République Algérienne Démocratique et Populaire

وزارة التعليم العالي والبحث العلمي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie d'Oran Mohamed Boudiaf

Faculté des Mathématiques et Informatique
Département d'Informatique

THESE

Présentée par

Melle YAHLALI Mebarka

Pour l'obtention du diplôme de Doctorat en Science

Spécialité : Informatique

Option : Ingénierie des Logiciels et des Réseaux

Thème

Assemblage d'une application à base de composants : approche de calcul de qualité d'une composition

Soutenue publiquement le :

Devant le jury :

Mme BELBACHIR Hafida	**Professeur**	**Présidente**	**USTO-MB**
Mr BOUZIDI Laid	**Professeur**	**Examinateur**	**UNIV. LYON**
Mr YAGOUBI Belabbes	**Professeur**	**Examinateur**	**UNIV.ORAN**
Mr MALKI Mimoun	**Professeur**	**Examinateur**	**UNIV. SIDI BEL ABBES**
Mme NOUREDDINE Myriam	**Maître de Conférence-A**	**Examinatrice**	**USTO-MB**
Mr CHOUARFIA Abdallah	**Professeur**	**Rapporteur**	**USTO-MB**

Année universitaire : 2014/2015

A Mes très chers Parents

Remerciement

Mes remerciements vont tout premièrement à Dieu tout puissant pour la volonté et la santé.

Mes plus sincères remerciements vont à M. Abdallah CHOUARFIA , Professeur à l'Université des Sciences et de la Technologie d'Oran « USTOMB » pour avoir bien voulu rapporter cette thèse et pour tous ce qu'il m'a apporté : idées, orientations et encouragements.

Je remercie les membres du jury d'avoir accepté d'évaluer ce travail de thèse. D'abord, je remercie Professeur H.BELBACHIR d'avoir accepté de présider ce jury. Aussi, je remercie Professeur L.BOUZIDI, Professeur B.YAGOUBI, Professeur M.MALKI et Docteur M.NOUREDDINE d'avoir accepté d'examiner cette thèse. Je suis honorée par leur présence dans mon jury de soutenance.

Merci également à mes collègues de travail à l'Université de Saida et l'Université des Sciences et Technologie d'Oran, pour leurs encouragements, et leur sympathie.

Je désire manifester ma gratitude à ma famille et surtout ma mère, mon père, ma chère Mimi, Docteur Mohamed, Professeur Khaled, Capitaine Azza, Zohra, Djamilla et Noudjoud. Merci pour le soutien indéfectible.

RESUME

L’approche CBSE (Component-Based Software Engineering) vise à développer des logiciels par l'assemblage et le déploiement d'unités réutilisables, appelés composants logiciels. Cette approche tente à améliorer la flexibilité, la réutilisabilité et la maintenabilité des applications et permette le développement des applications complexes déployées sur une large gamme de plateformes, en connectant les composants, plutôt que de les construire à partir de zéro.

Dans ce contexte, l'étape de sélection est très importante. Elle consiste à rechercher et à sélectionner les composants logiciels appropriés à partir d'un ensemble de composants candidats afin de satisfaire aux exigences spécifiques de développeurs. Dans le processus de sélection, les exigences fonctionnelles et non fonctionnelles sont généralement considérées.

Dans cette thèse ; nous proposons une méthode permettant d'évaluer la qualité de l'assemblage de composants logiciels et de choisir la meilleure composition en terme de qualité.

ABSTRACT

The paradigm of Component-Based Software Engineering (CBSE) aims to develop software by assembling and deploying reusable units, called software components. This approach tries to improve the flexibility, re-usability and maintainability of applications, and helps develop complex and distributed applications deployed on a wide range of platforms, by plugging commercial off-the-shelf (COTS) components, rather than building them from scratch.

In this context, the selection step is very important. It consists of searching and selecting appropriate software components from a set of candidate components in order to satisfy the developer-specific requirements. In the selection process, both functional and non-functional requirements are generally considered.

In this thesis we propose a method enabling quality evaluation of software component assembly. This method allows us to choose the best composition in term of quality.

ملخـص

النهج CBSE (Component-Based Software Engineering) : هندسة البرامج القائمة على المكون يهدف إلى البرمجة من خلال تجميع و تشغيل وحدات قابلة لإعادة الاستخدام تسمى مكونات البرامج .

يحاول هذا النهج تحسين المرونة, إعادة الاستخدام , صيانة البرامج وتمكين تطوير التطبيقات المعقدة وذلك بربط المكونات بدلا من البرمجة من النقطة صفر.

في هذا السياق , خطوة اختيار المكون مهمة جدا فهي تهدف إلى بحث وتحديد المكون المناسب من مجموعة من المكونات المرشحة لتلبية المتطلبات المحددة من طرف المبرمجين . في عملية الاختيار , المتطلبات الوظيفية وغير الوظيفية تأخذ بعين الاعتبار.

في هذه الأطروحة نقترح طريقة لتقييم نوعية البرامج القائمة على المكون و اختيار أفضل تركيبة من حيث الجودة.

Table des matières

CHAPITRE II : QUALITE DE LOGICIEL

CHAPITRE III : RECHERCHE ET SELECTION DE COMPOSANTS LOGICIEL

PARTIE 2 : CONTRIBUTION

CHAPITRE IV : APPROCHE PROPOSEE

SoftwareComponent MetaData

Les Processus d'évaluation

CHAPITRE V: IMPLEMENTATION

Table des figures

INTRODUCTION GENERALE

Introduction Générale

I. Contexte

Cette thèse s'inscrit dans le domaine de l'ingénierie à base de composants. Elle s'adresse plus spécifiquement à la problématique de l'évaluation de la qualité d'une application basée sur l'assemblage de composants logiciels.

Le paradigme "composant" est apparu en réponse aux limites de l'approche de conception par objets. Il a introduit une nouvelle méthode pour la conception des applications logicielles. Cette méthode dénommée CBSE (Component-Based Software Engineering) est basée sur l'assemblage d'entités logicielles préfabriquées appelées « composants ».

Dans CBSE, on doit faire la distinction entre deux cycles de vie : le premier pour le développement de composants réutilisables (*Design for Reuse*) et le deuxième pour le développement d'applications à partir de ces composants (*Design by Reuse*). Les différentes phases du second processus sont : l'analyse et la spécification des besoins, la conception générale du système logiciel, la sélection et le test des composants qui vont être intégrés, la vérification, la validation du système complet et enfin la maintenance. Dans le cycle *Design by Reuse* , la phase d'implémentation est remplacée par la sélection des composants qui consiste à déterminer l'adéquation de composants déjà développés au contexte d'un système particulier

Les composants sont élaborés à partir des besoins fonctionnels des utilisateurs. Leur qualité ne fait pas souvent partie du processus de développement. Ainsi, face à l'immensité de l'offre disponible (des milliers de logiciels peuvent assurer les mêmes services demandés). Il est nécessaire d'avoir un mécanisme de recherche automatique qui permet au moins de retrouver les composants les plus pertinents qui répondent aux besoins fonctionnels demandés avec un certain niveau de qualité (besoins non fonctionnels).

II. Objectif

Dans cette thèse, nous nous intéressons à la problématique de la qualité des applications basées sur l'assemblage de composants logiciels.

Le point de départ de cette recherche a été l'étude des processus de sélection de composants logiciels et les langages de description d'architecture. Au cours de cette recherche nous avons constaté que le concept *qualité* est absent dans la description et que la majorité des processus de sélection utilisent le principe : « comparaison paire à paire » des caractéristiques de qualité pour l'évaluer.

L'objectif principal de ce travail est de trouver un moyen qui facilite la sélection et permet à l'utilisateur de choisir le meilleur assemblage parmi plusieurs propositions équivalentes en termes de besoins fonctionnels et non fonctionnels (qualité).

III. Contributions

Les principales contributions de cette thèse sont :

1. Nous avons proposé deux processus d'évaluation de la qualité :
- le premier nommé SCEP: Software Component Evaluation Process, pour le calcul de la qualité d'un composant logiciel.
- et le deuxième SCAEP : Software Component Assembly Evaluation Process, basé sur le premier et permet l'évaluation de la qualité d'un assemblage de composants logiciels.

2. Dans chaque processus d'évaluation de qualité la présence d'une description des caractéristiques est cruciale. C'est pourquoi une métadonnée baptisée SCM (Software Component Metadata) a été proposée pour la description des composants et des assemblages. Elle s'intéresse aux aspects fonctionnels et non fonctionnels des composants logiciels.

3. En plus des outils proposés, nous avons développé un environnement de simulation de qualité qui facilite la comparaison des applications basées sur l'assemblage de composants logiciels.

III. Organisation du document

La suite de ce manuscrit est organisée de la façon suivante :

1. Partie I : Etat de l'art

Cette partie présente un état de l'art des travaux en relation avec le sujet de la thèse. Elle est constituée de trois chapitres :

- **Chapitre I : Approche à base de composants (CBSE)**

  Ce chapitre donne un aperçu sur les composants logiciels : définition, architecture, caractéristiques…, et présente aussi le cycle de vie de développement d'une application logiciel ainsi qu'un état de l'art sur les principaux langages de description d'architecture.

- **Chapitre II : La Qualité de Logiciel**

  Le deuxième chapitre est entièrement consacré à la notion et l'historique de la qualité en général et de la qualité du logiciel en particulier. Nous présentons aussi Le Standard ISO 9162 et les langages de spécification de la qualité.

- **Chapitre III : Recherche et sélection de composants**

  Ce chapitre présente un aperçu des processus de sélection de composants. Nous donnons les techniques de recherche de composants ensuite une étude comparative des processus de sélection est présentée.

2. Partie II : La contribution

Cette partie est constituée de deux chapitres :

- **Chapitre IV : L'approche proposée**

  Notre approche se compose de deux sous parties :

  1. Software Component Metadata : est consacrée à la présentation des métadonnées proposées et les définitions régulières (grammaire).

  2. Les processus d'évaluation : présente les deux processus d'évaluation de qualité proposés.

- **Chapitre V : Implémentation**

  Dans ce chapitre nous présentons le prototype de l'environnement de simulation de qualité développé.

Nous terminons ce manuscrit par la conclusion de nos travaux et nous en décrivons les perspectives.

PARTIE 1 :

ETAT DE L'ART

CHAPITRE I : APPROCHE A BASE DE COMPOSANTS

"CBSE : COMPONENT BASED SOFTWARE ENGINEERING"

SOMMAIRE

I. Introduction

L'approche à base de composants est depuis une dizaine d'années considérée comme un nouveau paradigme de développement des systèmes d'information. Elle apporte une nouvelle vision au domaine du génie logiciel où la construction d'applications se base essentiellement sur des entités existantes appelées « composants logiciels ».

L'idée générale de ce nouveau paradigme de développement est de pouvoir construire son application comme on assemble un puzzle.

Cette approche est motivée par des arguments économiques, tels que la réduction du temps et du coût de développement, ou bien la spécialisation des acteurs intervenant dans le cycle de vie du développement.

Ce chapitre présente un état de l'art de l'approche à base de composants. Nous commençons par la présentation de quelques notions générales sur les composants, ensuite nous présentons le développement à base de composants et dans la dernière section nous étudions l'assemblage de composants logiciels.

II. Notions générales sur les composants

II.1. Définition d'un composant

Si l'idée de composant logiciel est l'essence même du CBSE, sa définition est sujette à de nombreuses propositions différentes. Les nuances entre les diverses définitions sont toutefois assez subtiles mais celle de Szyperski et Pfister [1] la plus reprise dans la littérature, semble faire consensus

"A Software component is unit of composition with contractually specified interfaces and explicit context dependencies only. A Software component can be deployed independently and is subject to composition by third parties".

De cette définition on peut déduire les points suivants :

- Un composant logiciel est une unité de composition ayant des interfaces spécifiées de façon contractuelle et possédante des dépendances de contextes explicites.
- Un composant logiciel peut être déployé sur différentes plates-formes.
- Un composant Peut être sujet de composition par un tiers (Assemblage de composants pour la conception d'applications logicielles).

II.2. Caractéristiques d'un composant logiciel

Les caractéristiques globales d'un composant définies par Medvidovic et Taylor **[2]** sont :

- **L'interface d'un composant :** L'interface d'un composant est la description de l'ensemble des services offerts et requis par le composant. L'interface est un moyen d'expression des liens du composant ainsi que ses contraintes avec l'extérieur.
- **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.
- **La sémantique d'un composant :** La sémantique du composant est exprimée en partie par son interface. Cependant, l'interface 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.
- **Les contraintes d'un composant** : Les contraintes d'un composant définissent les limites d'utilisation d'un composant et ses dépendances intra composants. Une contrainte est une propriété qui doit être vérifiée sur un système. Si celle-ci est violée, le système est considéré comme un système incohérent et inacceptable.
- **Les propriétés non fonctionnelles d'un composant** : Les propriétés non fonctionnelles présentent l'ensemble des propriétés liées à la sécurité, la performance, la portabilité. Ces propriétés permettant 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.

II.3. Architecture d'un composant

L'architecture d'un composant présentée dans la figure 2 spécifie les entrées et les sorties d'un composant afin de faciliter la description de son comportement (l'ensemble des services offerts).

Un composant logiciel possède principalement trois éléments **[3]** :

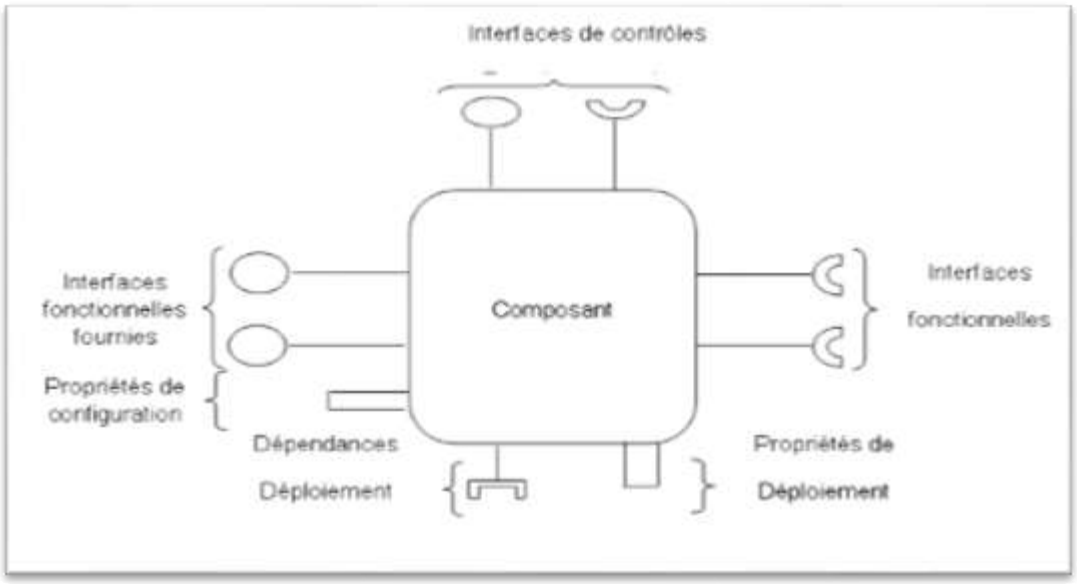


Figure I.1 : Architecture d'un composant

a) **Interfaces fonctionnelles et propriétés de configuration :** Les interfaces fournies (sorties) et les interfaces requises (entrées) peuvent être des méthodes (des fonctions promises aux clients). Les interfaces fonctionnelles requises doivent être satisfaites lorsqu'une instance de composant est créée pour que celle ci puisse être utilisée à travers les interfaces fournies.
Les propriétés de configuration permettent de configurer une instance de composant, par exemple, changer le nom d'un bouton.

b) **Interfaces de contrôle (fournies / requises) :** Les interfaces de contrôle présentent l'ensemble des méthodes qui permettent de gérer le cycle de vie des instances du composant pendant l'exécution**.** Ces méthodes sont destinées à être appelées par l'environnement d'exécution du modèle à composants.

c) **Dépendances et propriétés de déploiement :** Les dépendances sont propres à une implémentation de composant. Elles doivent être satisfaites au moment du déploiement d'une classe de composant pour permettre son utilisation. Elles peuvent représenter par exemple une dépendance envers une version particulière de

l'environnement d'exécution, des librairies, des ressources binaires (images, sons), des fichiers de configuration …

Les propriétés de déploiement sont définies au niveau de l'implémentation, sont similaires aux variables de classe dans l'approche objet. Les propriétés d'implémentation sont employées pour configurer des caractéristiques communes à toutes les instances.

II.4. Autres concepts

1. **Implantation d'un composant :** L'implantation d'un composant est la réalisation exécutable du composant, obéissant aux règles du modèle de composant. Elle peut être fournie en code compilable C, en forme binaire etc.… [8].

2. **Paquetage d'un composant :** Un paquetage d'un composant est une entité « déployable » (souvent une archive) contenant son type, au moins une de ses implantations, ainsi que les contraintes techniques associées à chaque implantation [6]

3. **Déploiement :** Le déploiement est la phase qui consiste à diffuser, installer et paramétrer des composants sur les sites d'une infrastructure matérielle et système, afin de rendre une application prête à être utilisée [6].

II.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 [4] :

- **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.

II.6. Modèle de composants

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. Concrètement, un modèle de composants décrit certains aspects des composants comme la définition de composants à partir d'objets (classes, modules, etc.), les relations entre les composants, les propriétés Le modèle de composants permet de définir les entités et leur mode d'interaction et de composition [5].

II.6.1. Exemples de modèles de composants logiciels

II.6.1.1. CORBA Component Model (CCM)[11]

CCM est un modèle riche qui définit le cycle de vie complet de la conception d'une application à base de composants. Il propose un modèle abstrait pour spécifier un composant, un modèle concret pour séparer son code métier du code de gestion, un modèle d'assemblage avec un langage de description d'architecture et enfin, un modèle de déploiement pour définir les étapes du déploiement d'un assemblage.

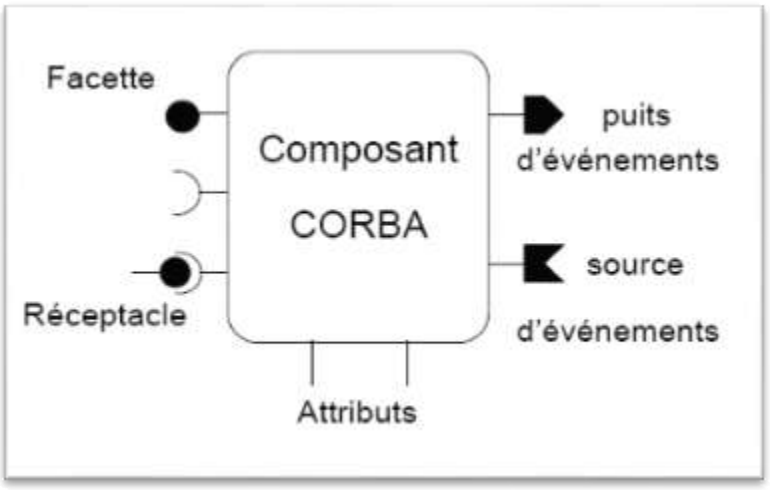


Figure I.2 : Composant CORBA et les différents types de port.

Un composant CORBA a la possibilité d'avoir plusieurs interfaces associées avec un unique objet. Quatre types de ports sont définis dans le contexte du CCM.

- **Les facettes** : une facette est une interface fournie par un type de composant et qui est utilisée par des clients en mode synchrone. Seule l'interface dont le client a besoin est fournie.
- **Les réceptacles** : un réceptacle est une interface utilisée par un type de composant en mode synchrone. Il s'agit d'un point de connexion conceptuel. Il permet donc d'assembler des composants et des objets.

Il y a deux types de ports qui traitent des événements :

- **Les puits** : un puits d'événements est une interface fournie par un type de composant et utilisée par ses clients en mode asynchrone. Il permet à un composant de recevoir des événements d'un certain type.
- **Les sources** : une source d'événements est une interface utilisée par un type de composant en mode asynchrone. Il y a deux catégories de sources d'événements: les connectables en mode un vers un, et les connectables en mode un vers n.

II.6.1.2. Fractal [12]

A l'origine, l'objectif de FRACTAL est d'offrir un modèle dédié à la construction, au déploiement et à l'administration des systèmes logiciels complexes, comme les systèmes d'exploitation. Toutefois, ce modèle est aussi adopté pour concevoir des applications distribuées.

Un composant est constitué de deux parties :

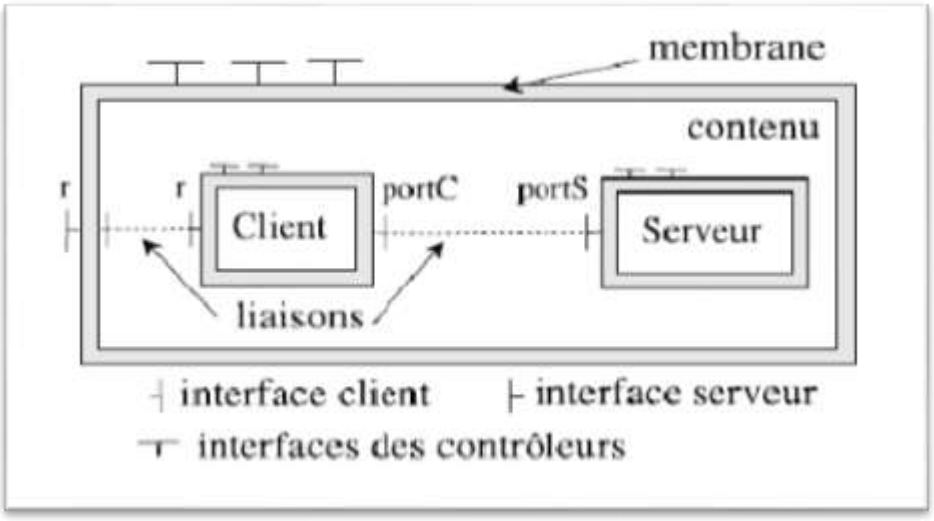


Figure I.3 : Exemple de composant FRACTAL

- **Un contenu**, représenté par l'intérieur blanc du rectangle, est constitué de sous-composants ou bien d'une implémentation dans le cas d'un composant primitif.
- **Une membrane**, représentée par la bordure grise d'un rectangle, expose les ports du composant.

Les ports, nommés *interfaces* dans FRACTAL, spécifient des communications par appels de procédures ou de méthodes. De plus, les interfaces peuvent être *externes* ou *internes* au composant. Les interfaces internes permettent de relier les interfaces externes d'un composant composite aux interfaces des sous-composants.

II.6.1.3. Service Component Architecture [13]

Ce modèle a pour objectif de pouvoir réaliser la composition de services indépendamment des technologies utilisées pour la réalisation de ces services et indépendamment de la plate-forme implémentant la spécification SCA.

Un composant peut exposer une fonctionnalité fournie via un port serveur nommé service, ou bien une fonctionnalité utilisée via un port client nommé référence. Le mode de communication spécifié par un port peut être le passage de message, Services Web ou appel de procédures ou de méthodes à distance (RPC/RMI). SCA définit aussi le concept de propriété pour désigner un attribut d'un composant.

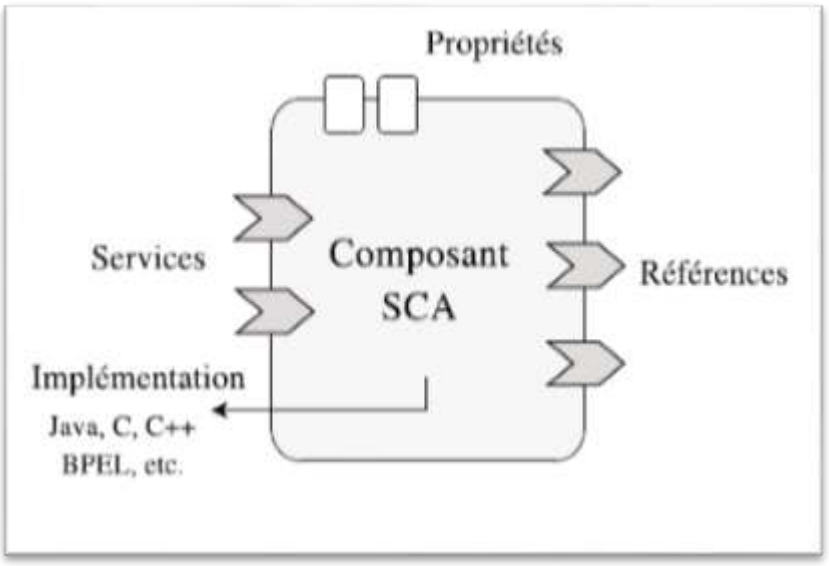


Figure I.4 : Un composant SCA

III. Développement à base de composants

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).

