'architecture logicielle,22juin2011,thése.

[18] Mohamed Oussama Hassan,Contribution à l'analyse d'impact des modifications des architectures logicielles, Thèse.2009

[19] Ruben Prieto-Diaz. Implementing faceted classification for software reuse. Communications of the ACM, vol 34 (5) pages 88.97, May 1991.

[20] Z. Zhang, L. Svensson, U. Snis, C. Srensen, H. Fgerlind, T. Lindroth, M. Magnusson, and C. Stlund. Enhancing component reuse using search techniques. In Proceedings of IRIS 23, Laboratorium for Interaction Technology, University of Trollhttan Uddevalla, 2000.

[21] Amy Zaremski and Jeannette Wing. Signature matching : a tool for using software libraries. ACM Transactions On Software Engineering and Methodology (TOSEM), vol 4 Issue 2 , pages 146.170, April 1995.

[22] Podgursky A., Pierce L., Behaviour sampling : a technique for automated retrieval of reusable components, 14th International conference on software engineering, pages 300-3004, new york, NY :ACM Press, 1992.

[23] Podgursky A., Pierce L., Retrieving reusable software by sampling behavior, acm Trans on Software Engineering and Methodology, vol 2 issue 3 pages :286-303, July 1993.

[24] Park Y., Bais P., Generating samples for component retrieval by execution, Technical report, University of Windsor, Canada, 1997.

[25] YAHLALI Mebarka,Assemblage d'une application à base de composants :approche de calcul de qualité d'une composition, Thèse pour l'obtention du diplôme de Doctorat.Université de l'USTO 2014/2015.

[26] Amy Zaremski and Jeannette Wing. Signature matching : a tool for using software libraries. ACM Transactions On Software Engineering and Methodology (TOSEM), vol 4 Issue 2 , pages 146.170, April 1995.

[27] Karine Macedo , Modélisation d'aspects qualité de service en UML : application aux composants logiciels, Rapport de thèse, l'université de Rennes, Mai 2004 .

[28] El Boussaidi, Ghizlane, and Hafedh Mili. "Les langages de description d'architectures." Synthèse 4 (2006) : 2.

Table des figures