III.1. Design by reuse

Les différentes phases de ce processus sont [7] :

1. **Conception générale du système logiciel**. La conception d'un assemblage est réalisée en étudiant la façon d'intégrer un ensemble de composants préexistants pour créer une composition représentant soit une partie ou bien la totalité d'une application.
2. **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.

3. **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.
4. **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.

III.2. Design for reuse

Dans cette phase la réutilisabilité est l'enjeu principal : les composants sont crées pour être réutilisés dans différentes applications [4]. 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 (design by reuse).
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.

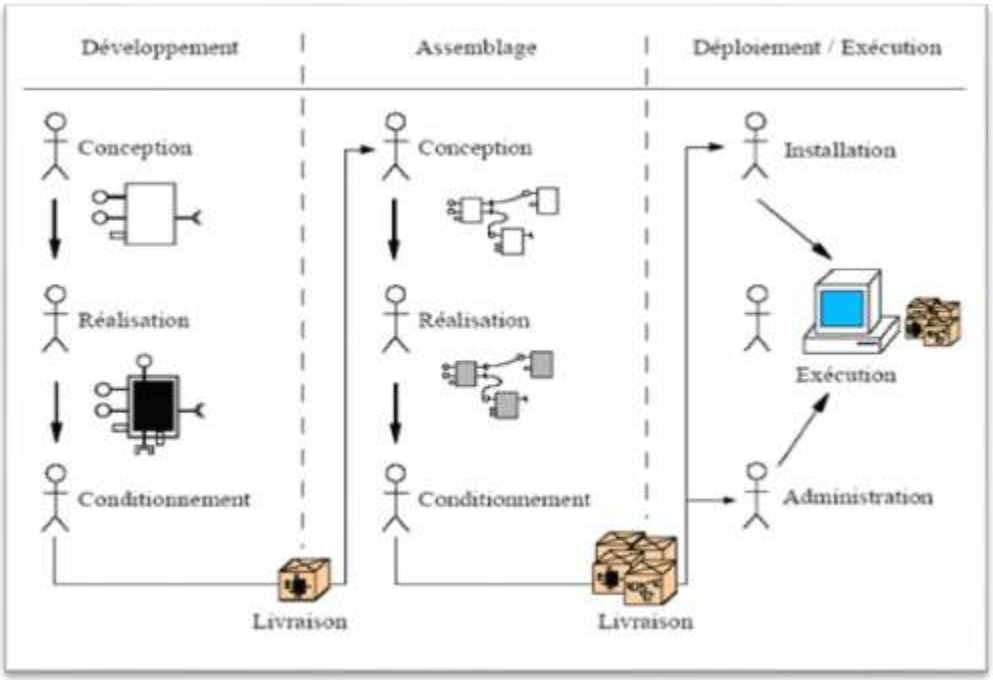


Figure I.5: Développement à base de composants [3]

IV. Assemblage de composant logiciels

Selon [8] l'assemblage de composant est découpé en deux parties :

- L'aspect ensembliste de l'assemblage : la composition d'un assemblage (les différentes entités qui constituent un composant).
- L'aspect de communication entre composants : les interactions entre composants.

Les relations de dépendance (cf. Figure I.3) entre les interfaces de composants sont :

a. **Dépendances de services offerts (liaisons de type 1) :** désignent les liaisons entre interfaces offertes visibles du composite et interfaces offertes par des composants membres du composite. Dans ce cas, la spécification offerte du composite, rendue visible, est identique à la spécification offerte d'un composant membre (principe de délégation).

b. **Dépendances de services requis (liaisons de type 2) :** désignent les liaisons entre interfaces requises de composants membres et les interfaces visibles du composite (principe de délégation).

c. **Dépendances inter-composants (liaisons de type 3) :** il s'agit de liaisons d'assemblage entre interfaces requises et interfaces offertes. Dans le cas général, il existe une relation de dépendance entre services offerts et requis.

d. **Dépendances intra-composants (liaisons de type 4) :** ces liaisons sont créées pour supporter les dépendances d'implantation entre interfaces offertes et interfaces requises d'un composant.

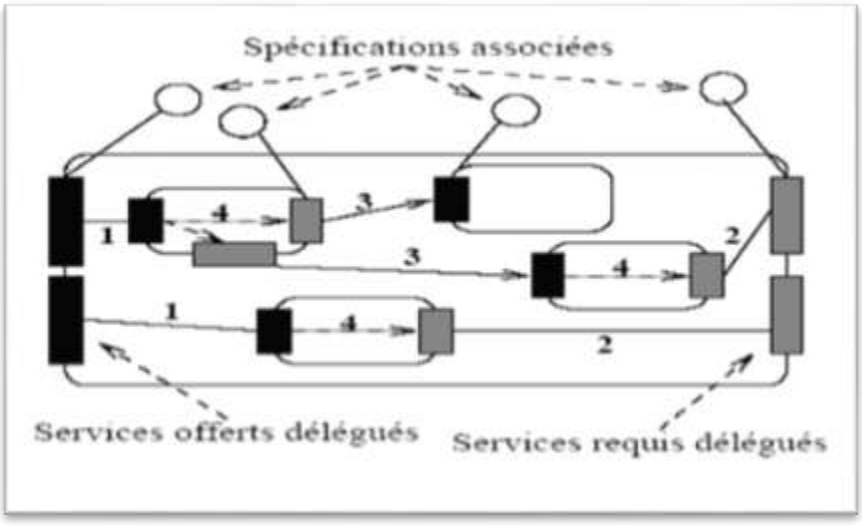


Figure I.6 : relation de dépendance entre les interfaces de Composants

V. Les langages de description d'architecture

Un ADL (Architectur Description Language) est un langage offrant des formalismes pour modéliser une architecture logicielle à base de composants d'un système. Ils sont en général graphiques et textuels et possèdent des outils associés [45].

V.1. Les exigences minimales fondamentales

Un ADL doit, en principe, pouvoir décrire une architecture logicielle sous la forme des trois C : les **C**omposants, les **C**onnecteurs et les **C**onfigurations (topologies) [46].

1. **Composant :** Un composant représente une unité de calcul ou de stockage de données à laquelle est associée une unité d'implémentation. Il peut être aussi petit qu'une procédure simple ou aussi grand qu'une application. Il est l'élément de base, et doit être composable, auto descriptif, configurable, réutilisable et autonome [48]. Tous les ADL existants modélisent les composants sous une forme ou une autre.
2. **Connecteur** : Est un bloc de construction utilisé pour exprimer les interactions entre composants ainsi que les règles qui gouvernent cette interaction [2]. Ils sont des entités architecturales qui relient des ensembles de composants et agissent en tant que médiateurs entre eux. Par exemple, un connecteur peut décrire des interactions simples de type appel de procédure, ou encore des interactions complexes telles que des protocoles d'accès à des bases de données avec gestion des transactions, …

   Un connecteur comprend également deux parties. La première correspond à la partie visible du connecteur, c'est-à-dire son interface, qui permet la description des rôles des participants à une interaction. La seconde partie correspond à la description de son implantation [49].
3. **Configuration** : Une configuration définit la structure et le comportement d'une application formée de composants et de connecteurs. Il existe deux types de configurations [2]:

   - **La configuration structurelle :** correspond à un graphe connexe des composants et des connecteurs formant l'application.
   - **La configuration comportementale :** modélise le comportement en décrivant l'évolution des liens entre composants et connecteurs, ainsi que

l'évolution des propriétés non fonctionnelles comme les propriétés spatio-temporelles ou la qualité de service.

V.2. Les principaux ADLs :

V.2.1. ACME [50]

ACME a été développé à l'université Carnegie Mellon. Il a pour buts principaux de fournir un langage pivot qui prend en compte les caractéristiques communes de l'ensemble des ADLs, qui soit compatible avec leurs terminologies et qui propose un langage permettant d'intégrer facilement de nouveaux ADLs. Dans ACME, la structure architecturale est décrite à l'aide des concepts suivants:

1. **Le composant** : représente l'unité de traitement ou de donnée d'une application. Il est décrit par des interfaces composées de ports (fournis ou requis). Un composant peut être primitif ou composé.
2. **Le connecteur :** décrit une connexion entre les composants. Il est également décrit par des interfaces définies par un ensemble de rôles.
3. **Un système** : représente la configuration d'une application, c'est-à-dire l'assemblage structurel entre les composants et les connecteurs
4. **Un attachement** relie le port d'un composant à un rôle d'un connecteur.
5. **Une liaison** relie les ports d'un composant composite aux ports de ses composants internes.
6. **Représentation** : chaque description (générale ou détaillé) d'un élément est appelée une représentation.
7. **Carte de représentation :** la correspondance entre l'élément et ses représentations est spécifiée grâce à la carte de représentation. Ainsi, la carte de représentation permet d'établir la correspondance entre les ports de l'interface d'un composant et ceux définis dans les interfaces de ses sous composants.

L'exemple suivant décrit une architecture client-serveur en ACME :

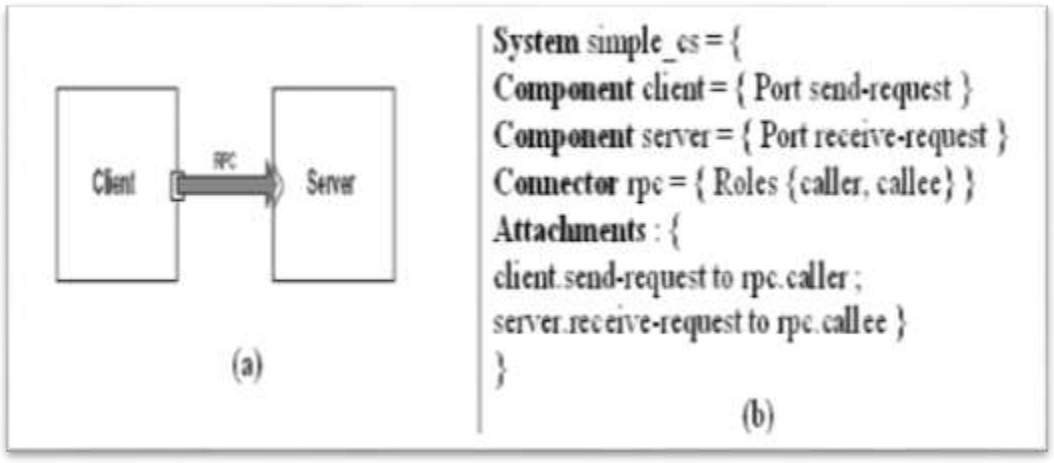


Figure I.7: (a) Exemple d'architecture client /serveur
(b) Description en ACME

Synthèse

ACME fournit des bases simples qui forment un point de départ pour le développement de nouveaux ADLs [47]. Il permet l'intégration des outils pour décrire la sémantique comme par exemple la logique de premier ordre. Cependant, ACME ne permet pas la spécification directe des propriétés non fonctionnelles.

V.2.2. Rapide [52]

RAPIDE, développé à l'Université de Stanford. Il a pour but initial de vérifier par la simulation la validité d'une architecture logicielle donnée. Une application est construite sous la forme de modules ou composants communiquant par échange de messages ou évènements. Le simulateur associé à Rapide permet ensuite de vérifier la validité de l'architecture. Les concepts de base du langage Rapide sont :

1. **L'événement :** est une information transmise, c'est-à-dire une demande de service, soit une valeur particulière d'un attribut. Il permet de construire des expressions appelées *eventpatterns* qui caractérisent les événements circulant entre composants. La construction de ces expressions se fait avec l'utilisation d'opérateurs qui définissent les dépendances entre événements. L'ensemble de ces opérateurs est répertorié dans le tableau suivant :

Opération	Sémantique
A>B	L'événement B est envoyé après l'événement A
A->B	B dépend causalement de A
A||B	A et B ne sont pas causalement dépendants
A~B	A et B sont différents
A and B	A et B sont vérifiés simultanément

Tableau I.1 : Expressions d'événements dans Rapide [53]

2. **Le composant (**module) est défini par une interface qui est constituée d'un ensemble de services fournis et d'un ensemble de services requis. Les services sont de trois types :
    - **Provides :** fournis par le composant, ils sont appelés de manière synchrone par d'autres composants,
    - **Requires :** demandés par le composant, ils sont appelés de manière synchrone,
    - les « **Actions** » qui correspondent à des appels asynchrones entre composants ; deux types d'actions existent : les actions **in** et **out** qui sont des événements acceptés et envoyés par un composant.

L'architecture contient la déclaration des instances de composants et les règles de connexions entre ces instances. Toutes les instances sont déclarées sous forme de variables. La règle d'interconnexion est composée de deux parties. La première est la partie gauche qui contient une expression d'événements qui doit être vérifiée, la seconde est la partie droite qui contient également une expression d'événements qui doivent être déclenchés après la vérification de l'expression de la partie de gauche. Les contraintes (*constraint*) peuvent être utilisées pour décrire l'architecture. Elles permettent de restreindre le comportement de l'architecture en définissant des patrons d'événements à appliquer pour certaines connexions entre composants.

```
with Client, Serveur;
...
-- déclaration des instances de composants de l'application susceptible
d'exister
?s : Client; -- fait référence à une instance de Client
!r : Serveur; -- fait référence à toutes les instances de serveur
?d : Data; -- fait référence à un bloc de paramètre d'un certain type Data
...
-- Une règle d'interconnexion
?s.Send(?d) => !r.Receive(?d);;
-- Si un client transmet un événement de type Send avec ce type de
paramètres,alors, l'événement est transmis à tous les serveurs de
l'application avec ces paramètres.
```

Figure I.8 : Exemple d'expression d'événement Rapide avec un client et un serveur[2]

Synthèse

En utilisant un modèle d'événements, RAPIDE permet la spécification de la dynamique d'un système. En plus, il est doté d'un environnement textuel et graphique pour décrire une architecture. Cet environnement inclut un analyseur syntaxique, un compilateur et un outil permettant l'analyse d'une architecture par simulation. Cependant, RAPIDE comme la plupart des ADLs, ne supporte pas la modélisation des propriétés non fonctionnelles.

V.2.3. Kmelia [51]

Kmelia est un modèle et un langage à composants multiservices. Les services décrivent des fonctionnalités avec la possibilité d'interagir avec d'autres services. Les caractéristiques principales du modèle Kmelia sont :

1. **Composant :** un composant est défini par un espace d'états, des services accessibles par leur nom. L'espace d'état est un ensemble de constantes et de variables typées. Dans l'interface d'un composant on distingue les services offerts qui réalisent des fonctionnalités et les services requis qui déclarent les besoins du composant.

```
COMPONENT C1
  INTERFACE
    provides : <ServName list>
    requires : <ServName list>

  //espace d'états du composant
  TYPES
       <Type Defs>
  VARIABLES
       <Var list>
  INVARIANT
       <Predicate>
  INITIALIZATION
    ...
    // affectations des var

  //services du component
  SERVICES
    ...
    // voir détail des services

  END_SERVICES
```

Figure I.9 : Structure d'un composant [51]

- **Service :** Un service modélise une fonctionnalité élémentaire ou complexe. Outre sa signature fonctionnelle, un service est constitué d'une interface (à l'image de l'interface d'un composant, c'est l'ensemble des services dont il dépend), d'un espace d'états, d'un contrat sous la forme de pré/post-conditions et éventuellement d'un comportement dynamique.

```
provided aService_1
       (<param>) : <ResultType>

  Interface
    subprovides : <Serv list>
    calrequires : <Serv list>
    extrequires : <Serv list>
    intrequires : <Serv list>
    ...

  Pre <Predicate>

  Variables # espace d'état local
     <Var list>
  Initialization
    ... // affectations

  Behavior
    Init  <initial state>
    Final <final states>
    //eLTS
    {<transition list>}

  Post <Predicate>
End

required aService_2 ()
 ...  // défini de la même manière
```

Figure I.10 : Structure d'un service [51]

Synthèse

Kmelia est un langage et un modèle à composants multiservice où les composants sont abstraits et formels de façon à pouvoir y exprimer des propriétés et les vérifier. La hiérarchisation des services et des composants est l'une des caractéristiques de Kmelia qui permet une bonne lisibilité, la flexibilité et une bonne traçabilité dans la conception des architectures.

Kmelia présente certains inconvénients tels que la vérification manuelle des assemblages de composants en plus Kmelia ne décrit pas les propriétés non fonctionnelles des composants.

VI. Conclusion

Ce chapitre a présenté un état de l'art sur l'approche à composant. Dans la première section nous avons donné les concepts proposés dans la littérature pour la notion de composant. La section suivante présente une description des deux cycles de vie de développent à base de composants : Design for reuse et Design by reuse. Enfin, une étude de quelques langages de descriptions d'architecture a été présentée. A travers cette dernière nous avons constaté que les ADLs existants ne supportent pas l'aspect non fonctionnel qui fait l'objectif du chapitre suivant.

CHAPITRE II :

QUALITE DE LOGICIEL

SOMMAIRE

I. Introduction

La recherche de la qualité est un des aspects les plus constants dans l'histoire de l'humanité. L'ère industrielle donne un coup d'accélérateur à la qualité en introduisant des notions, comme la rentabilité, le contrôle a posteriori….

Le début du XXe siècle est celui des efforts de normalisation de la qualité et de la création des organismes afférant. Il est marqué par l'émergence des gourous (Deming, Juran, Crosby, Taguchi, Isikawa, Shingo, Feigenbaum, etc.) qui posent les bases de la qualité.

La Qualité ne dispose pas d'une définition consensuelle qui conviendrait à tous les domaines auxquels elle s'applique. Quelque soit le contexte, la qualité recouvre des propriétés non fonctionnelles d'une entité informatique. Le niveau de qualité est caractérisé par les "valeurs" particulières des propriétés non fonctionnelles déterminées par les types de gestion choisis. Maintenir le niveau de qualité stable malgré les variations d'environnement est l'un des grands défis des recherches d'aujourd'hui.

II. Historique [14]:

1943-1970 :

ENAC, premier ordinateur de l'histoire qu'est apparu le premier bogue informatique.

Cause : un insecte qui s'était glissé dans les circuits de l'ENAC.

Pour les ingénieurs travaillant sur l'ENAC, l'activité de dépannage revenait à chercher des insectes qui se seraient infiltrés dans les circuits de la grosse machine, d'où le terme déboguer. Progressivement, les ordinateurs n'ont pas pris autant d'espace que l'ENAC et ont commencé à pénétrer les domaines sensibles, tels que l'espace, la défense, la recherche, etc.

- **En 1962** la fusée Mariner1 s'est écrasée **.**

Cause : erreur d'interprétation d'une formule par le langage Fortran.

Les erreurs et pannes sur les logiciels sont pendant cette période liées aux langages de programmation et à ceux qui les conçoivent. Pour des organisations sensibles comme la NASA, les questions se posent sur la façon de produire des logiciels fiables sans pannes ni de problème de qualité. Comment définir cette qualité suivant les besoins et les attentes ? Durant cette période il n'existe pas à proprement parler de modélisation de la qualité.

- **Fin des années 1970 :**

- Halstead a proposé le premier modèle de la qualité orienté logiciel, en introduisant des métriques pour mesurer les efforts et les coûts des logiciels en termes de ressources. Cependant son modèle ne dégage pas la qualité logicielle en termes de propriétés à respecter. Ce modèle est orienté mesure, ce qui limite une utilisation beaucoup plus large.
- Les travaux de Mc-Call et ceux de Boehm consacrent la naissance de premiers modèles de qualité à proprement parler. De ces deux modèles vont dériver d'autres modèles. Ces modèles ont été mis en œuvre pour répondre aux besoins de grandes organisations comme la NASA et General Electric.

1980-1996 :

- Au début des années 80 Technical Comity 176 (TC 176) est crée . Il a en charge de la normalisation dans le domaine de la qualité.
- En 1987, l'ISO lancera la série des normes ISO-9000. Des modèles adaptés à l'ingénierie du logiciel sont lancés, ce sont les normes telles que ISO-9126, ISO 9001/9000-3.

III. Définitions :

1. **La qualité :** Il y a plusieurs définitions de la qualité. Nous pouvons citer
    - La qualité est la conformité à des spécifications ou exigences [15].
    - Pour l'ASQ (American Society for Quality), la qualité est un terme subjectif pour lequel chaque personne a sa propre définition. Elle est définie comme les caractéristiques des produits ou services et concerne leur capacité à satisfaire des besoins donnés. Elle est également définie comme un produit présentant une absence de déficiences.
    - Ensemble des caractéristiques d'une entité qui lui confèrent l'aptitude à satisfaire des besoins exprimés et implicites (ISO 8402).
    - Obtention de la satisfaction durable du client, en répondant à ses besoins et attentes, au sein d'un organisme s'engageant à améliorer constamment son rendement et son efficacité (ISO9000).

2. **La Qualité de Service :** c'est l'effet global procuré par la qualité de fonctionnement d'un service qui détermine le degré de satisfaction de l'utilisateur [15] .

3. **Standard :** Ensemble de recommandations développées et préconisées par un groupe représentatif d'utilisateurs.

4. **Norme** ***(****Définition officielle ISO) :* Document établi par un consensus et approuvé par un organisme reconnu, qui fournit, pour des usages communs et repérés, des règles, des lignes directrices ou des caractéristiques, pour des activités où leurs résultats, garantissant un niveau d'ordre optimal dans un contexte donné.

IV. Différent types et niveaux de qualité

En général; il existe une double vision pour la qualité (cf. Figure II.1) :

- Vision du développeur (organisme).
- Vision du client.

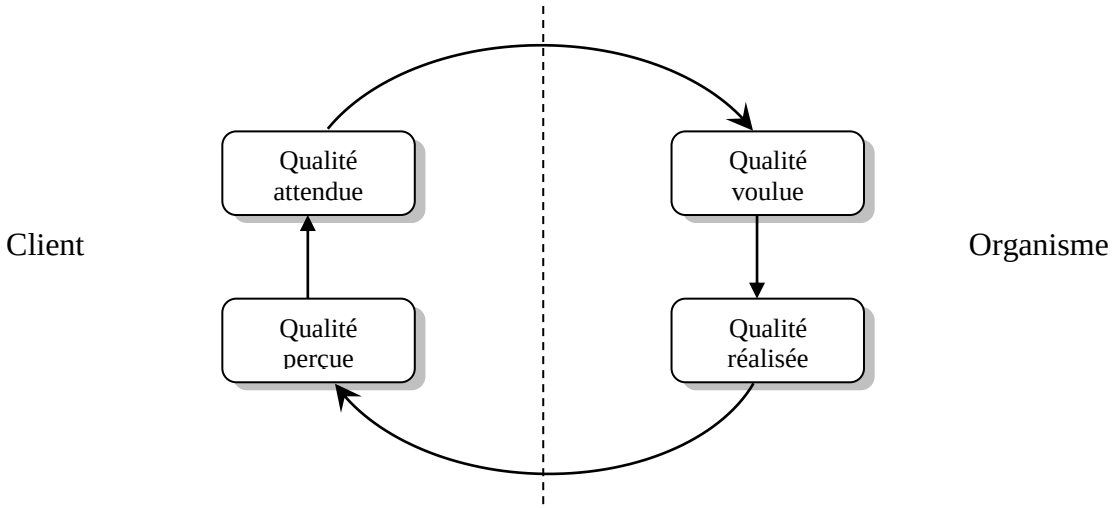


Figure II.1 : Différent type de qualité

Du point de vue de l'organisme, nous distinguons **[16]** :

- **La qualité voulue** : caractéristique la qualité que l'organisme souhaite atteindre pour répondre à la qualité attendue. C'est la prestation qu'il veut fournir à ses clients.
- **La qualité réalisée** : caractéristique la qualité réellement réalisée par l'organisme.

Du point de vue du client, nous distinguons **[16]**:

- **La qualité attendue** : caractéristique la qualité souhaitée par les clients, c'est-à-dire la réponse à leurs besoins et attentes. Cela signifie que les conditions d'obtention de ce niveau de qualité ont été prédéfinies.
- **La qualité perçue** : caractéristique la qualité ressentie, de façon plus ou moins confuse par le client à partir de ses propres besoins et attentes. Elle est l'expression de sa satisfaction.

L'organisme évalue le degré de satisfaction de ses clients en mettant en place des outils de mesure de la satisfaction des clients et de la performance du service rendu.

Selon les différents points de vue de l'ODP (Open Distributed Processing) décrits dans [4], le niveau de QoS sera différent :

- Du point de vue de l'entreprise, la QoS est subjective et non formelle : elle est orientée vers la spécification des utilisateurs par exemple : «le son doit être clair et net quelles que soient les conditions.» ;
- Du point de vue de l'information, la QoS est objective. Les besoins de QoS sont dérivés de la spécification subjective de la qualité de service. Cette spécification est souvent indépendante de l'application, par exemple, une connexion réseau à 100 Mb/s.
- Du point de vue calculatoire, les objets sont identifiés et les caractéristiques de QoS de l'application sont souvent décrites en terme de qualité de médias et de relations entre ceux-ci, par exemple le rafraîchissement des images peut être exprimé en images/s.
- Du point de vue ingénierie, la QoS peut être basée soit sur le système, soit sur le réseau.
    1. Sur le système, c'est la description des besoins de QoS du système d'opérations, par exemple la taille du buffer ou la mémoire en Mb;
    2. Sur le réseau, c'est la description des besoins de QoS du réseau, par exemple le débit en Mbit/s.

- Du point de vue technologique, les mécanismes de QoS utilisés pour implanter le système sont spécifiés, par exemple le format vidéo pour l'application sera le format PAL.

V. Les aspects standards de la qualité

- **Facteurs de qualité :** caractéristique du logiciel qui contribue à sa qualité.
  Ils concernent les caractéristiques d'utilisation liées à :
  - l'environnement d'exploitation.
  - l'environnement de suivi et de maintenance.

  Les facteurs traduisent la vision **externe** que peut en avoir le demandeur.

- **Critères de qualité :** Attribut du logiciel par l'intermédiaire duquel un facteur peut être évalué **.**Ils sont :
  - Orientés réalisateur.
  - Les composantes des facteurs de qualité.
  - Reliés à des métriques.

  Les critères de qualités concernent les caractéristiques d'utilisation en fonction d'une vision **interne** (structure du logiciel).

- **Métriques de qualité:** Les critères de qualité sont reliés à des métriques qui sont des mesures à posteriori.
  Il existe trois types de métrique pour mesurer les critères [17]:

  1.Présence : Cette métrique identifie, qu'un attribut soit présent dans un composant ou non. elle est décrite par une valeur de type booléen.

  2.dénombrement : Cette métrique est employée pour indiquer des valeurs exactes. Elle est décrite par une variable de type entier.

  3.Ratio : Cette métrique est employée pour décrire des pourcentages.

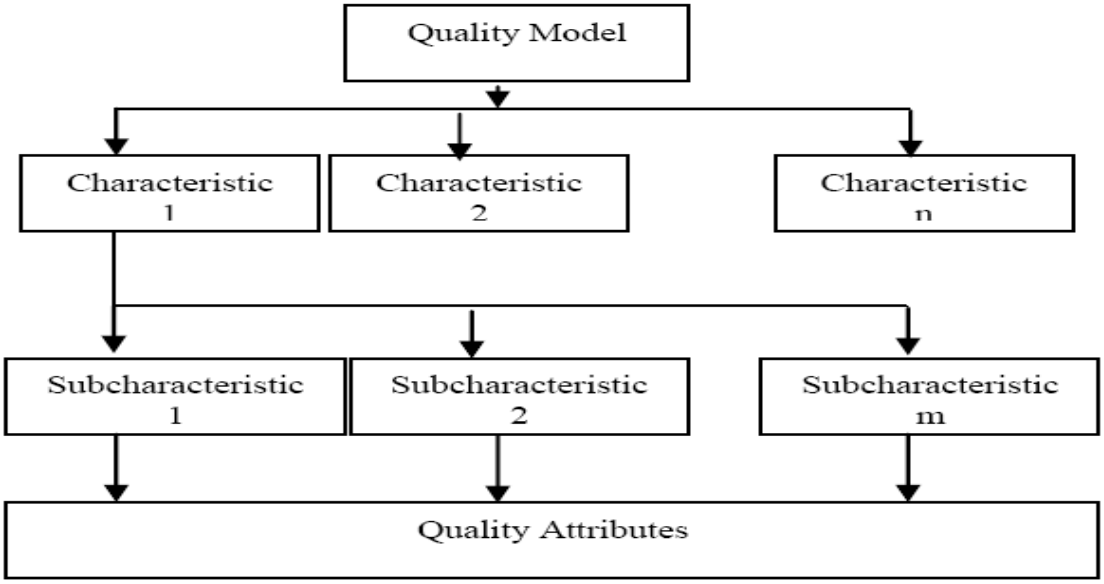


Figure II.2: Les aspects standards de la qualité

VI. Modèles de qualité

Un modèle de qualité est défini comme suit: ". L'ensemble des caractéristiques et les relations entre eux qui fournissent la base pour la spécification et l'évaluation des exigences de qualité» [18].

Selon [19], il ya plus de 300 standards élaborés et maintenues par plus de 50 organisations différentes. Le modèle le plus populaire est ISO / IEC 9126-1 [20], qui spécifie les exigences pour le système de gestion de la qualité au sein d'une organisation. Ce standard est un modèle de qualité logicielle générique et peut être appliqué à tout produit logiciel en l'adaptant à un but précis.

Le premier modèle de la qualité considéré comme un standard a été développé et publié par l'Organisation internationale de normalisation en 1991 (ISO 9126 [21]). Dix ans plus tard une nouvelle initiative internationale « *Software product QUality Requirements and Evaluation (SQuaRE)* [22] » a été mise en place Cette nouvelle approche est considérée comme une nouvelle génération de modèles de qualité de logiciel.

VI.1. Le Standard ISO 9162

Le tableau suivant représente le modèle de qualité défini dans [23].

Caractéristiques (Facteurs)	Sous_Caractéristiques (Critères)
Fonctionnalité	Exactitude , Sécurité, Aptitude, Interopérabilité, Conformité
Fiabilité	Maturité, Tolérance de fautes, Récupération
Utilisabilité	Configurabilité , Compréhension ,Facilité d'exploitation
Efficacité	Comportement de temps , Comportement de Ressource , Scalabilité
Maintenabilté	Facilité d'analyse, Facilité de modification , Stabilité, Facilité de test.
Portabilité	Déployabilité , Adaptabilité, Réutilisabilité

Tableau II.1 : le standard de qualité ISO 9126[23]

VI.1.1. Les facteurs de qualité

Le standard ISO 9126 spécifié six caractéristiques de qualité : Functionality, Reliability, Usability, Efficiency, Maintainability, Portability. Ces caractéristiques sont définies comme suit :

1. **Capacité fonctionnelle (Functionality) :** la capacité d'une application à délivrer les fonctions répondant aux besoins explicites et implicites dans un contexte donné.

2. **Fiabilité (Reliability)** : La capacité d'une application à maintenir le niveau de service spécifié dans des conditions spécifiées.

3. **Facilité d'utilisation (Usability)** : la capacité d'un logiciel à être compris, appris et attractif pour les utilisateurs, selon des circonstances spécifiques d'utilisation.

4. **Rendement (Efficiency) :** la capacité d'une application à fournir le niveau attendu de performances, en fonction des ressources utilisées selon des conditions fixées.

5. Maintenabilité (Maintainability) : la capacité d'une application à être modifiée. Les modifications incluent les corrections, les améliorations et l'adaptation lors d'un changement d'environnement, d'exigences ou de spécifications fonctionnelles.

6. Portabilité (Portability) : la capacité d'une application à basculer d'un environnement à un autre.

VI.1.2. Les critères de qualité

Chacun des facteurs définit ci-dessus est présenté par un ensemble de critères :

1. Capacité fonctionnelle :

- **Exactitude (Accuracy)** : La capacité d'une application à fournir les résultats attendus avec le degré d'exactitude attendue.
- **Aptitude (Suitability) :** La capacité d'une application à fournir les fonctions appropriées pour répondre aux tâches spécifiques et aux besoins de l'utilisateur.
- **Interopérabilité (Interoperability)** : La capacité d'une application à interagir avec un ou plusieurs systèmes spécifiés.
- **Sécurité (Security)** : La capacité d'une application à protéger les informations et les données de manière à ce que les personnes non autorisées ne puissent lire ou modifier celles-ci tandis que les personnes autorisées puissent y avoir accès.
- **Conformité (Compliance)** : La conformité d'une application par rapports aux standards, conventions et règles définies dans le cahier des charges fonctionnel.

2. Fiabilité:

- **Maturité (Maturity):** La capacité d'une application à éviter les défaillances résultant de défauts dans le logiciel.
- **Tolérance aux pannes (Fault tolerance)** : La capacité d'une application à maintenir un niveau spécifié de performances en cas de pannes ou de violation de son interface.
- **Facilité de récupération (Recoverability)** : La capacité d'une application à rétablir son niveau de performances et à récupérer les données affectées en cas de panne.

3. Facilité d'utilisation:

- **Facilité de compréhension (Understandability)** : la capacité d'une application à être comprise et la facilité pour un utilisateur de comprendre comment l'application doit être utilisée pour une tâche particulière et dans des conditions d'utilisation données.
- **Facilité d'apprentissage (Learnability)** : la facilité pour un utilisateur à apprendre à utiliser l'application.
- **Facilité d'exploitation (Operability) :** la facilité pour un utilisateur à contrôler l'application.

4. **Rendement:**
   - **Comportement temporel (Time behaviour) :** la capacité d'une application à fournir la réponse appropriée dans un délai de traitement défini.
   - **Utilisation des ressources:** Attributs du logiciel portant sur la quantité de ressources utilisées et sur la durée de leur utilisation lorsqu´il exécute sa fonction.

5. **Maintenabilité:**
   - **Facilité d'analyse (Analysability) :** la capacité d'une application à être diagnostiquée en cas de pannes et de défaillance, ou lors de besoin de modification du logiciel.
   - **Facilité de modification (Changeability) :** la capacité d'une application à faciliter l'implémentation des modifications.
   - **Stabilité (Stability) :** la capacité d'une application à éviter les effets non souhaités lors de sa modification.
   - **Facilité de test (Testability)** : Attributs du logiciel portant sur l´effort nécessaire pour valider le logiciel modifié.

6. **Portabilité :**
   - **Facilité d'adaptation (Adaptability)** : la capacité d'une application à être adaptée à différents environnements sans appliquer d'actions autres que celles prévues pour cela.
   - **Facilité à l´installation (Installability)** : Attributs du logiciel portant sur l'effort nécessaire pour installer le logiciel dans un environnement donné.

- **Conformité (Compliance)** : Attributs du logiciel permettant à celui-ci de se conformer aux normes ou conventions ayant trait à la portabilité.
- **Interchangeabilité (Replaceability)** : la capacité d'une application à être utilisée à la place d'une autre pour le même but dans le même environnement.

VI.1. 3. Les attributs de qualité

Conformément au standard "ISO/IEC 9126". Les attributs constituent le niveau le plus bas de la hiérarchie et sont mesurables par des métriques. Dans [24] toutes les métriques de qualité sont définies.

Par exemple les métriques associées aux attributs auditabilité et contrôlabilité du critère sécurité sont définies ainsi :

Nom	but	Formule de mesure	Interprétation de la valeur mesurée	Type de mesure
Auditabilité	Évaluer le taux d'accès que le système a enregistrés dans la base d'accès.	X= A / B A= nombre "de tentatives d'accès au système et aux données » enregistré dans la base d'accès. B= nombre "de tentatives d'accès au système et aux données » faites durant l'évaluation.	0<=X<=1 Plus proche à 1 est le meilleur.	A : dénombrement B : dénombrement
Contrôlabilité	Compte le nombre d'opérations illégales détectées	X= A / B A= nombre d'opérations illégales détectées. B= nombre d'opérations illégales spécifiées	0<=X<=1 Plus proche à 1 est le meilleur.	A : dénombrement B : dénombrement

VII. Spécification de la qualité [4]

De nombreux travaux sur la spécification de la QOS ont donné le jour à des langages supportant la QoS. Certains sont spécifiques à une catégorie de QoS comme la fiabilité où la QoS relative au temps ; alors que d'autres sont plus génériques et couvrent tous les domaines de QoS. Ce paragraphe présente un aperçu des principaux travaux relatifs à la QoS.

VII.1. QuO : Quality of service for Objects

QuO est une architecture qui supporte la QoS d'un objet CORBA et établit un rapprochement conceptuel entre les garanties du réseau et les besoins de l'application. QuO se focalise sur les connections entre les objets distribués. Ces connections ont un comportement qui adapte les applications aux conditions considérées. Chaque propriété du système est une dimension de QoS et QuO divise cet espace multidimensionnel en régions définies par des prédicats provenant des propriétés du système. Deux niveaux de régions sont définis :

- Le niveau de négociation qui définit les conditions du système avec lesquelles le client et le serveur essaient de travailler.
- Le niveau réel : dans les régions de négociation, il peut y avoir plusieurs régions réelles qui sont les conditions mesurées des systèmes.

QuO se focalise sur les connections entre clients et serveurs et supporte seulement les dimensions numériques et mesurables de QoS. Cette contrainte restreint les types de QoS qui peuvent être définis. La spécification des caractéristiques subjectives telles que la sémantique des pannes (halt, initial state, rolled back), la productivité ou la satisfaction ne peuvent pas être décrite naturellement.

VII.2. QML : Quality of service Modeling Language

C'est un langage de spécification de QoS qui sépare la spécification de la QoS de la spécification des aspects fonctionnels. QML a trois principaux mécanismes d'abstraction : contract type, contract et profile. Bien que les profiles définissent les QoS offertes par le service fournisseur, il n'est pas possible de spécifier ce que le service pourra réellement fournir en regard des conditions environnementales.

VII.3. QDL : Quality of service Definition Language

QDL définit les relations de QoS en définissant des objets de QoS. Un objet de QoS contient une attente et une obligation de QoS (QoS offerte). Une obligation de QoS contient un nombre de propriétés classées soit simples soit complexes. Une propriété simple est une paire nom-valeur alors qu'une propriété complexe dépend d'autres propriétés provenant d'autres objets de QoS. Un besoin de QoS est spécifié comme une contrainte sur les propriétés des autres objets de QoS. QDL utilise OCL pour spécifier les relations de QoS.

IX. Conclusion

Le but de ce chapitre était de présenter ce que cache le mot **qualité**. Les 4 premières sections présentent les concepts nécessaires pour l'étude de qualité. La section VI présente le Standard ISO 9162. Et enfin nous avons présenté les travaux réalisés pour la spécification de la QoS.

Après ces deux chapitres nous avons constaté qu'il ya une dichotomie entre l'aspect fonctionnel et non fonctionnel. Il n'existe pas un ADL qui supporte la description de la qualité de même il n'existe pas un langage de description de la qualité qui décrit les caractéristiques fonctionnels.

CHAPITRE III :

RECHERCHE ET SELECTION DE COMPOSANTS LOGICIEL

SOMMAIRE

I. Introduction

La réutilisation de composants logiciels est d'un intérêt majeur pour le développement logiciel. Cependant, rendre la réutilisabilité effective pose des problèmes tels que la recherche de composants.

Face à une bibliothèque contenant un grand nombre de composants, il est nécessaire d'avoir un mécanisme de recherche automatique qui permet de retrouver les composants les plus pertinents.

Dans ce chapitre nous entamons la problématique de recherche de composants logiciels. La section II donne un aperçu général sur le principe de la recherche de composant. La troisième section du chapitre présente les techniques de recherche de composants Ensuite, dans la quatrième section nous présentons un état de l'art sur les principaux processus de sélection de composants.

II. Principe de la recherche de composants

Selon [25]. Le problème de recherche de composants est formulé comme suit :

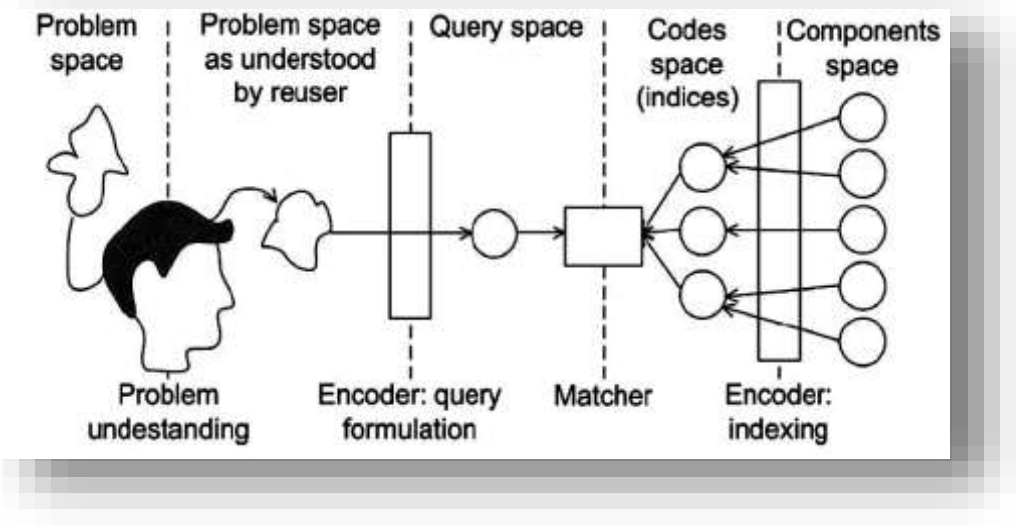


Figure III.1 : Modèle la recherche de composants [25]

Le modèle de recherche est décomposé en cinq espaces :

1. **Problem space :** ou espace de problème représente l'ensemble des besoins d'une application particulière, exprimés de manière informelle.

2. **Problem space as understood by reuser understanding :** après l'interprétation de chacun de ces problèmes par l'utilisateur (Problem understanding) , un espace des problèmes modifié se produit . Par exemple, l'ensemble des services qui peuvent répondre aux besoins exprimés dans l'espace précédent.
3. **Query space :** Pour chaque service identifié par l'utilisateur, il faut maintenant formuler une requête.
4. **Components space : l'espace** de composants ou la bibliothèque où la recherche sera effectuée.
5. **Codes space (indices) :** avant la recherche , il faut traduire les composants afin d'obtenir un ensemble de descriptions appel´e espace des indexes. En effet, chaque description doit être de format identique à celui de la requête afin que la comparaison puisse se faire.

III. Techniques de recherche de composants

III.1. Techniques de classification externe

Le principe de ces techniques est d'indexer les composants à partir d'une représentation abstraite.

III.1.1 Recherche 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.

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 [26].

III.1.2 Recherche par facette

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 [26].

L'approche de classification par facettes a été utilisée dans plusieurs systèmes de recherche de composants comme [27][28].
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 [5].

III.1.3 Utilisation de langage naturel

Dans [29] 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.
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 œuvre [5].

III.2. Techniques de classification 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.

III.2.1. Techniques d'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.

Par exemple dans [30] 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'´egalité 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.

A partir de cette définition, Zaremski et Wing 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 t_l (est le type d'une fonction appartenant à un composant dans une bibliothèque) et t_q (un type entré comme requête) :

1. t_l « **matche exactement** » t_q : s'ils sont égaux au renommage de variables prés. Par exemple t_l (x,x)→bool et t_q(y,y)→bool . Alors t_l match exactement t_q . Avec une substitution V[y/x] on bien V.t_l=$_T$ t_q.
2. t_l « **généralise** » t_q : si on peut retrouver t_q à partir de t_l via une séquence de substitutions de variables.
3. Dans le **matching spécialisé**, à l'inverse, c'est t_q qui généralise tl .
4. Il y a un **matching réordonné** quand il existe une permutation des paramètres du type t_l 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.

III.2.2. Appariement de spécifications

La spécification d'un composant informe non pas sur sa syntaxe, mais sur le son comportement. Ces spécifications peuvent être exprimées sous forme logique ou par des contrats (invariants, pré- et post-conditions) [26]. Les techniques de recherche de

composants par appariement de spécifications tentent de retrouver les composants de la base dont les spécifications correspondent à la spécification de la requête [5].

Ces techniques représentent généralement les composants logiciels selon le modèle suivant :

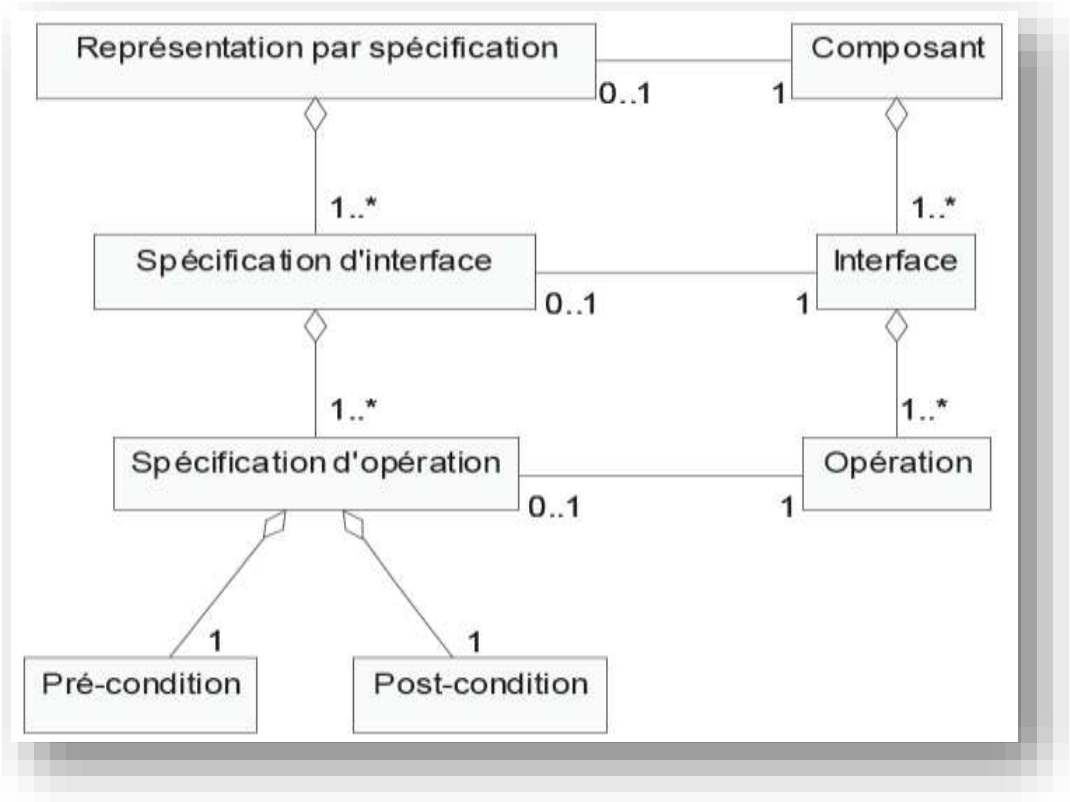


Figure III.2. Modèle de représentation par spécifications [54]

Parmi les travaux qui ont été proposés dans ce cadre :

- Le matching de spécification Zaremski et Wing [54]. C'est le prolongement de leurs travaux sur le matching de signature. A chaque opération on associe un ensemble de pré- et post-conditions qui représentent son comportement. Le matching de spécification est défini par la formule suivante :

  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 et *R*3 sont des connecteurs logiques tels que : $\Rightarrow$ implique. , $\Leftrightarrow$ équivaut, ou ^ et .

Ils représentent les requêtes et les composants par un ensemble de paires de pré et post conditions (une paire par méthode).

- Dans [55] Rollins et Wing présentent une base de composants utilisant une approche de classification par spécifications. Les composants sont spécifiés en langage λ-Prolog. Cette technique exploite les mécanismes d'inférence du langage λ-Prolog pour l'appariement des spécifications. Seuls les composants dont les spécifications sont pertinentes par rapport à la spécification de la requête sont sélectionnés.
- Enfin, Hemer et Lindsay proposent dans [56] une base de modules. Un module est un ensemble d'unités. Une unité peut être une procédure, une classe ou une structure de données, Si un développeur a besoin d'un module qui implante toutes les opérations sur les listes chaînées alors il doit spécifier toutes les primitives dont il a besoin sous la forme d'unités, puis il cherche dans la base, le module qui contient ces unités. Les auteurs proposent trois stratégies de recherche de modules :
    1. **ALL-match** : vérifie que toutes les unités de la requête correspondent à des unités distinctes du composant.
    2. **SOME-match** : relaxe le critère ALL-match en se contentant de vérifier qu'au moins un sous-ensemble non vide d'unités de la requête correspond à un sous-ensemble non vide d'unités du composant.
    3. Et Le critère **ONE-match** : vérifie qu'au moins une des unités de la requête correspond à une des unités du composant.

III.3. Techniques de recherche comportementale

Ces approches s'intéressent à l'aspect dynamique des composants en analysant leur comportement lors de l'exécution [5].

III.3.1. Approches par analyse des traces d'exécutions

Parmi les travaux qui se sont intéressés à cette approche, on peut citer ceux de Podgursky et Pierce [31] [32]. 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 [33], 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.

III.3.2. Approches par spécification comportementale

Dans [34] les auteurs 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 (coefficient de Jaccard's).

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,
- un outil de classification de spécifications,
- un gestionnaire de bases de composants
- et un outil de recherche de spécifications.

IV. Analyse multicritère

Les méthodes d'analyse multicritère ou, plus exactement, les méthodes d'aide multicritère à la décision sont des outils d'aide à la décision développé autour des années 60. Elles ont été introduites par Koopmans ainsi que par Kuhn et Tucker. Depuis, la prise de décision multicritères s'est développée en tant que domaine de recherche appliqué à de nombreux autres domaines [26].

MCDM(Multi-Criteria Decision Making) est une discipline de la recherche opérationnelle, laquelle traite des problèmes de décision en présence d'un certain nombre de critères[35]. Elle est utilisée pour porter un jugement comparatif entre des projets ou des mesures hétérogènes.

On distingue souvent trois types de problématique en aide à la décision [57]:

- la problématique du choix, qui consiste à vouloir choisir la ou les solutions considérées comme optimales pour le problème considéré;
- la problématique du classement (ranking), qui consiste à vouloir classer du premier au dernier toutes les solutions connues du problème considéré ;
- la problématique du tri (sorting), qui consiste à affecter les solutions à des catégories (ordonnées ou non).

IV.1.Concepts de base : [36][58]

Les méthodes MCDM se partagent un ensemble de concepts qui sont :

- **Le décideur** : est l'entité intervenant dans le processus de décision que les modèles mis en œuvre cherchent à éclairer, c'est l'entité pour le nom de qui, ou au compte de qui, l'aide à la décision s'exerce.
- **Alternatives :** Les alternatives représentent les différents choix d'action disponibles au décideur. Généralement, l'ensemble des solutions est supposé fini.
- **Attributs :** Chaque problème MCDM est associé à de multiples attributs qui représentent les différentes dimensions à partir desquelles les alternatives peuvent être vues. Ils peuvent être arrangés d'une façon hiérarchique. Ainsi, quelques attributs peuvent être des attributs majeurs. Chaque attribut majeur peut être associé à plusieurs sous-attributs. De façon similaire, chaque sous-attribut peut être associé à des sous-sous-attributs différents, et ainsi de suite.
- **Unités incommensurable :** Différents attributs peuvent être associés à différentes unités de mesure. Par exemple, dans le cas de l'achat d'une voiture d'occasion, les attributs « coût » et « kilométrage » peuvent être mesurés, respectivement, en termes d'euros et de milliers de kilomètres.
- **Poids :** La majorité des méthodes MCDM propose d'attribuer aux attributs des poids désignant leurs importances. Généralement, ces poids sont normalisés de telle sorte que leur somme doit être égale à *1*.
- **Matrice de Décision :** Un problème de MCDM peut être facilement exprimé sous forme d'une matrice. Une matrice de décision **M** est une matrice ($m \times n$) dans laquelle l'élément a_{ij} indique la valeur des attributs C_j correspond à l'alternative *Ai* (pour $i = 1, 2,3,..., m,$ et $j = 1, 2,3,..., n$). Il est également supposé que le décideur ait déterminé les poids de la performance relative des critères de décision.

Attributs / Alternatives	C_1	C_2	C_3	…..	C_n
	Poids C_1	**Poids C_2**	**Poids C_3**	……	**Poids C_n**
A_1	a_{11}				
A_2			a_{23}		
……					
A_m					

Figure III.3 : Représentation d'une matrice de décision

IV.2. Les Méthodes MCDM

Il existe trois grandes familles de méthodes, à savoir : les méthodes basées sur la théorie de l'utilité multi-attributs, les méthodes de surclassement et les méthodes interactives.

IV.2.1 WSM (Weighted Scoring Method)

- A Chaque critère d'évaluation on associe un poids qui représente son importance.
- A chaque couple critère-candidat on associe un score qui représente la capacité du candidat à remplir le critère.
- le candidat qui possède le score total le plus élevé est considéré comme le meilleur.
- La formule WSM la plus commune utilise l'addition comme fonction d'agrégation :

$$score_c = \Sigma_{j=1}^{n}(weight_j * score_{cj})$$

Evaluation criteria	Weight	Score Comp. A	Score Comp. B	Score Comp. C
Usability	2.0	3.0	3.0	3.0
Compatibility	4.0	1.0	5.0	2.0
Cost	3.0	3.0	5.0	1.0
Functionality	5.0	4.0	4.0	3.0
Security	4.0	1.0	2.0	5.0
Technical support	5.0	2.0	5.0	3.0
Total score	XXXX	**53.0**	**94.0**	**67.0**

Par exemple, pour le critère « sécurité », l'utilisateur a considéré que le candidat *A* n'était pas du tout sûr, que le candidat *B* l'était peu, et que le candidat *C* avait le niveau maximal.

Par_exemple, le score total du candidat *C* est égal à : 2 * 3 + 4 * 2 + 3 * 1 + 5 * 3 + 4 * 5 + 5 * 3 = 67). On s'aperçoit que le candidat *B* possède le meilleur score.

Evaluation :

L'inconvénient majeur de cette technique réside dans la difficulté de définir un ensemble de critères d'´evaluation pertinents, puis de leur assigner les bons poids, surtout quand ils sont nombreux. De plus, l'´evaluation des scores locaux et l'assignation des poids restent manuelles, ce qui, rend cette technique fastidieuse face à un nombre élevé de critères et de candidats.

IV.2.2 AHP : Analytic Hierarchy Process

Par rapport au WSM , AHP aide à décrire le besoin de manière organisée. Tout d'abord, il faut définir l'objectif principal à atteindre pour prendre une décision sur la sélection d'un produit candidat. A partir de la, on décompose cet objectif en un arbre hiérarchique de critères et de sous critères d'évaluation, dont les feuilles sont les candidats à évaluer. Plus on descend dans l'arbre, plus les critères d'évaluation se précisent.

AHP est donc un moyen de définir et de hiérarchiser les différents critères d'évaluation avec précision. Cela permet d'utiliser WSM en assignant des poids cohérents. En effet, à l'intérieur de chaque nœud, les poids des nœuds fils sont bornés les uns par rapport aux autres(somme des poids égale à 1) .

Evaluation :

Il est difficile d'employer AHP tel quel pour des systèmes logiciels complexes, qui comporteraient un nombre élevé de critères et de sous-critères, pour lesquels il y aurait une vaste bibliothèque de candidats à évaluer.

IV.2.3 MAGIQ: Multi-Attribute Global Inference of Quality [37][38] :

MAGIQ a été initialement développé pour valider les résultats la méthode (AHP). Bien MAGIQ n'a pas été soumis à des recherches approfondies, la technique s'est avérée très utile dans la pratique.

La technique MAGIQ utilise le concept ROC (Rank Order Centroids). Les centroides de classement constituent un moyen de convertir des rangs (1ier, 2eme, 3eme) en notes ou pondérations qui sont des valeurs numériques. Si n est le nombre des attributs, la pondération de l'attribut K est :

$$W(Ak) = \frac{\left(\sum_{i=k}^{n} \frac{1}{i}\right)}{n}$$

C'est-à-dire que la production de la qualité globale d'un système logiciel implique :

1. Le classement des critères (et des attributs) selon leur importance.
2. La conversion des rangs en pondération à l'aide de ROC.

V. Les processus de sélection de composants logiciels

V.1. OTSO (Off-The-Shelf-Option) [40]

Ce processus est mis au point par J. Kontio. La figure 2 montre les principales phases du processus OTSO ainsi que les paramètres et données utilisées.

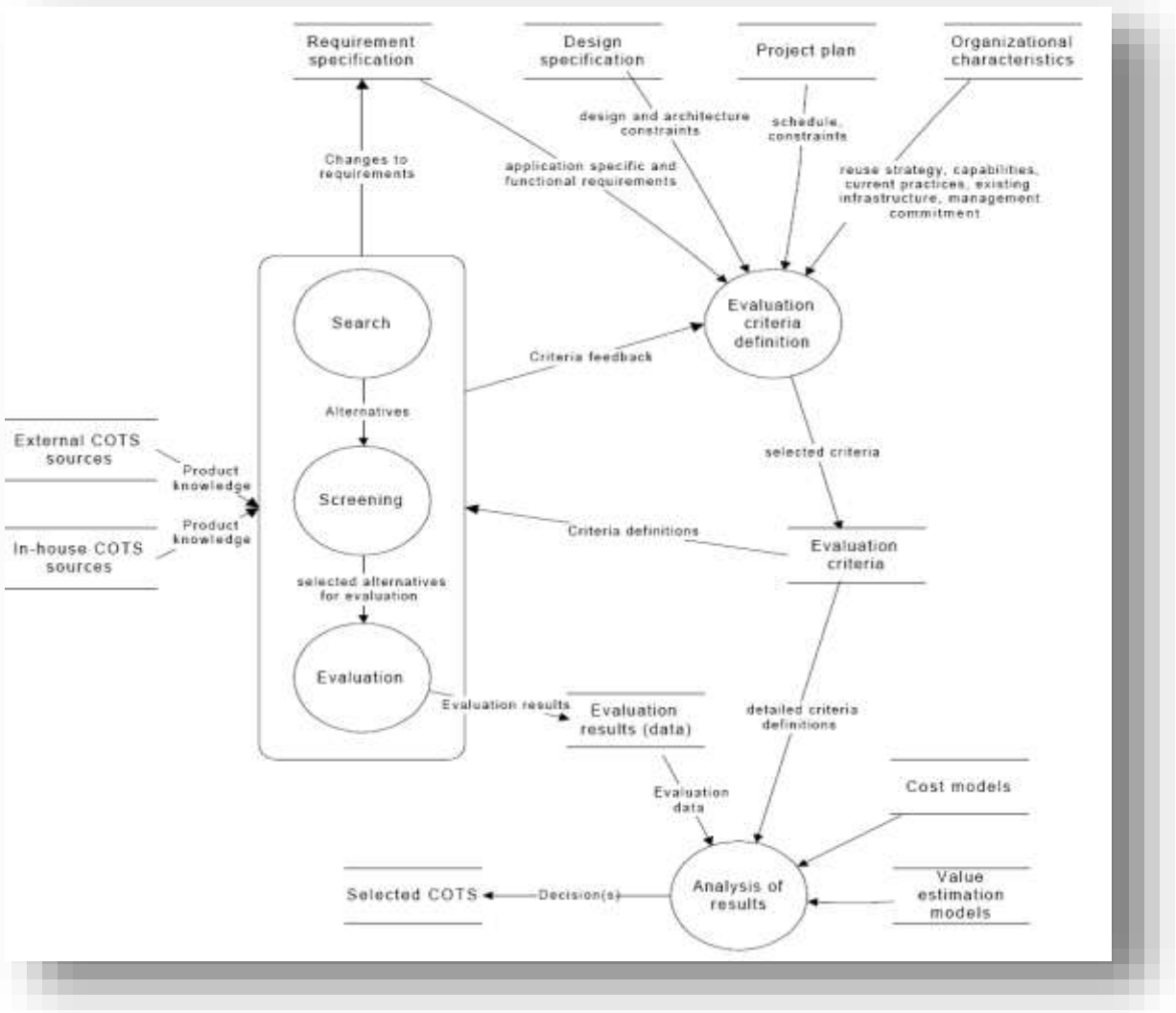


Figure III.4. : Processus de sélection OTSO [40]

1. **Phase N°1 : Définition des critères d'évaluation (Evaluation criteria definition**) La première étape consiste à définir les critères d'´evaluation selon quatre paramètres liés à l'application dans laquelle on va intégrer le COTS candidat : Le premier paramètre (**Requirement specification**) concerne la spécification des besoins de l'application, cela inclut en particulier les besoins fonctionnels. Le deuxième paramètre (**Design specification)** concerne la spécification de l'architecture globale de l'application. Le troisième **(Organisational characteristics)** et le quatrième paramètre **(Project plan)** sont les contraintes d'organisation et de projet liées à son développement.

2. **Phase N°2 :Recharche (Search)**, la recherche dans le processus de sélection consiste à identifier les candidats potentiels.
3. **Phase N°3 :Pré-sélection (Screening)** consiste alors à filtrer les candidats les plus pertinents au moyen de critères de base et de certaines informations sur ces candidats (COTS Sources).
4. **Phase N° 4 : Evaluation**, consiste à évaluer en détail chaque candidat au moyen de critères plus précis.
5. **Phase N°5 : Analyse des résultats (Analyse of results),** La phase finale consiste à analyser les résultats de ces évaluations en fonction des critères précédemment définis.

Un inconvénient majeur du processus OTSO est qu'elle ne propose rien pour spécifier les besoins. De plus, ce sont surtout les besoins fonctionnels de l'application qui sont pris en compte, au détriment des besoins non-fonctionnels [41].

V.2. PORE (Procurement-Oriented Requirements Engineering) [42]

Maiden et Ncube ont mis au point la méthode PORE. A la base, PORE définit trois types de besoins : les besoins fonctionnels principaux, les besoins fonctionnels secondaires, et : les besoins non-fonctionnels. Afin de satisfaire ces trois types de besoins, PORE utilise un processus itératif, décomposé en ci processus génériques :

1. **Identification des produits (Identify Product) :** consiste à repérer les candidats potentiels au moyen d'une étude de marché ou d'une expertise.
2. **Acquisition de l'information (Acquire Information) :** ce processus consiste à obtenir des informations sur les besoins de l'utilisateur (**customer requirements**) et sur les candidats (**product features**). L'acquisition de l'information se fait au moyen de cas d'utilisation et de techniques d'ingénierie des connaissances.
3. **Analyse de l'information acquise (Analyse Acquired Information) :** consiste à analyser l'information acquise pour produire des données exploitables dans le processus suivant.

4. **Prise de décision (Make Decision) :** le but de ce processus est déterminer si les candidats satisfont les besoins ou non et ce en utilisant les techniques de prise de décision multi-critères (MCDM).
5. **Sélection de produit (Select Product)** : la sélection de produit consiste à rejeter les candidats qui ont été évalués comme non satisfaisants par le processus précédent.

Une fois que la sélection est effectuée, deux choix sont possibles :

1. Soit on recommence l'opération en retournant au premier ou au second processus pour détailler les besoins et réduire la liste des candidats,
2. Soit on arrête la sélection quand on estime que la requête est satisfaite. Chaque processus peut être répété plusieurs fois dans le cas où l'information est insuffisante.

Le processus PORE présente certains inconvénients, par il est parfois difficile de savoir comment arrêter l'itération [43] . Et dans [44], les auteurs trouvent peu claire la manière d'utiliser les besoins dans le processus d'´evaluation, ainsi que la manière d'´eliminer les candidats.

V.3.CRE (COTS-based Requirements Engineering) [44]

CRE est une approche semblable à PORE, elle possède quatre phases itératives :

1. **Phase N° 1 : Identification** : permet de définir les objectifs en se basant sur : besoins des utilisateurs, architecture de l'application, objectifs et restrictions du projet, disponibilité des composants et infrastructure de l'organisation.

2. **Phase N°2 : Description :** Dans cette phase, les critères d'évaluation doivent être élaborés en détail en insistant sur le rôle des NFR (Non Fonctionnel Requirements). La méthode CRE utilise le NFR Framework pour la représentation et l'analyse des besoins non fonctionnels. Ce Framework est une approche orientée processus où les NFR sont explicitement représentés comme des objectifs à atteindre. Ces objectifs seront ensuite décomposés en sous-objectifs en utilisant une structure arborescente et des opérateurs logiques reliant le nœud père avec ses fils.

3. **Phase N°3 : Evaluation :** la décision de sélection d'un produit particulier COTS est basé sur le coût estimatif par rapport à l'analyse des avantages de chaque solution de rechange COTS. Le processus CRE suggère l'utilisation de modèles de coûts tels que les COnstructive COTS (COCOTS).

VI. Conclusion

Dans ce chapitre, nous avons présenté les différents mécanismes de recherche de et de sélection de composants logiciels. Les processus de sélection existants ont en commun un certain nombre de phases qui sont présentées dans [60] sous l'appellation de « processus de sélection général » (GCS pour General COTS Selection) : 1- Définir les critères d'évaluation basés sur les exigences et les contraintes du développeur;-*2*-: Rechercher les composants à partir des référentiels,-3- Filtrer les résultats de recherche basée sur les exigences. Cela permet la définition d'une liste des composants candidats prometteurs qui doivent être évalués de manière plus détaillée, -*4*-: Évaluer les composants candidats de la liste restreinte.

Après l'étude de ces processus, nous avons remarqué que :

- La majorité des méthodes de sélection n'ont pas traités suffisamment les besoins non-fonctionnels.
- En plus la notion de l'assemblage est absente : Les processus de sélection ne traitent pas le cas où le service sera fourni par un assemblage de composants.

PARTIE 2 :

CONTRIBUTION

CHAPITRE IV :

APPROCHE PROPOSEE

SOMMAIRE

I. Introduction

CBSE représente un nouveau paradigme de développement: « construire un système par l'assemblage des composants préexistants » cf. figure IV.1

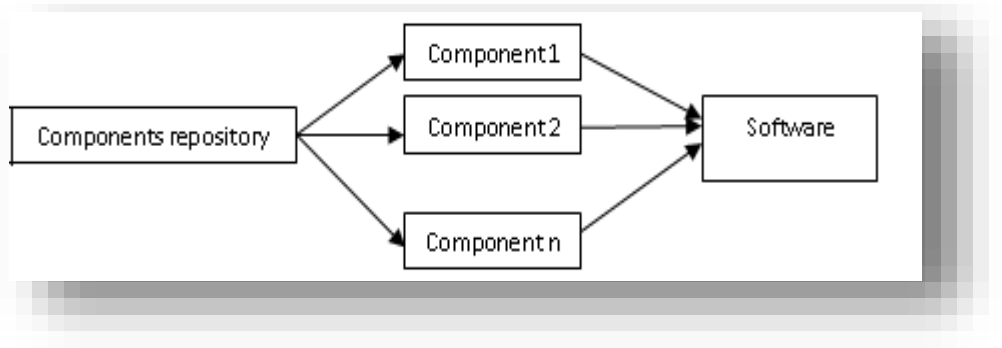


Figure IV.1 : Développement à base de composant [61]

Pour un besoin fonctionnel donné, nous pouvons trouver plusieurs composants assurant les mêmes services. Il est donc possible d'avoir plusieurs compositions (systèmes candidats). Le but de cette thèse est de répondre à la question suivante:

«Parmi plusieurs systèmes équivalents (assemblage), quel est le plus approprié ? ».

Comme il est présenté dans la figure IV.2, notre travail touche plusieurs domaines dans l'ingénierie à base de composant. Premièrement La partie évaluation : le but de ce module est d'évaluer les systèmes logiciels (composants ou composites). Cette étape nécessite l'existence d'une base de données qui doit fournir toutes les informations nécessaires pour permettre une évaluation pertinente. Pour cela nous avons proposé des métadonnées pour la description des composants et des assemblages. Donc en premier lieu les composants doivent être enregistrés dans la base « l'indexation ».

L'entrée principale de cette phase est le fichier descriptif (Métadonnée). Ensuite si un utilisateur demande un service, le système interroge la base de composants et renvoie les composants qui peuvent assurer le service demandé. Deux cas peuvent être distingués : le service sera fourni par un composant ou par un assemblage. Dans les deux cas les candidats passent par l'évaluation pour sélectionner le meilleur.

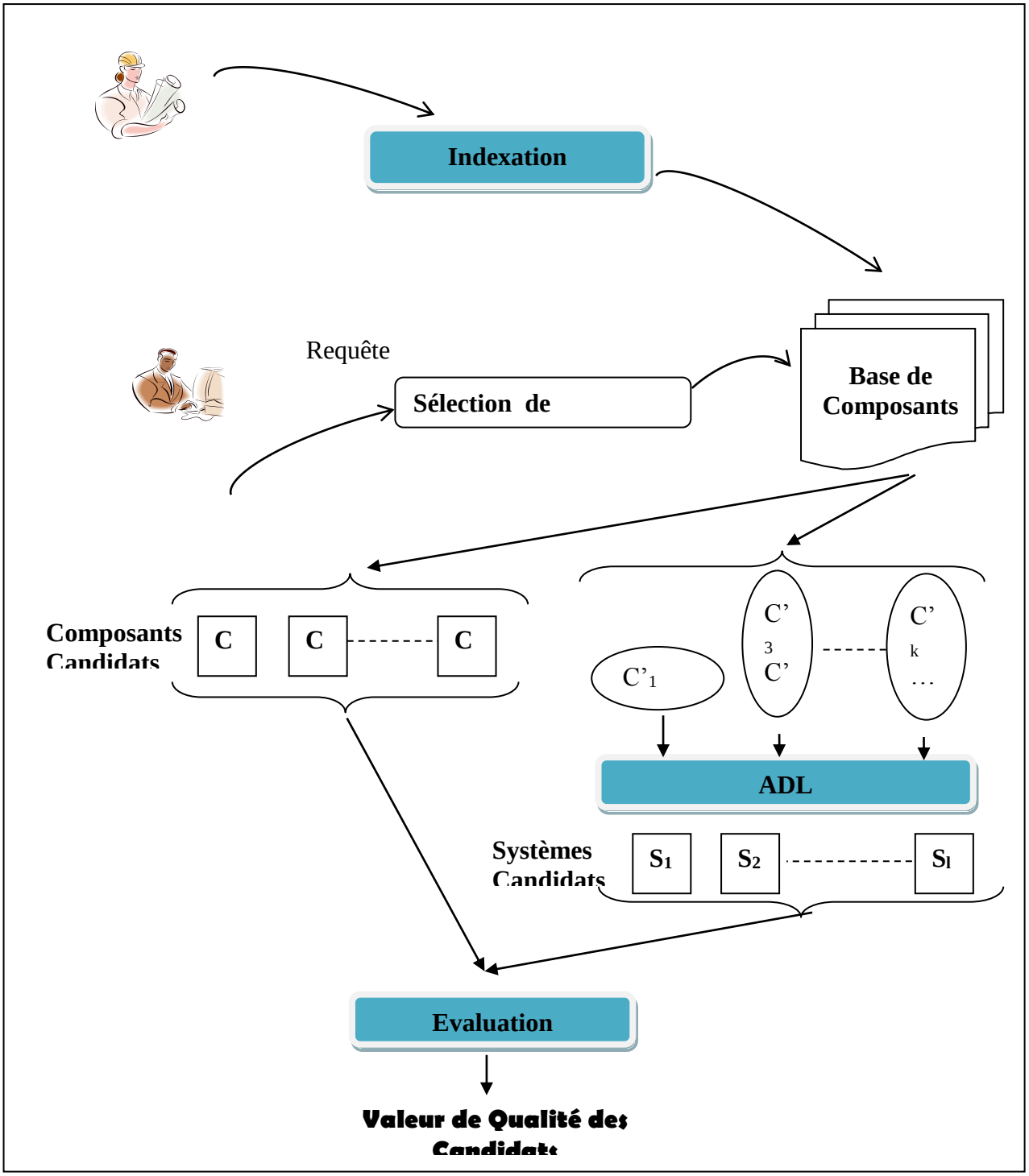


Figure IV.2 : Indexation, Recherche et évaluation de composants et Assemblage

Notre contribution est divisée en trois sections :

1. La première s'intéresse à la description des composants et des assemblages. Dans cette dernière, nous présentons les métadonnées proposées.

2. Dans la deuxième nous entamons l'évaluation. Nous donnons une description détaillée des processus d'évaluation proposés.

3. Et dernièrement nous présentons le Framework développé.

Software Component Metadata (SCM)

I. Introduction

L'étude des différents modèles de description de composants logiciels nous a permis de constater leurs insuffisances pour décrire les différents aspects des composants logiciels.

Afin de permettre la sélection et la comparaison des composants, quelles sont les informations nécessaires pour les décrire?

Pour répondre à cette question nous proposons des métadonnées Appelées SCM (Software Component Metadata) pour la description des composants et des assemblages.

II. Description d'un composant logiciel

La métadonnée de description des composants est devisée en deux parties :

- La première concerne l'aspect fonctionnel : le développeur doit donner une description de l'ensemble des services fournis et requis en précisant des mots clés.
- La deuxième partie couvre l'aspect non fonctionnel.

La figure suivante présente une description générale :

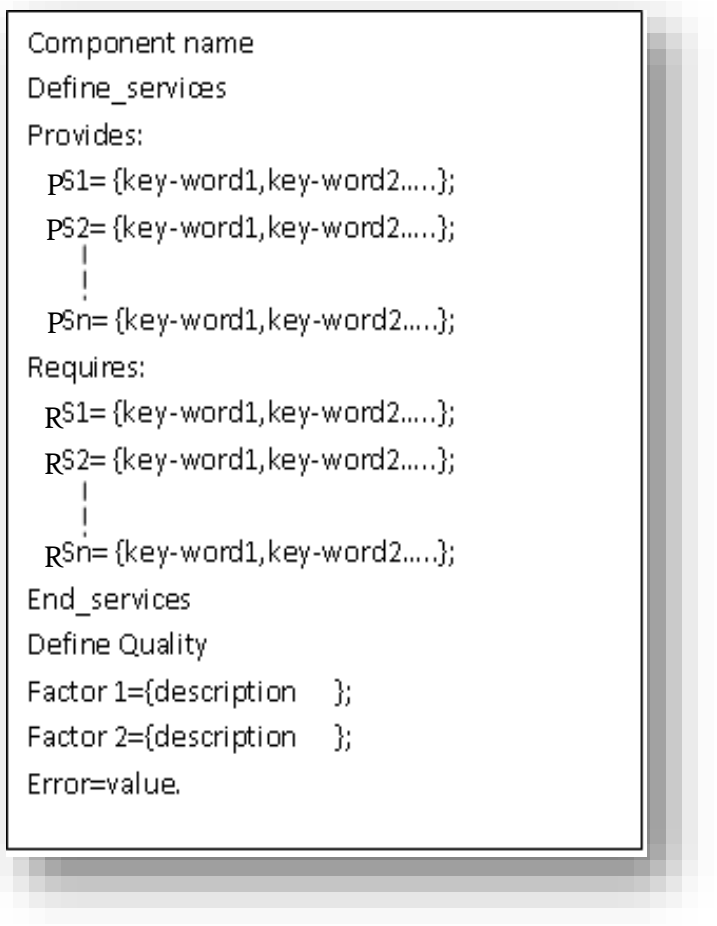


Figure IV.3 : Métadonnée de description d'un composant logiciel

II.1. Définition Régulière

Dans cette section nous donnons une description précise des expressions régulières permettant de construire la métadonnée définie dans la figure ci-dessus (Figure IV.3).

```
<description>::<functional-description><non-functional-description>[1]
<functional-description> ::'component'<ident> <serv-desc>[2]
<serv-desc>::'Define_services'<provid_serv><requir_serv>'End_Services'[3]
<provid_serv> ::'Provides :'<serv_list>[4]
<requir_serv> ::'Requires:'<serv_list>[5]
<serv_list> :: <serv_list>','<serv_def>[7]
              |<serv_def>[8]
<serv_def> :: <ident> '={'<key_words>'}'[9]
<key_words>::<key_words>','<ident>[10]
              |<ident>[11]
<non-functional-description> ::'Define_Quality' <qual-defi>'.' <error>[12]
<qual-defi> ::<qual-defi> ',' <carac-defi >[13]
                       | <carac-defi >[14]
<carac-defi>  :: <ident> '='<subs-caracs>[15]
<subs-carcs>::<num-val>[16]
                        | '{' <subs-carac> '}'[17]
<subs-carac>::<subs-carac> ',' <sub-carac>[18]
                        | <sub-carac>[19]
<sub-carac> ::<ident> '=' <attributes>[20]
<attributes> ::<num-val>[21]
                        | '{' <attribute> '}'[22]
<attribute> ::<attribute> ',' <attribute1>[23]
                        | <attribute1>[24]
<attribute1>::<ident> '=' <val-num>[25]
< val-num>::<num> ',' <num>[26]
< num>::<number>+ [27]
< ident> :: <letter>(<letter>| <number>)*[28]
<letter> ::  a | b | c | . . . | z | A | B| C| . . . | Z[29]
<error> :: 'error =' <val-num>[30]
<number>::0| 1 | 2 | 3 | . . . | 9[31]
```

Figure IV.4 : Grammaire pour générer des métadonnées de description Des composants logiciels [61][62]

La description d'un composant <description> inclut la description fonctionnelle <functional_description> et la description non fonctionnelle <non_functional_description>.

La description fonctionnelle présente l'ensemble des services fournis <provid_serv> et requis <requir_serv>. Chaque service est définit <serv_def> par un ensemble de mots clés <key_words> .

La description non fonctionnelle est constituée d'une séquence de définition de caractéristiques <carac-defi> qui peuvent avoir des valeurs numériques ou présentées par un ensemble de sous-caractéristiques <subs-carcs>. De même une sous-caractéristique est présentée sous la forme d'une séquence d'attributs <attributs>. Un attribut <attribut1> est présenté par une valeur numérique.

Exemple : l'exemple suivant correspond à une métadonnée d'un composant offre un service S1 et requit un service S2

```
Component C1
Define-Services
Provides
S1={fact}
Requires
S2={mult}
End-Service
DEFINE-QUALITY
Fiabilité = {
        Tolerance-faute = {
                Disponibilité-mécanisme =0,6;
                Efficacité-mécanisme    =0,4
            };
        Recupuration =  {
                Gestion-Erreur  =0,5
            }
    };
Utilisabilité = {
        Configurabilité ={
                Effort-configuration =0,3
            };
        }.
Error=0.001
```

Figure IV.5 : Exemple de Description d'un composant

III. Description de l'assemblage de composants Logiciels

Figure IV.6 : Description d'un assemblage de composants

La métadonnée de description d'un assemblage de composants logiciels présentée dans la figure IV.6 est constituée de quatre rubriques :

1. **name** : un identificateur représente le nom de l'assemblage.
2. **Components** : donne la liste des composants appartenant à l'assemblage.
3. **Define_Services** : spécifie l'ensemble des services fournis (Provides) et requis (Requires) de l'assemblage.

4. **Links :** dans un assemblage, les services requis par certains composants sont connectés aux services offerts par d'autres composants. Cette rubrique représente les liaisons nécessaires à la réalisation de l'assemblage (**Connect( )**).

Exemple :

La figure IV.7 représente l'architecture pour un Guichet Automatique Bancaire. Cette architecture met en évidence deux composants : BASE_GAP et INTERFACE_CLI.

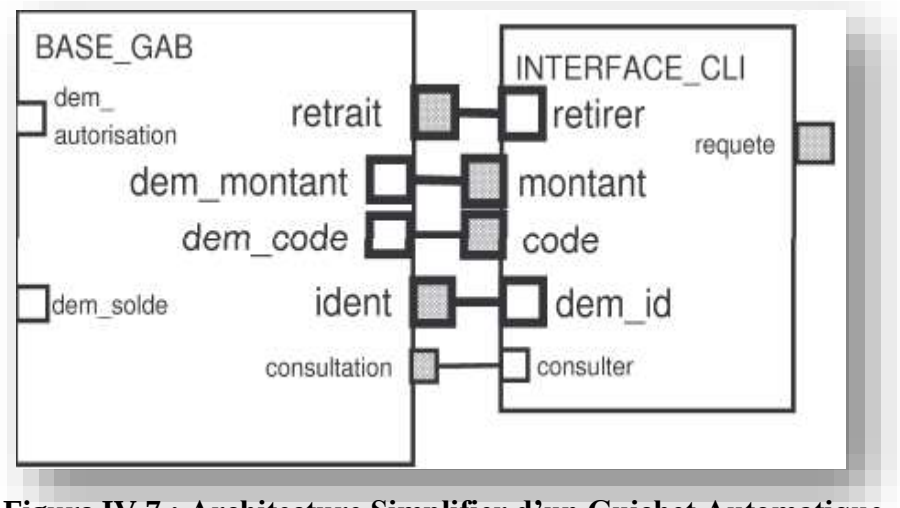


Figure IV.7 : Architecture Simplifier d'un Guichet Automatique Bancaire (GAP) [63]

Ce système est décrit comme suit :

Assembly GAP

Components:

B:BASE_GAP, I: INTERFACE_CLI

Define_Services

Provides :

I.Requete

Requires :

B. Dem_autorisation, B.Dem_solde

End_Services

Links:

Connect(B.retrait,I.retirer)

Connect(B. dem_montant ,I. montant)

Connect(B. dem_Code,I.code)

Connect(B.ident, I.dem_id)

Connect(B.consultation, I.consulter)

End_Assembly

Figure IV.8 : Description de l'Architecture (GAP)

Les Processus d’évaluation

I. Introduction

Le but principal de ce travail est d'évaluer la qualité d'un système afin de trouver le meilleur parmi plusieurs équivalents (offrent les mêmes besoins fonctionnels). Comme solution nous proposons d'accorder à chaque système une valeur numérique qui représente sa valeur globale de qualité.

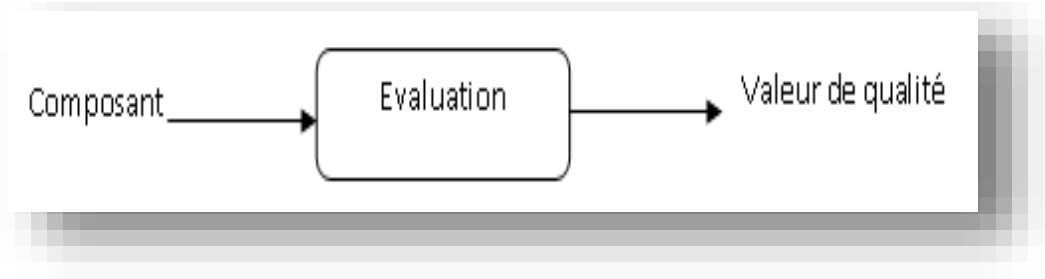


Figure IV.9 : l'objectif de l'évaluation

Dans cette section est nous présentons les processus d'évaluation proposés. Nous commençons dans la section II par l'évaluation de la qualité d'un composant. Ensuite dans la section III, les résultats obtenus sont généralisés sur l'assemblage des composants logiciels.

II. Evaluation de la qualité d'un composant

II.1. La relation de représentation

Un modèle de qualité est décomposé en trois niveaux. Le premier niveau représente les facteurs ou les caractéristiques, le second correspond aux critères où les sous-caractéristiques et le dernier pour la spécification des attributs.

Supposant que :

- **F** : l'ensemble de facteurs :

$$F= \{f_{i / i=1..n}\}$$

- **C** : l'ensemble de critères :

$$C= \{f_{j / j=1..m}\}$$

- **A** : l'ensemble de critères :

$$A= \{a_{l / l=1..k}\}$$

Pour cela nous définissons la relation **β** : relation de représentation comme suit :

$E_i \beta e_{j/j=1..n}$ ➔signifie que l'élément E est présenté par l'ensemble $(e_1,e_2\ldots,e_n)$

Deux cas peuvent être distingués :

- ✓ **Cas N°1 :** relation critère / attribut

$$\forall \; c_i \in C, c_i \; \beta \; a_j \qquad (I)$$

⇒ Chaque critère cj est présenté par un ensemble d'attributs.

- ✓ **Cas N°2 :** relation facteur / critère

$$\forall \; f_j \in F, f_j \; \beta \; c_l \qquad (II)$$

⇒ Chaque facteur fj est présenté par un ensemble de critères

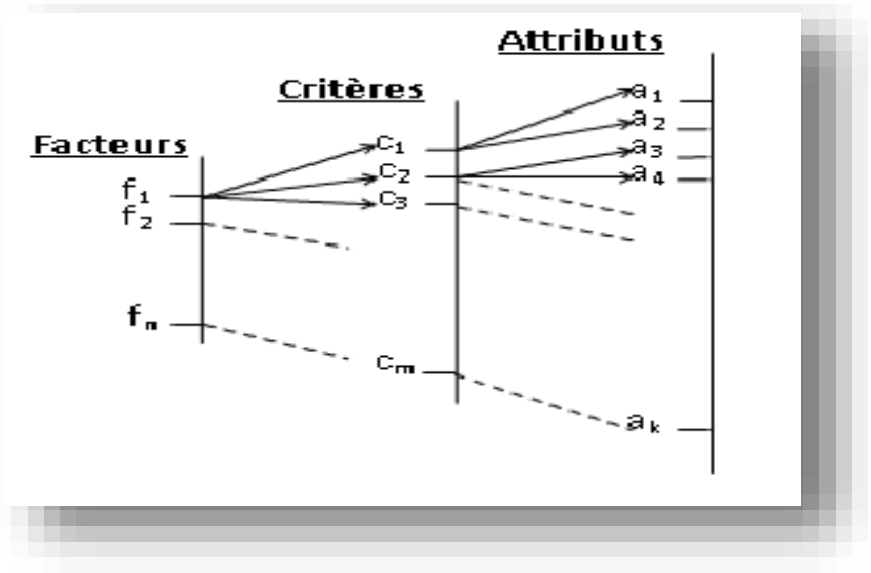


Figure IV.10. La relation de représentation

Exemple : on suppose que nous avons la description suivante :

```
DEFINE-QUALITY
Reliability = {
        Fault-Tolerance = {
                        Mechanism availability = 0,6,
                        Mechanism Efficiency = 0,4
                        },
        Recoverability = {
                        Error Management = 0,5
                        },
         },
Usability = {
        Configurability = {
                        Configuration Effort = 0,3
                        },
}.
Error = 0,01.
```

Figure IV.11 : Exemple de description d'un composant (partie non fonctionnel)

Pour ce cas nous avons les relations suivantes :

1. Usability β configurability,
2. Configurability β configuration effort,
3. Reliability β (recoverability, fault tolerance),
4. Recoverability β error management,
5. Fault tolerance β (mechanism-availability, mechanismeffectiveness).

II.2.La fonction d'évaluation

L'objectif de ce travail est de faciliter la comparaison des composants équivalents pour les différents types d'utilisateurs. L'idée est d'attribuer à chaque composant une valeur numérique (QV : Quality Value) représente la valeur globale de qualité. Pratiquement ; seuls les attributs sont mesurables.

Le problème est :

Comment calculer QV à partir des valeurs fournis des attributs ?

Comme solution nous définissons la fonction d'évaluation £ comme suit :

$$\text{Si } E_i \beta e_{j\ /\ j=1..n} \Rightarrow V(E_i) = £(V(e_j))$$

Si l'élément E_i est présenté par l'ensemble d'éléments (e_1, e_2, e_3… e_n), la valeur de qualité de E_i : $V(E_i)$ peut être calculée par les valeurs des éléments e_j: $V(e_j)$.

$$(I) \Rightarrow V(c_i) = £(V(a_j)) \quad /j=1..n$$

$$(II) \Rightarrow V(f_j) = £(V(c_l)) \quad /l=1..m$$

$$\text{De même} \quad QV = £(V(f_k)) \quad /k=1..r$$

La fonction £ représente la somme pondérée.

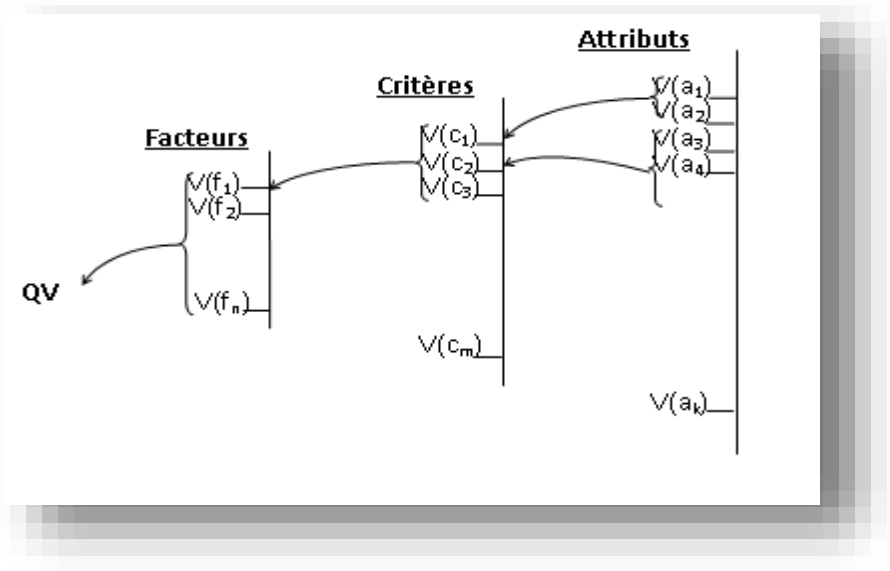


Figure IV.12. Evaluation d'un composant

Sachant que :

- $V(F_i)$: la valeur du facteur F_i.
- $V(C_j)$: la valeur du critére C_j.
- $V(A_l)$: la valeur de l'attribut A_l.

II.3. Processus N°1: Le Processus d'évaluation de la qualité de composants (SCEP: Software Component Evaluation Process) [64]

Le processus d'évaluation proposé est basé sur la méthode d'analyse MAGIQ, il contient quatre étapes: la spécification des besoins, le calcul des poids, l'évaluation et le choix du composant optimal.

1. **<u>Etape N°1</u> : Spécification des besoins**

Dans cette étape l'utilisateur doit :

- Classer les caractéristiques de qualité (facteurs, critères et attribut) ou préciser le poids de chaque élément.
- Spécifier le type de la recherche :
    1. **Exact** : Pour cette option l'utilisateur doit préciser les valeurs des caractéristiques non fonctionnelles du composant recherché.
    2. **Meilleur parmi** : le but de ce type est de comparer des composants et de désigner le (s) meilleur(s) candidat(s).

2. **<u>Etape N°2</u> : Calcul des poids**

Dans le cas où l'utilisateur donne un classement (sans poids), le processus convertis les rangs en poids en utilisant le concept ROC : Si n est le nombre d'attributs, le poids (W) de l'attribut k (A_k) est :

$$W(Ak) = \frac{\left(\sum_{i=k}^{n} \frac{1}{i}\right)}{n} \quad (R1)$$

Les poids des facteurs sont calculés selon le classement de l'utilisateur en utilisant la règle R1. Pour les critères et les attributs nous proposons deux suggestions : P1 et P2

- **P1**: l'intervention de l'utilisateur est décisive pour le classement des critères et des attributs (utilisation de la règle R1).

- **P2**: utiliser le même poids, c'est à dire . si un élément E_i est présenté par n élément e_j **(E_i β $e_{j\ /\ j=1..n}$)**, alors le poids de chaque élément e_j est égale : $\frac{1}{n}$

$$W(e_j) = \frac{1}{n} \quad (R2)$$

2.1. <u>Etape N°3</u> : Evaluation:

Le but de cette étape est de produire la valeur globale de qualité (**QV**) en utilisant les règles suivantes qui sont basées sur la relation β et la fonction £ (cf. figure IV.13):

$$QV = \sum_{i=1}^{n} \big(V(Fi) * W(Fi)\big) \pm error \qquad \text{(R3)}$$

$$V(Fi) = \sum_{j=1}^{m} \big(V(Cj) * W(Cj)\big) \qquad \text{(R4)}$$

$$V(Ci) = \sum_{p=1}^{k} \big(V(Al) * W(Al)\big) \qquad \text{(R5)}$$

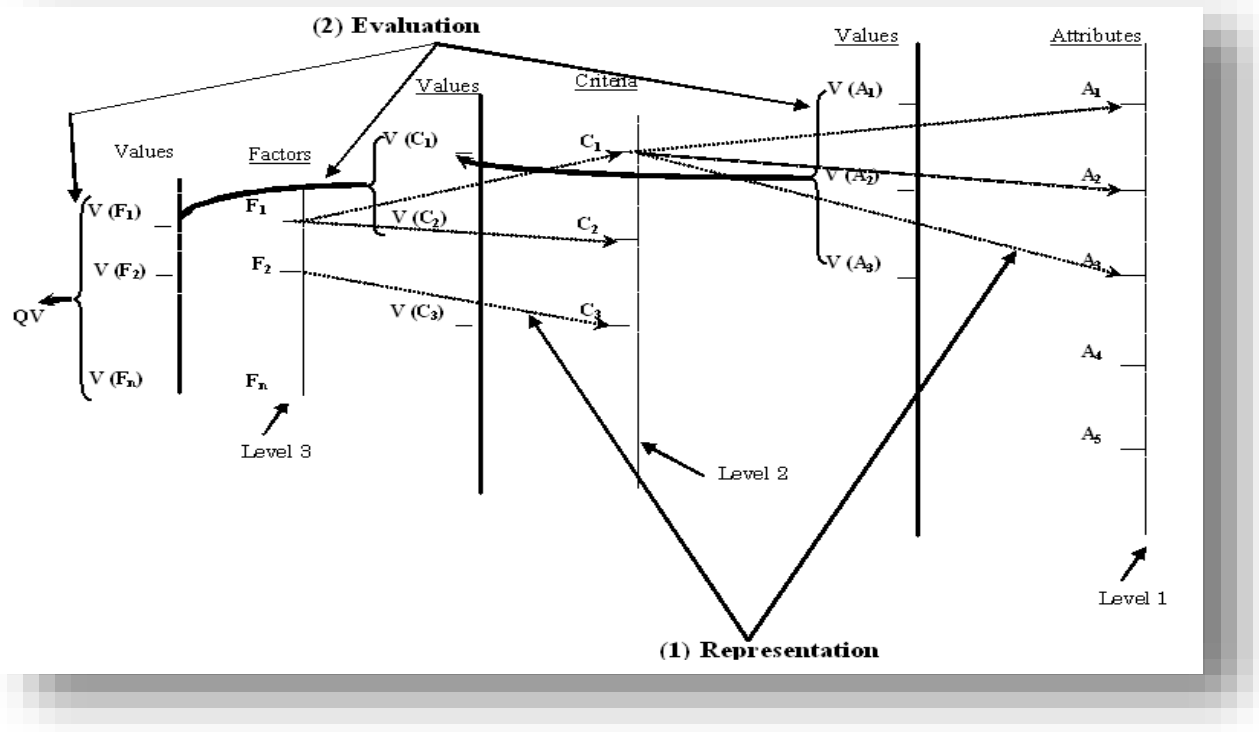


Figure IV.13 : Production de la valeur globale de qualité

3. Etape N°4 : choix du composant optimal

Dans le cas d'une recherche de type « Meilleur parmi » le composant optimal est celui qui a une valeur de qualité maximale. Alors que dans le cas d'une recherche « exact » et afin de sélectionner le composant le plus proche (plus pertinent) au composant recherché, nous proposons de mesurer la distance entre les composants candidats et le composant recherché. Pour cela nous utilisons la distance euclidienne pondérée.

$$D(X,Y) = \sqrt[2]{\sum_{i=1}^{k} \mathrm{w}^2 (x_i - y_i)^2}$$

II.3.1 Application du processus

Soit un composant décrit (partie non fonctionnel) comme suit :

```
DEFINE-QUALITY
Reliability = {
        Fault-Tolerance = {Mechanism availability = 0,6;
                           Mechanism Efficiency = 0,31};
        Recoverability = {Error handling= 0,20};
            };
Usability = {
        Configurability = {Configuration Effort = 0,3};
        Operability={
                      Provide_interface=0.55;
                      Required_interface=0.36;
                    }
        Learnability={
                      time_configuration=0.69;
                     }
            }.
Error = 0,01
```

Figure IV.14: Exemple de description d'un composant

Etape N°1: Classement

Nous supposons que:

1. Le facteur fiabilité (reliability) est plus important que le facteur utilisabilité (Usability).
2. Les critères et les attributs ont la même importance (proposition P2).

<u>Etape N°2</u>: Calcul des poids

1. **Poids des facteurs (utilisation de la règle R1)**
    1. **La fiabilité :**

$$W(reliability) = \sum_{i=1}^{2} \frac{\left(\frac{1}{i}\right)}{2}$$

$$= \left(\frac{1}{1} + \frac{1}{2}\right)/2 = 0.75$$

Pour la fiabilité nous distinguons les relations suivantes :

- Reliability β (Recoverability, Fault tolerance),
- Recoverability β error handling
- Faulttolerance β (mechanismavailability, mechanismeffectiveness).

En utilisant **R2** on trouve :

Critère	Poids	Attribut	Poids
Recoverability	0.5	error handling	1
Faulttolerance	0.5	mechanismavailability,	0.5
		mechanismeffectiveness	0.5

    2. **Utilisabilité:**

$$W(Usability) = \sum_{i=2}^{2} \left(\frac{1}{i}\right)/2$$

$$= \left(\frac{1}{2}\right)/2 = 0.25$$

Pour ce facteur nous avons ces relations :

- Usability β (Configurability,Operability, Learnability)
- Configurability β Configuration-Effort
- Operability β (Provided-interface, Required-interface)
- Learnability β Time-Configuration

En utilisant R2 on trouve :

Critère	Poids	Attribut	poids
Configurability	0.33	Configuration-Effort	1
Operability	0.33	Provided-interface	0.5
		Required-interface	0.5
Learnability	0.33	Time-Configuration	1

Etape N°3: Evaluation

Afin de déduire QV, les valeurs de qualité des facteurs et critères doivent être calculées.

1. **La valeur de qualité de l'Utilisabilité :**
   a) **Les valeurs de qualité des critères : (Application de R5)**

(r2) $\Rightarrow$ V(Operability)=V(Provided_interface)*W(Provided_interface)+ V(Required_interface)*W(Required_interface)= 0.5*0.55+0.5*0.36=0.45

V(Operability)= 0.45

La même règle **R5** sera appliquée pour trouver les valeurs de qualités des critères **Learnability** et **Operability**, nous trouverons:

V(Learnability)= 0.69 **et** V(Configurablity)=0.3

b) Les valeurs de qualité du facteur : (Application de R4)

(r1) $\Rightarrow$ V(Usability)=V(operability)*W(operability)+ (Learnability)*W(Learnability)+
V (Configurablity)*W(Configurablity) = 0.33*0.45+0.33*0.69+0.33*0.3=0.47

V(Usability)=0.47

2. **La valeur de qualité de la fiabilité :** nous appliquons le même processus nous trouverons :

V(Reliability)=0.34

3. **La valeur de qualité de globale de la qualité (QV) :** Pour ce niveau la règle R3 est utilisée :

QV=V(Reliability)*W(Reliability)+V(Usability)*W(Usability)$\pm$ error
=0.34*0.25+0.47*.75 $\pm$ 0.01

QV=0.43$\pm$ 0.01

II.3.2 Evaluation

Le problème de la sélection des composants a été abordé dans diverses recherches. La plupart des solutions ne considèrent que les exigences fonctionnelles.

Dans le champ CBSE, peu de travaux portent sur la qualité. Dans la plupart des approches existantes [16] [17] [18] [19], l'évaluation de la qualité des composants est fondée sur l'analyse des caractéristiques de qualité, ou bien elle est basée sur le concept « composants recherché » tel que [26] et [64].

Pour illustrer les avantages de la méthode proposée, nous allons :

- Appliquer notre processus d'évaluation sur le cas présenté dans [20] (**meilleur parmi**).
- Effectuer **recherche exact** sur une base de composants.

1. Le Meilleur composant :

Le cas suivant teste deux implémentations de protocole SIP : v2.24 et v3.06. Le tableau ci-dessous présente la description des deux versions :

Caractéristiques	Sous caractéristique	SIP v2.24	SIP 3.06
Analyzability	Complexity violation/ cyclomatic complexity	0.98	0.98
	Complexity violation/ number of statements	0.77	0.71
Changeability	Code duplication	0.25	0.26
	References violation	0.96	0.96
Stability	Global variable and timer usage	0.15	0.15
	Parameter reassignment	0.61	0.68

Tableau V.1 : description de deux version SIP (V2.24 et V3.06)

Nous avons les relations suivantes :

1. Analyzability β (cyclomatic complexité, number of statements),
1. Changeability β (code duplication, references violation),
2. Stability β (global variable and timer usage, parameter reassignment),

On suppose que les poids sont spécifiés comme suit :

Caractéristiques	Poids	Sous caractéristique	Poids	SIP v2.24	SIP 3.06
Analyzability	0.09	Complexity violation/ cyclomatic complexity	0.5	0.98	0.98
		Complexity violation/ number of statements	0.5	0.77	0.71
Changeability	0.10	Code duplication	0.5	0.25	0.26
		References violation	0.5	0.96	0.96
Stability	0.81	Global variable a nd timer usage	0.5	0.15	0.15
		Parameter reassignment	0.5	0.61	0.68

Après l'application du processus SCEP, on trouve les résultats suivants :

	SIP v2.24	***SIP 3.06***
V(Analyzability)	0.98*0.5+0.77*0.5 =0.88	0.98*0.5+0.71*0.5 =0.85
V(Changeability)	0.25*0.5+0.96*0.5 =0.61	0.26*0.5+0.96*0.5 =0.61
V(Stability)	0.15*0.5+0.61*0.5 =0.38	0.15*0.5+0.68*0.5 =0.42
QV	0.38*0.81+0.61*0.1+0.88*0.09 = **0.44**	0.42*0.81+0.61*0.1+0.85*0.09 **=0.48**

Discussion :

Les résultats montrent que :

1. Les deux protocoles assurent le même niveau de changeabilité (valeur 0.61 pour les deux),
2. SIP 2.24 est plus analysable que SIP 3.06 (valeurs: 0.88 et 0.85) ,
3. Et SIP 3.06 est plus stable que le que SIP 2.24 (valeurs :038 et 0.42).

Dans ce cas l'utilisateur cherche le plus stable. A partir des résultats finaux on peut conclure que SIP 3.06 est le plus approprié (0.44 et 0.48).

Si on effectue une analyse attribut par attribut on remarque que :

1. la valeur de l'attribut global variable and timer usage est égale à 0.15 pour les deux versions.
2. Pour l'attribut **parameter reassignment** les valeurs sont (0.61 , 0.68) respectivement (SIP v2.24 , SIP 3.06) .

Donc il est clair que la version 3.06 assure un niveau de stabilité plus élevé que la version 2.24. Ainsi les valeurs de qualité finales obtenues par notre processus illustrent cette différence.

Conclusion1 : la production de la valeur globale de qualité facilite la comparaison.

2. Recherche exact :

Pour faire une recherche exact, nous avons crée une base de composants. Chaque composant est représenté par trois attributs. les valeurs de ces derniers sont générées d'une façon aléatoire et réparties sur l'intervalle [0,1]. Le nombre total des enregistrements crée est : **9900.** On considère que les 9900 composants sans équivalents.

On suppose que les attributs sont classés comme suit :

(1) A1 (2) A2 (3) A3

En utilisant la règle R1 on trouve :

w(A1)=0.61 w(A2)=0.28 w(A3)=0.11

On suppose que les requêtes suivantes ont été posées par l'utilisateur :

- **Requête N° 1 : Composant recherché** : A1 :0,64 A2 :0,47 A3 :0,33

Selon SCEP : Qualité demandé=0.56.

L'application du processus d'évaluation sur la base donne 151 candidats assurant 0.56 comme valeur de qualité.

Ensuite nous avons mesuré la distance (Dis) entre les candidats et le composant recherché.

Le tableau suivant illustre les résultats obtenus:

N°	A1	A2	A3	Dis	A1*0,61	A2*0,28	A3*0,11
1	0,64	0,47	0,33	0,0000	0,3904	0,1316	0,0363
2	0,64	0,47	0,36	0,0033	0,3904	0,1316	0,0396
3	0,64	0,47	0,3	0,0033	0,3904	0,1316	0,033
4	0,64	0,47	0,39	0,0066	0,3904	0,1316	0,0429
5	0,64	0,52	0,24	0,0171	0,3904	0,1456	0,0264
6	0,64	0,42	0,45	0,0192	0,3904	0,1176	0,0495
….							
17	0,71	0,42	0,12	0,0505	0,4331	0,1176	0,0132
18	0,71	0,37	0,24	0,0520	0,4331	0,1036	0,0264
19	0,71	0,42	0,09	0,0521	0,4331	0,1176	0,0099
20	0,57	0,52	0,57	0,0521	0,3477	0,1456	0,0627
….							
149	0,36	0,97	0,6	0,2228	0,2196	0,2716	0,066
150	0,36	0,97	0,63	0,2233	0,2196	0,2716	0,0693
151	0,36	0,97	0,66	0,2238	0,2196	0,2716	0,0726

Tableau 1: Extrait du résultat de la requête R1

Le principe des processus d'évaluation de qualité existants tels que [26] et [64] est de comparer tous les composants similaires (attributs par attributs) avec le composant recherché :

- ➔ n*k opérations de comparaisons pour calculer le score de chaque composant.
- ➔ n opérations pour trouver le score maximum.

Tel que n est le nombre de composants et k le nombre d'attributs.

Par contre le principe du processus SCEP permet de minimiser le nombre de comparaisons en limitant celles-ci aux composants dont la valeur globale de qualité est proche à la valeur demandée :

- ➔ n opérations pour calculer la valeur globale de qualité de chaque composant.
- ➔ m opérations pour calculer la distance.
- ➔ m opérations pour trouver la distance minimale.

  Tel que m est le nombre des composants candidats

Conclusion N°2: La production de la valeur globale de qualité permet de réduire la complexité du processus de sélection.

Un autre avantage du processus proposé est la possibilité de faire une recherche en spécifiant seulement la valeur de qualité désiré et le classement des caractéristiques de qualité.

Requête N° 2 : Qualité >0,96

A1	A2	A3	Q
0,99	0,97	0,99	0,98
0,99	0,97	0,96	0,98
0,99	0,97	0,93	0,98
0,99	0,97	0,9	0,97
0,99	0,97	0,87	0,97
0,99	0,97	0,84	0,97
0,99	0,92	0,99	0,97
0,99	0,92	0,96	0,97

Tableau 2: les résultats de la requête N°2

III. L'évaluation de la qualité d'un assemblage [66]

L'ingénierie à base de composants logiciels est basée sur l'assemblage de composants préexistants (cf. figure IV.15). Plusieurs compositions peuvent assurer les besoins fonctionnels du système demandé.

L'objectif de cette section est d'étudier l'assemblage de composants logiciels pour trouver une méthode qui facilite la sélection du meilleur assemblage.

On suppose que :

1. RS (Requested System): le système demandé
2. F: l'ensemble des facteurs spécifiés par l'utilisateur :

$$F= \{f_{i\ /\ i=1..\ n}\}$$

3. Cm: l'ensemble de composants qui peuvent assurer les besoins du système RS :

$$Cm= \{cm_{j\ /\ j=1..\ m}\}$$

4. S: l'ensemble des systèmes candidats (équivalent à RS) produit de l'assemblage des composants cm_j :

$$S=\{S_{i\ /i=1..k}\}.$$

Un système S_i est représenté comme suit :

$$S_i=\mathcal{A}\ (Cm_{i/i=1.\ .k}\ /F)$$

Sachant que $\mathcal{A}$ représente le symbole d'assemblage.

5. AS_k: un ensemble d'attributs relatifs à S_k:

$$AS_k=\{a_{jk/j=1..m}\}$$

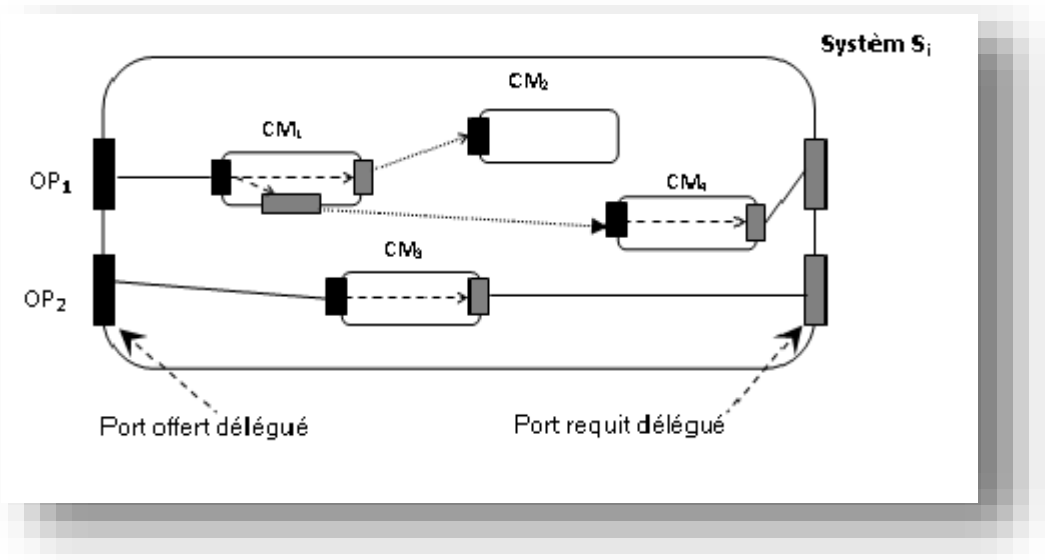


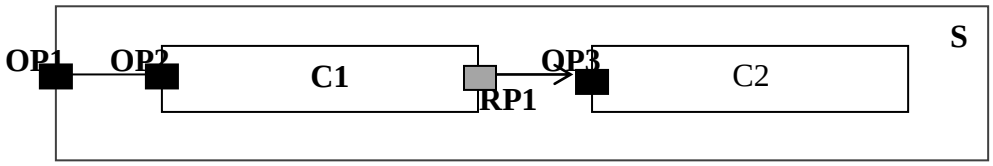
Figure IV. 15. L'assemblage de composants logiciels

La qualité du système S_i dépend de la qualité de ses composants [15]. Ainsi la valeur globale de qualité d'un système peut être déduite à partir des valeurs de qualité de ses composants.

Si on considère l'exemple présenté dans la figure IV.15, nous remarquons que les ports offerts OP_1 et OP_2 du système S_i dépendent des ports offerts des composants CM_1 et CM_2.

Le port offert de CM_1 est lié à deux ports requis utilisant les résultats des composants CM_2 et CM_4. Ainsi, la qualité offerte par ces dernières influe d'une manière ou l'autre sur la qualité de l'OP 1 (voir le cas ci-après).

III.1. Etude de cas



Si on considère l'attribut **Functional-adequacy.** Selon [24] la valeur de cet attribut est calculée comme suit :

$$\textbf{Val (Functional-adequacy)}=1-\frac{x}{y}$$

Tel que :

x= nombre de fonctions dans lesquelles des problèmes sont détectés.

y= nombre de fonctions évaluées

Supposant que:

- Pour le composant C_1 nous avons : x=0 and y=20

$\textbf{Val}\ (\textbf{Functional}-\textbf{adequacy}\ )=1-\frac{0}{20}=1$;

➔ La pertinence de C_1 est excellente (toutes les fonctions évaluées sont corrects)

- Et pour C_2 : x=10 and y=18

$val\ =1-\frac{10}{18}=0.44$;

➔ La pertinence de C_2 acceptable.

Question : Quelle est la valeur de pertinence de S ?

Nous Remarquons que le port fourni (OP1) du système S est réalisé par le port fourni (OP2) du composant C_1 (relation de délégation). Ce dernier est lié au port requit (RP1) qui utilise les résultats obtenus par le port OP3.
Ainsi, une opération fournie de C_1 peut utiliser les résultats faux d'une opération mal implémentée de C_2. D'où la possibilité d'erreur.
C'est-à-dire que le système S n'est pas pertinent à 100%. Donc il est préférable d'utiliser la moyenne …… (R_6).

$$\mathbf{Val\ (Functional-adequacy\ )} = \frac{\mathbf{1+0.44}}{\mathbf{2}} = \mathbf{0.72}$$

Si nous analysons l'ensemble des attributs présentés dans l'annexe nous constatons la même remarque.

La règle (R_6) sera généralisée sur l'ensemble des attributs de qualité.

III.2.Process 2: Le processus d'évaluation de la qualité d'un assemblage de composants logiciels (SCAEP : Software Component Assembly Evaluation Process)

Ce processus est basé sur le processus SCEP présenté dans la section II.3. Il contient quatre étapes :

1. **Etape N° 1: Spécification des besoins /calcul des poids**

   Elle partage les mêmes tâches que les étapes spécification des besoins et calcul des poids du processus SCEP.

2. **Étape N°2 : Assemblage**

   Assembler les composants logiciels afin de construire l'application demandée : déterminer toutes les compositions possibles (tous les systèmes candidats : $S_{n/\ n=1.\ g}$, g nombre de compositions possibles).

3. **Step3: Evaluation**

   Evaluer la qualité fournie (PQ : Provided Quality) par chaque système. L'évaluation est décomposée en deux phases:

   a) **Phase N°1: calcul des vecteurs $AS_{n/n=1..g}$:**

La valeur d'un attribut d'un système S_k peut être dérivée à partir des valeurs des attributs de ses composants.

Attributs / **composants**	a_1	a_2	a_3	....	a_m
cm_1	$V(a_{11})$	$V(a_{21})$	$V(a_{31})$	....	$V(a_{m1})$
cm_2	$V(a_{12})$	$V(a_{22})$	$V(a_{32})$	....	$V(a_{m2})$
. . .					
cm_k	$V(a_{1k})$	$V(a_{2k})$	$V(a_{3k})$	....	$V(a_{mk})$
System S_n	$V(a_{1Sn})$	$V(a_{2Sn})$	$V(a_{3Sn})$	....	$V(a_{mSn})$

Ainsi, sur la base de (R6) → $V(a_{Jsn}) = AVG(a_{ji})$ i= 1. .k

$$AS_n = \begin{cases} a_{1Sn} = AVG(a_{11}, a_{12}, ..., a_{1k}) \\ a_{2Sn} = AVG(a_{21}, a_{22}, , ..., a_{2k}) \\ \\ a_{mSn} = AVG(a_{m1}, a_{m2}, ..., a_{mk}) \end{cases}$$

$V(a_{jsn})$: La valeur de l'attribut j du système candidat S_n est égale à la moyenne des valeurs des attributs j des différents composants.

b) Phase N°2 : La production de la valeur globale de qualité : chaque assemblage (un système S_i) est considéré comme un composant. Après la production des valeurs du vecteur of AS . Nous appliquons l'étape d'évaluation (Etape N° 3) du processus SCEP.

4. <u>Etape N° 4</u>: Choix de l'application optimale

Elle a le même principe que la dernière étape : choix du composant optimal du processus SCEP.

La figure suivante résume le processus SCAEP

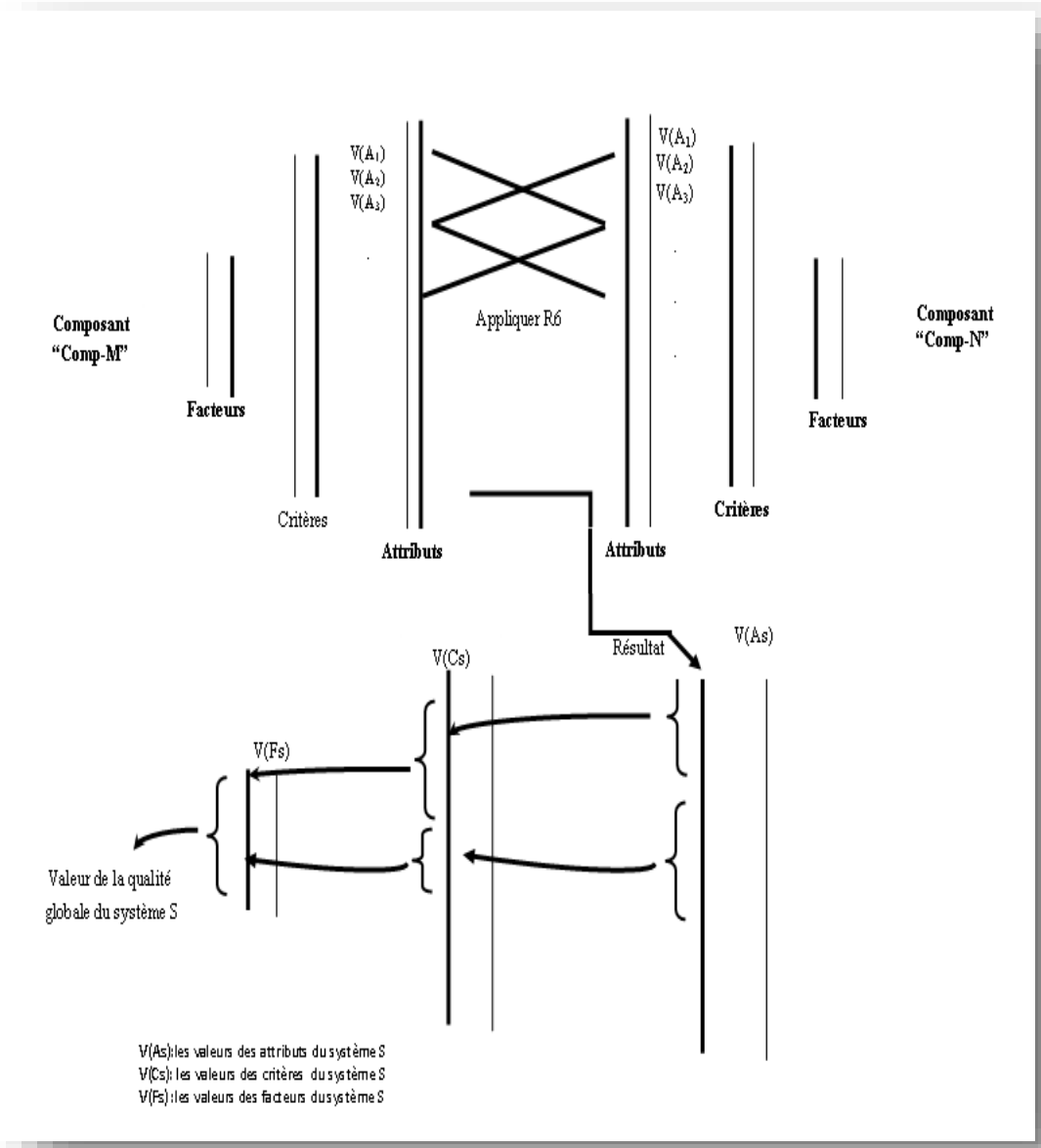


Figure IV. 16. Evaluation de l'assemblage de composants logiciels selon SCAEP

IV. Conclusion

La métadonnée présentée dans cette partie permet une description fonctionnelle et non fonctionnelle des composants logiciels. En plus elle est applicable à n'importe quel modèle de qualité. C'est l'entrée principale des processus d'évaluation de la qualité.

Le concept valeur globale de qualité introduit dans cette partie ; facilite la comparaison des logiciels et évite la comparaison paire à paire ce qui réduit la complexité des processus de sélection.

CHAPITRE V:

IMPLEMENTATION

SOMMAIRE

I. Introduction

Ce chapitre présente un prototype d'un environnement de simulation de qualité des applications basées sur l'assemblage de composants logiciels. Ce prototype permet de valider les propositions de cette thèse.

Nous commençons par l'architecture générale de l'environnement développé. Ensuite son comportement fonctionnel sera présenté par des diagrammes UML (Unified Modeling Language).

II. Architecture générale

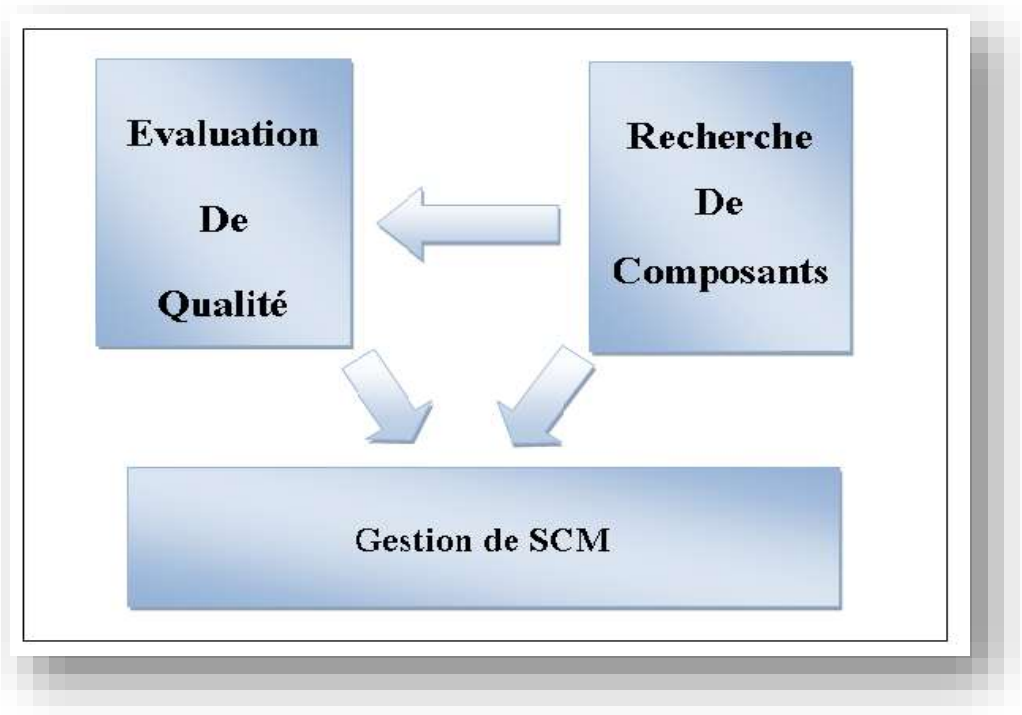


Figure V.1 : Architecture Générale du Simulateur

Le prototype de simulation de qualité des applications basées sur l'assemblage de composants présenté dans ce chapitre est composé de trois modules (cf. figure V.1) :

1. **Module N°1 : La gestion de SCM :** c'est le module de base de notre système. Son but est d'analyser les métadonnées des composants, d'extraire les principales caractéristiques des composants et de les enregistrer dans la base de composants.

2. **Module N°2 : La recherche de composants :** après la spécification des besoins demandés par les utilisateurs le système explore la base de composants et retourne les composants qui peuvent assurés les services requis.

3. **Module N°3 : L'évaluation de qualité :** ce module estime la qualité des composants retournés par le module N°2 selon les processus d'évaluation SCEP et SCAEP présentés dans le chapitre IV.

III. Comportement fonctionnel du système développé

Le diagramme décrit ci-dessous nous donne une vision globale du comportement fonctionnel du système développé.

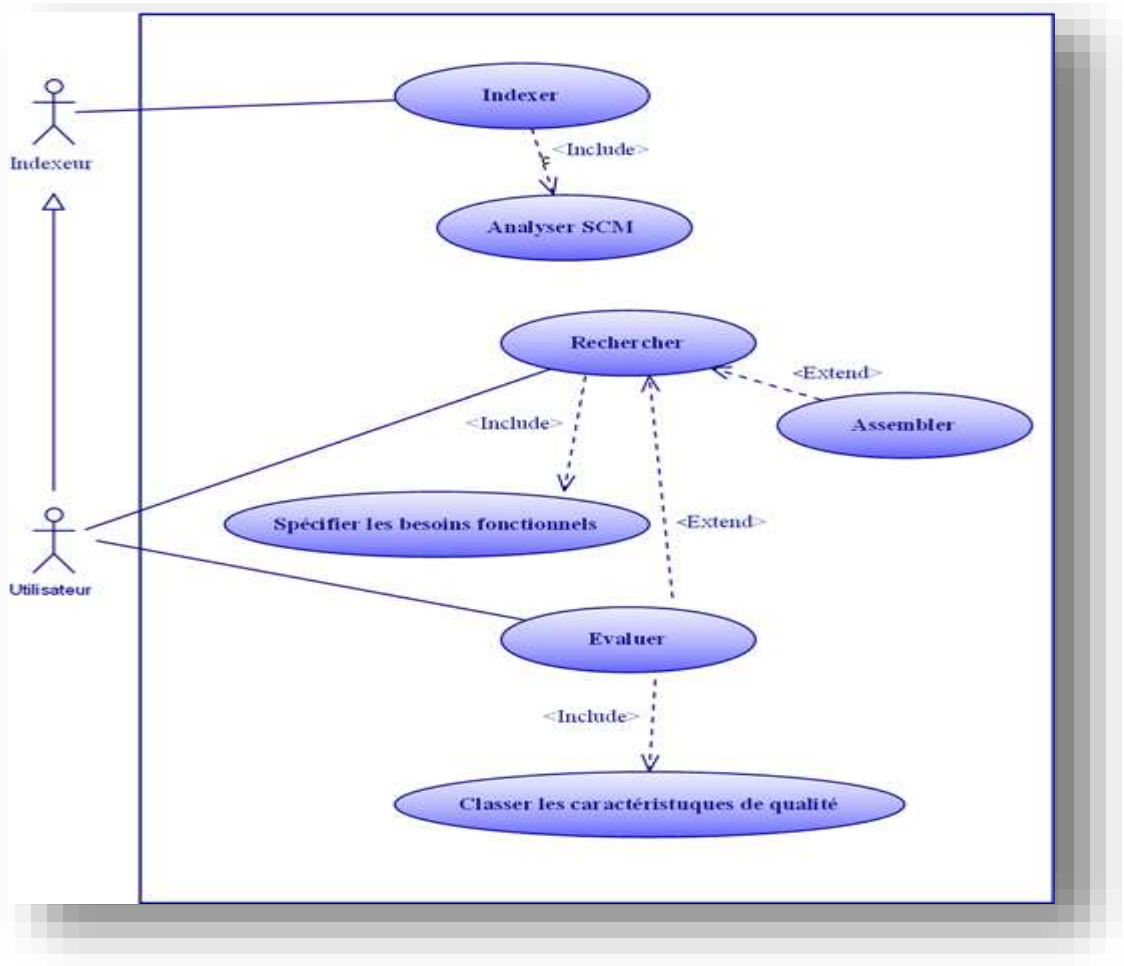


Figure V.2. Diagramme de cas d'utilisation du système

III.1. Les acteurs

Les utilisateurs de notre système sont regroupés en deux catégories :

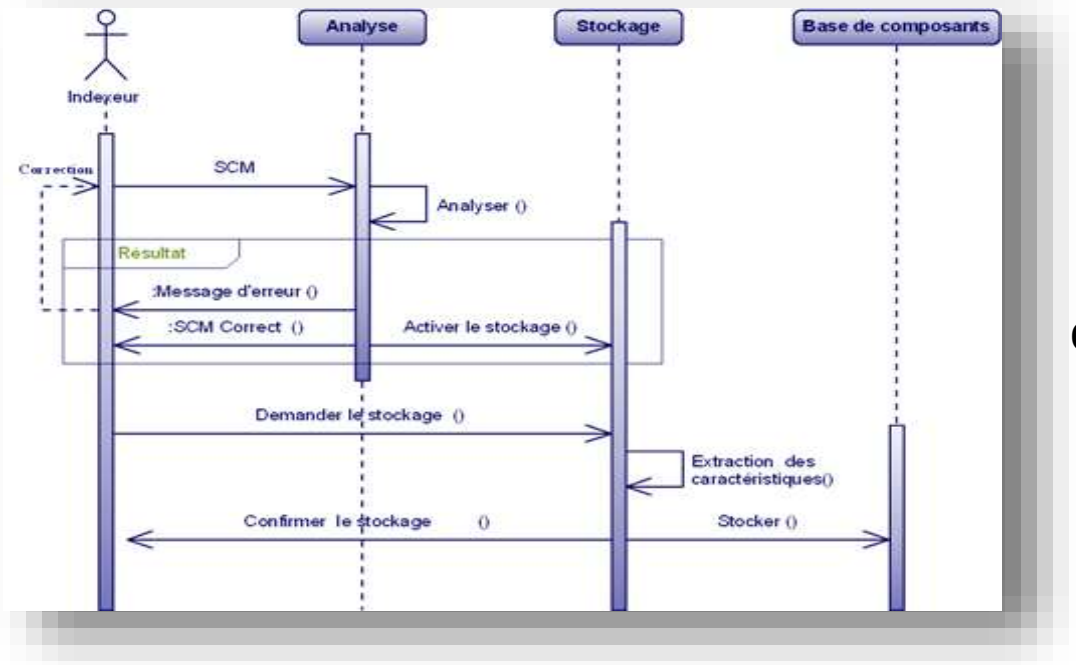
1. **Les indexeurs** : ont pour rôle de spécifier les métadonnées de description de composants.
2. **Les utilisateurs**: cette catégorie présente les utilisateurs sollicitant des services, ils peuvent être des développeurs qui recherchent des composants pour la construction de nouveaux logiciels (assemblage).

III.2. Analyse des principaux cas d'utilisation

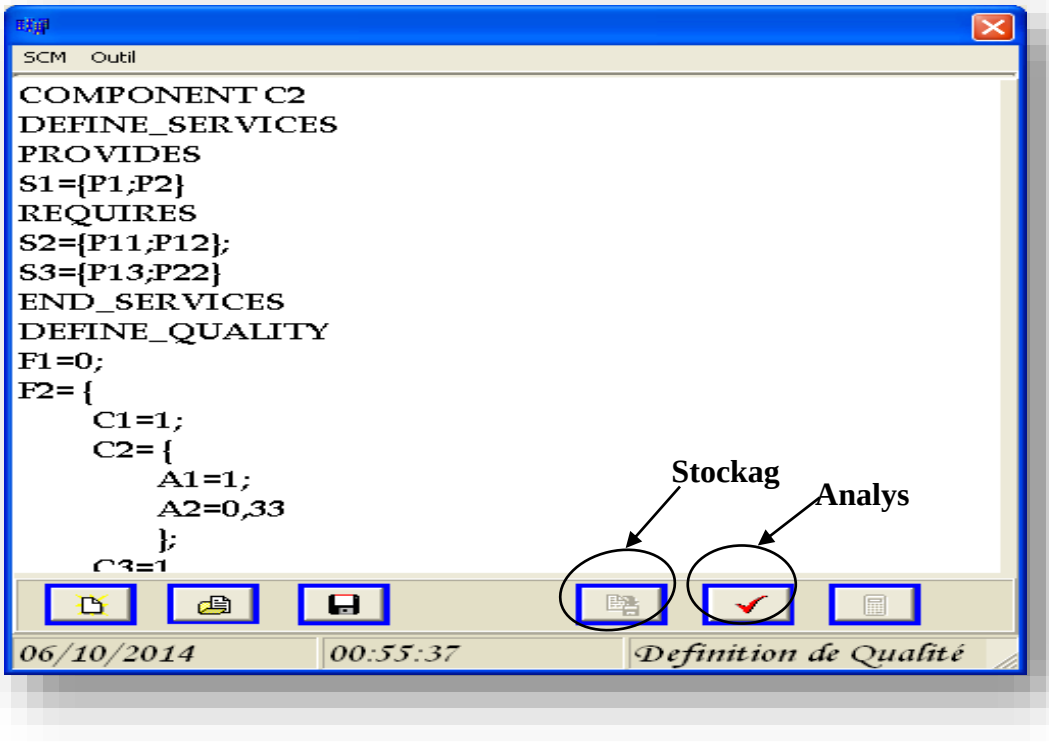
Notre environnement de simulation permet de faire trois tâches principales : Indexer, rechercher et évaluer des composants.

III.2.1. Le cas d'utilisation « Indexer » :

Dans ce cas, des caractéristiques de composant sont automatiquement extraites à partir des SCM et stockées dans la base de données. L'extraction des caractéristiques nécessite une analyse syntaxique de la métadonnée selon la grammaire présentée dans la section II.1 du chapitre IV**.** La figure suivante représente l'ordre chronologique des différentes interactions et l'implémentation du cas « indexer » :



(A)

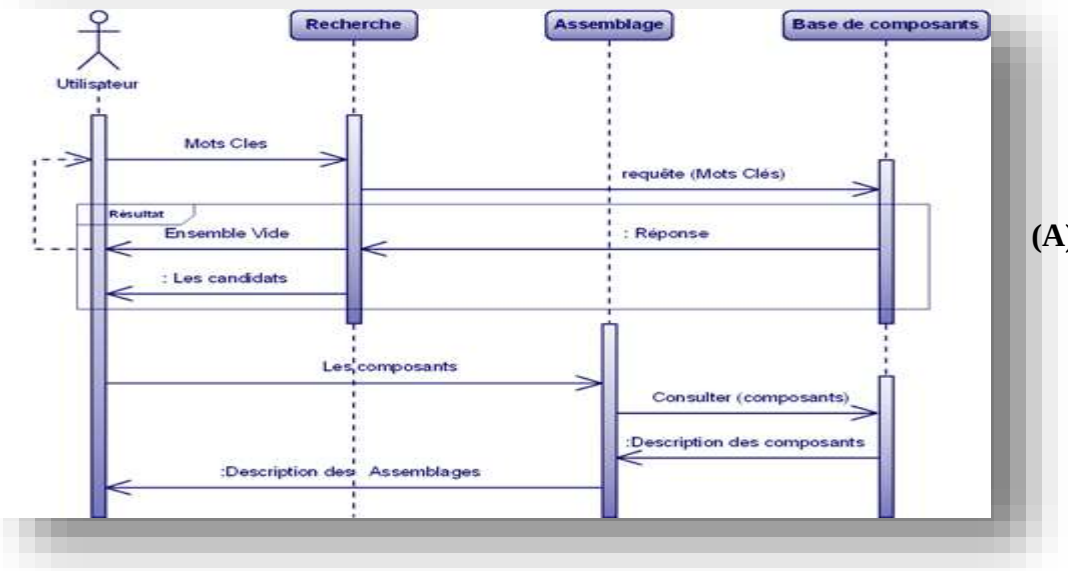


(B)

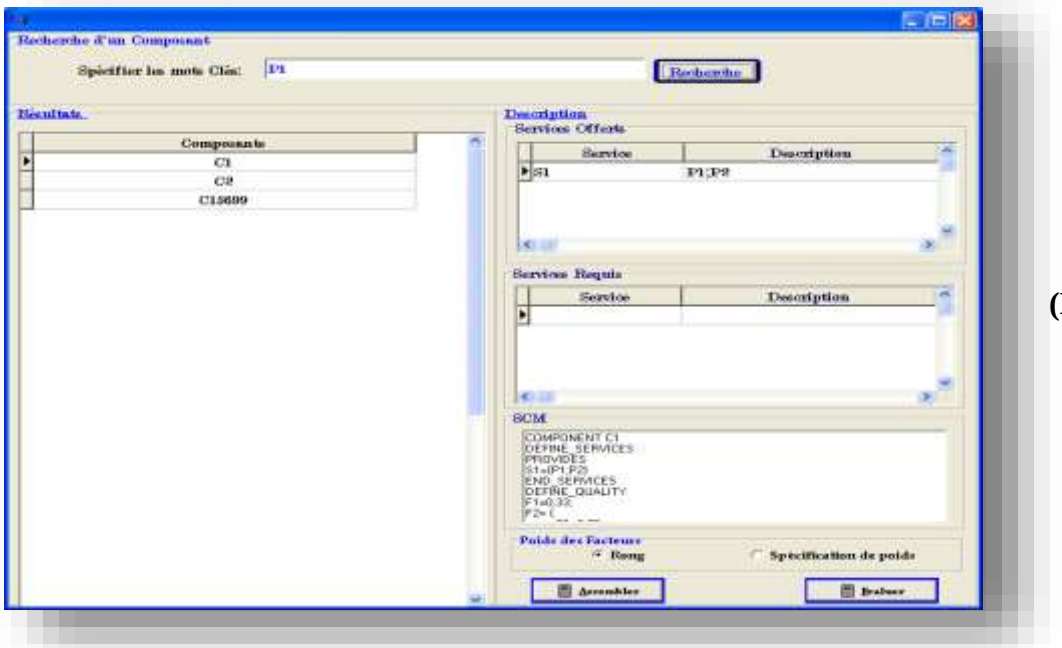
Figure V.3 : Le cas « Indexer ». (A) Diagramme de Séquence (B) Capture d'écran

III.2.2. Le cas d'utilisation « Rechercher » :

Pour faire la recherche d'un service l'utilisateur doit préciser ses besoins fonctionnels (mots clés). Le service demandé sera fourni par un composant ou par un assemblage. Le diagramme de séquence et l'implémentation du cas « Rechercher » sont présentés dans la figure V.4.



(A)

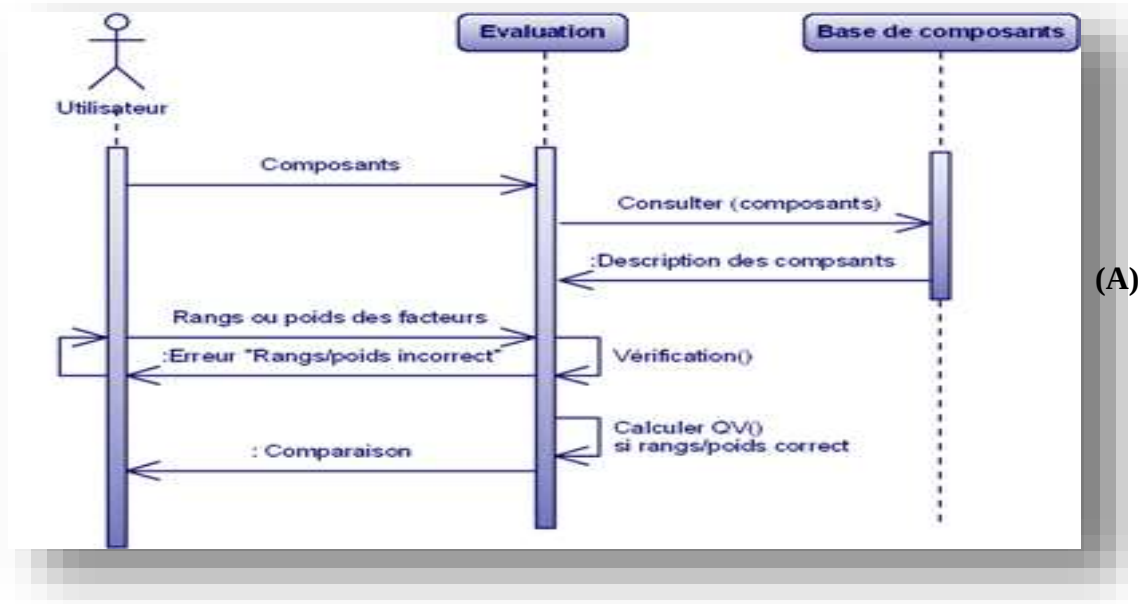


(B)

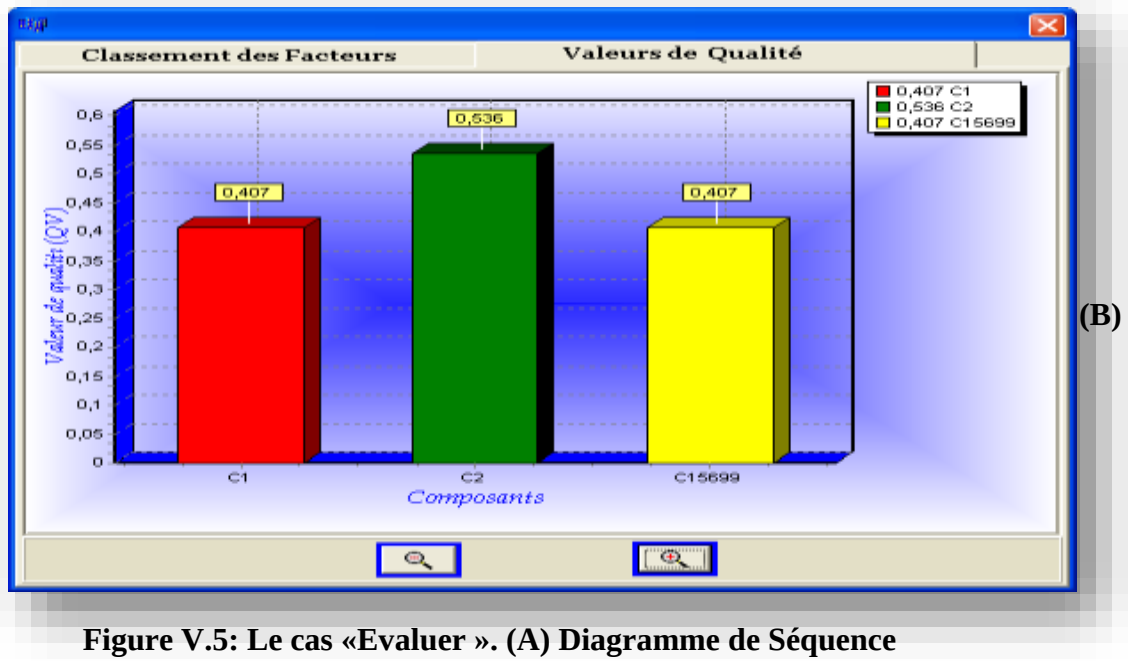
Figure V.4: Le cas «Rechercher ». (A) Diagramme de Séquence (B) Capture d'écran

III.2.3. Le cas d'utilisation Evaluer :

Après la recherche, l'utilisateur peut déclencher le cas « évaluer » pour sélectionner le meilleur. Pour cela il doit spécifier le poids des facteurs ou bien leur classement.



(A)



(B)

Figure V.5: Le cas «Evaluer ». (A) Diagramme de Séquence (B) Capture d'écran

IV. Diagramme de classe :

Les classes du système et leurs relations statiques sont représentées dans la figure V.6.

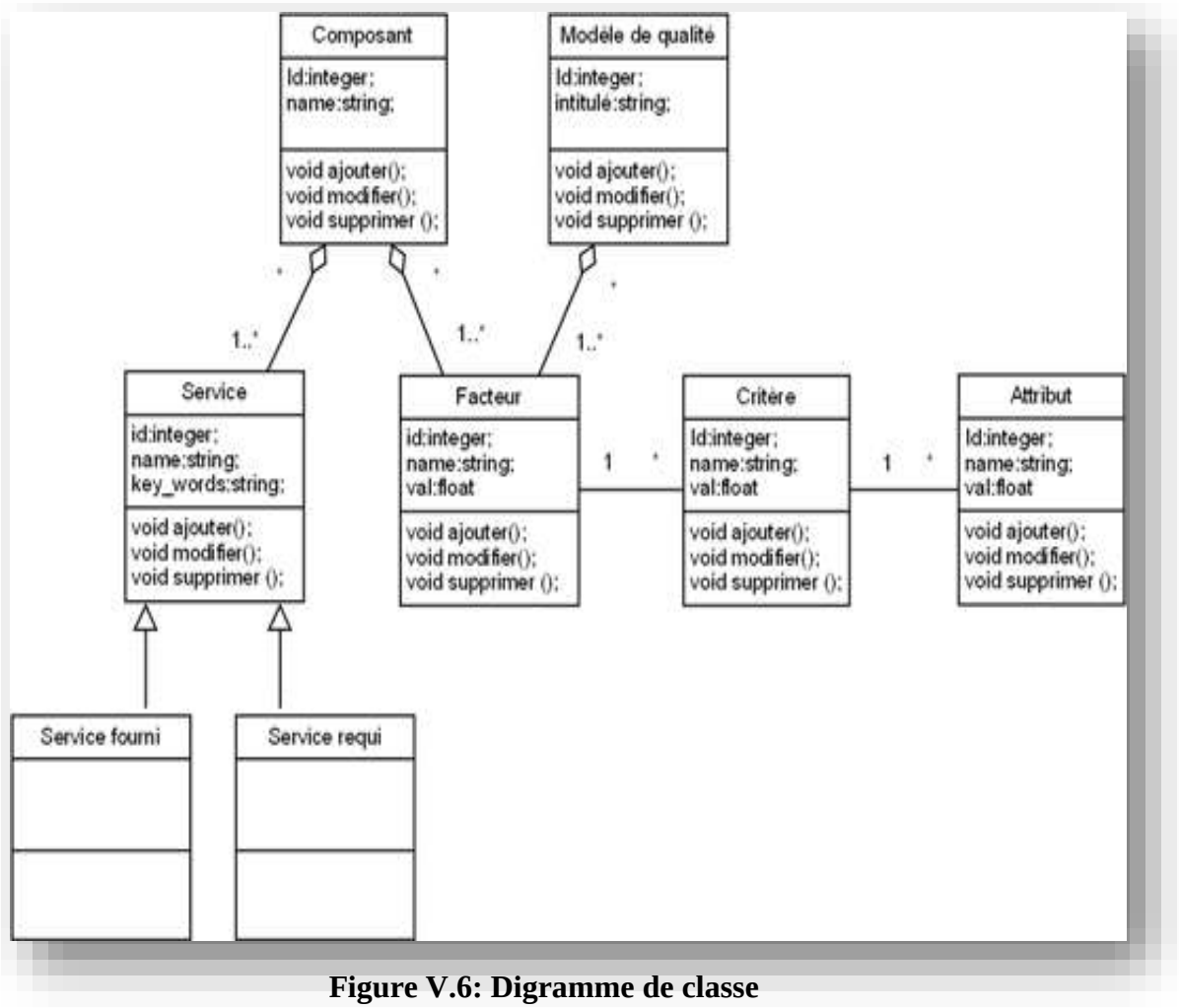


Figure V.6: Digramme de classe

Un composant (description) est constitué d'un ensemble de services et de facteurs. On peut trouver deux catégories de services : fournis ou requis.

Les facteurs sont les principaux composants d'un modèle de qualité et peuvent être représentés par un ensemble de critères qui peuvent être aussi décrits par un ensemble d'attributs.

V. Conclusion

Ce chapitre présente le Framework développé. Les modules de ce dernier implémentent les propositions présentées dans le chapitre précédent. Le Framework est constitué aussi d'un analyseur syntaxique pour la vérification de la structure des métadonnées de descriptions de composants.

Les diagrammes de séquence ont été utilisés pour représenter le comportement fonctionnel du système (vue dynamique). Alors que l'aspect statique a été modélisé par les diagrammes de cas d'utilisation et le digramme de classe.

CONCLUSION GENERALE

Conclusion Générale

La contribution principale de cette thèse est de faciliter la comparaison des composants et des assemblages. L'idée est d'accorder à chaque système une valeur numérique qui représente sa valeur globale de qualité.

Nous dressons dans cette partie un bilan des travaux de cette thèse ainsi que les perspectives.

Dans ce cadre, plusieurs outils ont été proposés : une métadonnée « SCM » fournissant les informations nécessaires à la sélection et à l'évaluation des composants, des processus permettant l'évaluation automatique des composants logiciels et des assemblages et enfin un Framework pour la simulation de la qualité.

1. Software Component Metadata (SCM) :

Après l'étude des différents langages de description d'architecture ou de spécification de la qualité, nous avons constaté qu'il n'existe pas un outil qui décrit les besoins fonctionnels et non fonctionnels en même temps. Pour cela nous avons proposé « SCM », métadonnées pour la description des composants et des assemblages. Nous avons aussi donné l'ensemble des expressions régulières qui permettent de générer ces métadonnées.

L'avantage de SCM est qu'elle s'intéresse aux besoins fonctionnels et non fonctionnels et elle est applicable à n'importe quel modèle de qualité.

2. Software Component Evaluation Processus (SCEP):

Ce processus évalue la qualité d'un composant. Il attribut à chaque composant une valeur de qualité globale qui dépend de la classification de ses caractéristiques de qualité. Nous avons prouvé à travers des exemples l'importance de cette valeur.

SCEP offre à l'utilisateur plusieurs options de recherche:

1. Recherche Exact,
2. Meilleur parmi,
3. ou en spécifiant seulement la valeur de qualité désirée.

3. Software Component Assembly Evaluation Processus (SCAEP) :

Ce processus évalue la valeur globale de la qualité d'un système à partir des valeurs de ses composants.

La phase principale de ce processus est la production du vecteur AS pour chaque système. Dés que ce vecteur est calculé, le système est considéré comme un composant et le processus SCEP est appliqué pour produire la valeur globale de qualité.

4. Environnement de simulation :

Pour valider l'approche proposée, en plus des outils cités, un environnement de simulation de qualité pour les applications basées sur l'assemblage de composants logiciels a été développé. Ce prototype implémente les propositions de ce travail. Il est constitué de trois modules : le premier (Gestion de SCM) permet d'analyser les métadonnées de description de composants, d'extraire les informations pertinentes et de stocker les composants dans la base de composants, le deuxième module recherche dans la base les composants qui peuvent répondre aux requêtes des utilisateurs et le troisième module permet l'évaluation de la qualité des composants et des assemblage.

Cet environnement permet d'atteindre l'objectif de composer et de choisir le meilleur assemblage parmi plusieurs offerts.

Le concept « valeur globale de qualité » introduit, facilite la comparaison des assemblages candidats, réduit la complexité des processus de sélection en évitant les comparaisons paire à paire des caractéristiques de qualité, et enfin, il est facile à comprendre et à implémenter.

Les expérimentations menées dans le cadre de l'approche proposée et les paramètres de choix de l'utilisateur et leur ordre permettent d'affirmer que l'assemblage proposé est le plus adéquat.

Comme perspectives nous envisageons :

1. **Extensions du format de description :**

Les métadonnées SCM, décrivent les besoins fonctionnels et non fonctionnels d'un composant et donnent une image globale des assemblages, mais la vérification syntaxique et sémantique n'est pas traitée.

Une extension possible de SCM consisterait donc à insérer des analyseurs syntaxiques et sémantiques, c'est-à-dire, le développement d'un nouvel ADL.

2. Normalisation des valeurs des attributs :

Dans quelques modèles de qualité les attributs peuvent avoir des valeurs de types différents (ratio, booléen ou dénombrement).
SCEP et SCAEP utilisent que des valeurs numériques entre 0 et 1, la normalisation des valeurs des attributs est nécessaire dans ces cas.

ANNEXE : METRQUES DE QUALITE

Cette partie donne les différents règles de calcul des attributs de qualité selon le modèle ISO-9126/3 [24] .

I) Functionality metrics

Attribut name	Measurement, formula and data element computations	Interpretation of measured value
Suitability metrics		
Functional adequacy	X=1-A/B A= Number of functions in which problems are detected in evaluation B= Number of functions checked	0 <= X <= 1 The closer to 1, the more adequate.
Functional implementation completeness	X=1-A/B A=Number of missing functions detected in evaluation. B=Number of functions described in requirement specifications	0 <= X <= 1 The closer to 1, the more complete.
Functional implementation coverage	X=1-A/B A= Number of incorrectly implemented or missing functions detected. B= Number of functions described in requirement specifications.	0 <= X <= 1 The closer to 1, the more correct.
Functional specification stability (volatility)	X=1-A/B A=Number of functions changed during development life cycle phases B=Number of functions described in requirement specifications	0 <= X <= 1 The closer to 1 the more stable.
Accuracy metrics		
Computational Accuracy	X=A/B A= Number of functions in which specific accuracy requirements had been implemented, as confirmed in evaluation. B= Number of functions for which specific accuracy requirements need to be implemented.	0 <= X <= 1. The closer to 1, the more complete.
Precision	X=A/B A= Number of data items implemented with specific levels of precision, confirmed in evaluation B= Number of data items that require specific levels of precision	0 <= X <= 1. The closer to 1, the more complete.
Interoperability metrics		
Data exchangeability (Data format based)	X=A/B A=Number of interface data formats that have been implemented correctly as in the specifications B=Number of data formats to be exchanged as in the specifications	0 <= X <= 1. The closer to 1, the more correct.
Interface consistency (protocol)	X=A/B A=Number of interface protocols implementing consistent format as in the specification confirmed in review B=Number of interface protocols to be implemented as in the specifications	0 <= X <= 1 The closer to 1, the more consistent.
Security metrics		

Access auditability	X=A/B A= Number of access types that are being logged as in the specifications B= Number of access types required to be logged in the specifications	0 <= X <= 1 The closer to 1, the more auditable.
Access controllability	X=A/B A= Number of access controllability requirements implemented correctly as in the specifications. B= Number of access controllability requirements in the specifications..	0 <= X <= 1 The closer to 1, the more controllable.
Data corruption prevention	X=A/B A= Number of implemented instances of data corruption prevention as specified confirmed in review. B= Number of instances of operation/access identified in requirements as capable of corrupting/destroying data	0 <= X <= 1 The closer to 1, the more complete.
Data encryption	X=A/B A=Number of implemented instances of encryptable/decryptable data items as specified confirmed in review B= Number of data items requiring data encryption/decryption facility as in specifications	0 <= X <= 1 The closer to 1, the more complete.
Compliance metrics		
Functional compliance	X=A/B A= Number of correctly implemented items related to functionality compliance confirmed in evaluation B= Total number of compliance items	0 <= X <= 1. The closer to 1, the more compliant.
Intersystem standard compliance	X=A/B A= Number of correctly implemented interfaces as specified, confirmed in review B= Total number of interfaces requiring compliance	0 <= X <= 1. The closer to 1, the more compliant.

II) Reliability metrics

Metric name	**Measurement, formula and data element computations**	**Interpretation of measured value**
Fault detection	X=A/B	0 <= X
.	A=Absolute number of faults detected in review	0 <= X <= 1.
	B=Number of estimated faults to be detected in review (using past history or reference model)	A high value for X implies good product quality,..
Fault removal	X=A/B A=Number of corrected faults design/coding B= Number of faults detected in review	0 <= Y <= 1 The closer to 1, the better. (more faults removed)
Test adequacy	X=A/B A=Number of test cases designed in test plan and confirmed in reviewB= Number of test cases required	0 <= X Where X is greater the better adequacy
Recoverability metrics		

Restorability	X=A/B A=Number of implemented restoration requirements confirmed in review B=Number of restoration requirements in the specifications..	0 <= X <= 1 Where X is greater, the better restorability
Restoration Effectiveness	X=A/B A=Number of implemented restoration requirements meeting target restore time B=Number of restoration requirements with specified target times	0 <= X <= 1 Where X is greater, the better effectiveness
Reliability compliance metrics		
Reliability compliance	X=A/B A= Number of correctly implemented items related to reliability compliance confirmed in evaluation B= Total number of compliance items	0 <= X <= 1. The closer to 1, the more compliant.

III) Usability Metrics

Understandability metrics		
Completeness of description	X= A/B A= Number of functions (or types of functions) described in the product description B= Total number of functions (or types of functions)	0<=X<=1 The closer to 1 the more complete
Demonstration capablity	X=A/B A= Number of functions demonstrated and confirmed in review. B= Total number of functions requiring demonstration capability	0<=X<=1 The closer to 1 the more capable.
Evident functions	X= A/B A= Number of functions (or types of functions) evident to the user. B= Total number of functions (or types of functions)	0<=X<=1 The closer to 1 the better
Function understandability	X= A/B A= Number of user interface functions whose purpose is understood by the user B= Number of user interface functions.	0 <= X <= 1 The closer to 1, the better.
Learnability metrics		
Completeness of user documentation and/or help facility	X= A/B A= Number of functions described B= Total of number of functions provided	0 <= X <= 1 The closer to 1, the more complete.
Operability metrics		
Input validity checking	X=A/B A=Number of input items which check for valid data B=Number of input items which could check for valid data	0 <= X <= 1 The closer to 1, the better.
User operation cancellability	X=A/B A=Number of implemented functions which can be cancelled by the user B= Number of functions requiring the precancellation capability	0 <= X <= 1 The closer to 1, the better cancellability

User operation Undoability	X=A/B A=Number of implemented functions which can be undone by the user B= Number of functions.	0 <= X <= 1 The closer to 1, the better undoability
Customisability	X=A/B A=Number of functions which can be customised during operation B=Number of functions requiring the customization capability	0 <= X <= 1 The closer to 1, the better customisability
Physical accessibility	X=A/B A=Number of functions which can be customised B=Number of functions	0 <= X <= 1 The closer to 1, the better physical accessibility
Operation status monitoring capability	X=A/B A=Number of functions having status monitoring capability B=Number of functions that are required to have monitoring capability.	0 <= X <= 1 The closer to 1, the better monitoring capability
Operational consistency	X=1 - A/B A=Number of instances of operations with inconsistent behaviour B=Total number of operations	0 <= X <= 1 The closer to 1, the more consistent
Message Clarity	X=A/B A=Number of implemented messages with clear explanations. . B=Number of messages implemented	0 <= X <= 1 The closer to 1, the more clear.
Interface element clarity	X=A/B A=Number of interface elements which are self-explanatory. B=Total number of interface elements	0 <= X <= 1 The closer to 1, the more clear.
Operational error recoverability	X=A/B A=Number of functions implemented with user error tolerance B=Total number of functions requiring the tolerance capability	0 <= X <= 1 The closer to 1, the more recoverable.

IV) Maintainability metrics

Analysability metrics		
Activity recording	X=A/B A=Number of implemented data login items as specified confirmed in review B=Number of data items to be logged defined in the specifications	0 <= X <= 1 The closer to 1, more data provided to record system status.
Readiness of diagnostic function	X=A/B A=Number of implemented diagnostic functions as specified confirmed in review . B=Number of diagnostic functions required	0 <= X The closer to 1, the better implementation of diagnostic functions..
Changeability metrics		
Change recordability	X=A/B A=Number of changes in functions/modules having change comments confirmed in review B=Total number of functions/modules changed from original code.	0 <= X <= 1 The closer to 1, the more recordable..
Stability metrics		
Change impact	X=1-A/B A=Number of detected adverse impacts after modifications B=Number of modifications made	0 <= X <= 1 The closer to 1, the better.

Modification impact localization	X=A/B A=Number of affected variable data by modification, confirmed in review B=Total number of variables	0 <= X <= 1 The closer to 0, the lesser impact of modification.
Testability metrics		
Completeness of built-in test function	X=A/B A=Number of implemented built-in test function as specified confirmed in review B=Number of built-in test function required	0 <= X <= 1 The closer to 1, the more complete.
Autonomy of testability	X=A/B A=Number of dependencies on other systems for testing that have been simulated with stubs B= Total number of test dependencies on other systems	0 <= X <= 1 The closer to 1, the better.
Test progress observability	X=A/B A=Number of implemented checkpoints as specified confirmed in review B=Number of designed checkpoints	0 <= X <= 1 The closer to 1, the better.

VI) Portability metrics

Adaptability metrics		
Adaptability of data structures	X=A/B A=Number of data structures which are operable and has no limitation after adaptation, confirmed in review B=Total number of data structures requiring adaptation capability	0 <= X <= 1 The closer to 1, the better.
Hardware environmental adaptability	X=A/B A=Number of implemented functions which are capable of achieving required results in specified multiple H/W environment as specified, confirmed in review B=Total number of functions with H/W environment adaptation capability requirements	0 <= X <= 1 The closer to 1, the better.
Organisational environment adaptability)	X=A/B A=number of implemented functions which are capable of achieving required results in specified multiple organizational and business environment as specified, confirmed in review B=Total number of functions with organizational environment adaptation capability requirements	0 <= X <= 1 The closer to 1, the better.
Installability metrics		
Ease of Setup Re-try	X=A/B A=Number of implemented retry operations for setup, confirmed in review B=Total number of setup operations required	0 <= X <= 1 The closer to 1, the easier.
Installation effort	X=A/B A=Number of automated installation steps confirmed in review B=Number of installation steps required	0 <= X <= 1 The closer to 1, the better.
Installation flexibility	X=A/B A=Number of implemented customizable installation operation as specified confirmed in review B=Number of customizable installation operation required	0 <= X <= 1 The closer to 1, the more flexible.
Replaceability metrics		
Continued use of data	X=A/B A=Number of software data items that continue to be used as specified after replacement, confirmed in evaluation. B=Number of old data items required to be used from old software	0 <= X <= 1 The closer to 1, the better.

Function inclusiveness	X=A/B A=Number of functions covered by new software that produces similar results, confirmed in review B=Number of functions in old software	0 <= X <= 1 The closer to 1, the better.
Portability compliance metrics		
Portability compliance	X=A/B A= Number of correctly implemented items related to portability compliance confirmed in evaluation B= Total number of compliance items	0 <= X <= 1 The closer to 1, the more complete.

REFERENCES

BIBLIOGRAPHIQUES

Références Bibliographiques

[1]C.Szyperski, C. Pfister, **WCOP'96 Workshop Report**, Workshop Reader ECOOP'96. Linz, Austria, Juin 1996.

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

[3]Humberto CERVANTES, **Vers un modèle à composants orienté services pour supporter la disponibilité dynamique**, Rapport de thèse, Université Joseph Fourier , GRENOBLE I, Juin 2004.

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

[5]Oualid KHAYATI. **Modèles formels et outils génériques pour la gestion et la recherche de composants,** Thèse de doctorat ; Institut National Polytechnique de Grenoble.2005.

[6]Nabila SALMI, **Analyse de performances des systèmes basés composants**, Thèse de doctorat, Université de Savoie,2008.

[7]Ivica Crnkovic, Stig Larsson, and Michel Chaudron. **Component-based development process and component lifecycle**. In 27th International Conference on Information Technology Interfaces (ITI), Cavtat, Croatia, June 2005. IEEE.

[8]Projet ACCORD, « **Modèle abstrait d'assemblage de composants par contrats**» Livrable 1.4, Juin 2003.

[9]Antoine Beugnard, **«Contributions à l'étude de l'assemblage de logiciels»,** Habilitation à Diriger des Recherches, Université de Bretagne Sud, 2005.

[10]Nicolas Desnos,Christelle Urtado,Sylvain Vauttier, Marianne Huchard, « **Assistance à l'architecte pour la construction d'architectures à base de composants**», LMO 2006 ,2006.

[11]OMG. **CORBA component model, v4.0**. Document formal, April 2006.

[12]E. Bruneton and T. Coupaye and J.B. Stefani. **The Fractal Component Model**, version2.0-3. Technical report, ObjectWeb consortium, February 2004.

[13]M. Beisiegel and H. Blohm and D. Booz. **SCA Service Component Architecture - Assembly Model Specification**, version 1.0. Technical report, Open Service Oriented Architecture collaboration (OSOA), 2007.

[14]Pierre Marie Oum Sack, **Contribution à l'étude de la qualité du logiciel Approche à base d'apprentissage automatique et de transformation de modèles,** Thèse de Doctorat, l'Université du Littoral Côte d'Opale, 2009.

[15]ITU-T Recommandations. **Terms and Definitions Related to the Quality of Service and Network Performance including Dependability**. August 1994.

[16] Augustin RADU, **Evaluation de la Qualité de Service par l'utilisateur final dans les systèmes mobiles**, thèse de doctorat, Université de Marne-La-Vallée 2004.

[17] Alexandre.A et Eduardo.S et Silivio.R, **Quality Attributes for a Component Quality Model**, Federal University of Pernambuco and C.E.S.A.R,2006.

[18]ISO/IEC 25021, "**Software Engineering–Software Product Quality Requirements and Evaluation (SQuaRE) – Quality measure elements**", 2011.

[19]N. Bawane and C. V. Srikrishna, **A Novel Method for Quantitative Assessment of Software Quality**, International Journal of Computer Science and Security, vol. 3, no. 6, **2010.**

[20]ISO/IEC, IS 9126-1, **Software Engineering–Product Quality–Part 1: Quality Model**, Technical report, ISO/IEC JTC1/SC7, 2001.

[21]ISO/IEC 9126, **JTC1/SC7, Information Technology–Software Product Quality**", International Standardization Organization, **1991.**

[22]ISO/IEC 25000 "JTC1/SC7, **Software Engineering - Software product Quality Requirements and Evaluation (SQuaRE).** International Standardization Organization, 2005.

[23]Blanca Gil, Witold Suryn, **The analysis and evaluation of ISO/IEC9126–3 internal quality measures applicability: state-of-the-art 2006.** Technical report, École de technologie supérieure, Québec, 2006.

[24]ISO/IEC, IS 9126-3, **Software engineering –Product quality – Part 3: Internal metrics,** Technical report, ISO/IEC JTC1/SC7, 2002.

[25]Hafedh Mili, Fatma Mili, and Ali Mili. **Reusing software : Issues and research directions**. IEEE Transactions On Software Engineering, 21 (6) :528.562, June 1995.

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

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

[28]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.

[29]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.

[30]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.

[31]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.

[32]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.

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

[34]Chou S.C., Chen J.Y., Chung C.G, **A behavior-based classification and retrieval technique for object oriented specification reuse**, Software – Practice and Experience, *vol* 26 issue 7 , pages 815-832, July 1996.

[35] Assia AIT ALI SLIMANE, **Méthode de sélection intentionnelle des services, en fonction des exigences non-fonctionnelles des agents métier**, Thèse de doctorat, Université Paris I – Panthéon-Sorbonne,2012.

[36]HENRIET .L, **Systèmes d'évaluation et de classification multicritère pour l'aide à la décision « Construction de modèles et procédures d'affectation »,** Thèse de Doctorat, Université de Paris-Dauphine, Janvier 2000.

[37]McCaffrey, J., & Koski, N. **Competitive Analysis Using MAGIQ**. MSDNMagazine. Octobre 2006.

[38]McCaffrey, J.D,**Using the Multi-Attribute Global Inference of Quality (MAGIQ) Technique for Software Testing**, Information Technology: New Generations, 2009.

[40] J. Kontio, S. Chen, K. Limperos R. Tesoriero, G. Caldiera, and M. Deutsch. **A cots selection method and experiences of its use**. In Proceedings of the 20th Annual Software Engineering Workshop (NASA), Greenbelt, Maryland, USA, 1995.

[41] Feras Tarawneh, Fauziah Baharom, Jamaiah Hj. Yahaya, and Faudziah Ahmad. **Evaluation and Selection COTS Software Process: The State of the Art.** International Journal on New Computer Architectures and Their Applications (IJNCAA) vol 1 issue 2 pages: 344-357,2011.

[42]Neil Maiden and Cornelius Ncube. **Acquiring cots software selection requirements.** IEEE Transactions on Software Engineering, vol 24 issue 3, page :46-56, March 1998.

[43]Douglas Kunda. **A Social-Technical Approach to Selecting Software Supporting COTSBased Systems**. Doctorate thesis, University of York, UK, 2001.

[44]C. Alves and J.Castro. Cre : A **systematic method for cots component selection**. Brazilian Symposium on Software Engineering, Rio de Janeiro, Brazil, October 2001.

[45]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 »,** Thèse de Doctorat, Université Ferhat Abbas de Sétif - UFAS (Algérie), Juin 2011.

[46] MEDVIDOVIC N., TAYLOR N. R., **« A Classification and Comparison Framework for Software Architecture Description Languages »,** IEEE Transactions on Software Engineering, vol. 26, no 1, 2000.

[47]Ghizlane El Boussaidi Hafedh Mili**, Les langages de description d'architectures» rapport technique,** LAboratoire de recherche sur les TEchnologies du Commerce Electronique,Université du Québec à Montréal,2006**.**

[48] F. BARBIER,C.CAUVET,M. OUSSALAH, D. RIEU et C.SOUVEYET**. Concepts clés et techniques de réutilisation dans l'ingénierie des systèmes d'information.** Hermes Science Publications, 2004.

[49]N. MEHTA, N. MEDVIDOVIC et S. PHADKE**. Towards a taxonomy of software connectors.** The 22nd International Conference of Software Engineering (ICSE'00), pages 178 – 187, Limerick, Ireland, 2000.

[50]Garlan D., Monroe R.T., Wile D.,**Acme: Architectural Description of Component-Based Systems,** Foundations of Component-Based Systems, Leavens G.T. and Sitaraman M. (eds), Cambridge University Press, pp. 47-68, 2000.

[51] Pascal André ,Gilles Ardourel ,Christian Attiogbé**. Kmelia : un modèle abstrait et formel pour la description et la composition de composants et de services,** RSTI - TSI . Objets, composants et services, pages 627 à 658,2011

[52] Rapide Design Team ,**Guide to the Rapide 1.0 Language Reference Manuals,** Program Analysis and Verification Group Computer Systems Lab Stanford University, Juillet 1997.

[53]DERDOUR Makhlouf**, Modélisation et implémentation d'un système d'information de gestion de flux multimédia pour des architectures logicielles intégrant des appareils sans fil mobiles,** thèse de Doctorat, Université BADJI Mokhtar-Annaba,2012.

[**54]** Zaremski A.M., Wing J.M., **Specification matching of software components**, In Proceedings, SIGSOFT'95: third ACM SIGSOFT Symposium on the Foundations of Software engineering, New York, NY: ACM Press, October 1995.

[55] J. Rollins ,Jeanet MWing . **Specifications as search keys for software libraries** , International Conference on Logic Programming, June 1991.

[56] Hemer D., Lindsay P., **Specification-based Retrieval Strategies for Module Reuse**, Australian Software Engineering Conference, Canberra, Australia, August 2001

[57] Antoine Rolland, **Aide à la décision multicritère et apprentissage automatique pour la classification**, Rapport technique, Laboratoire ERIC - Université Lumière Lyon II,2012.

[58] E. Triantaphyllou, B. Shu, S. Nieto Sanchez, and T. Ray ,**Multi-Criteria Decision Making: An Operations Research Approach**, Encyclopedia of Electrical and Electronics Engineering, New York, NY, Vol. 15, pp. 175-186, (1998).

[59] Xia Cai, Michael R. Lyu, Kam-Fai Wong, Roy Ko (2000), **Component-based software engineering: technologies, development frameworks, and quality assurance ,schemes**, Software Engineering Conference, APSEC 2000. Proceedings. Seventh Asia-Pacific.

[60] Zahid Javed, Ahsan Raza Sattar, Muhammad Shakeel Faridi, **Unsolved Tricky Issues on COTS Selection and Evaluation,** Global Journal of Computer Science and Technology, Vol 12 Issue 10 Version 1.0 , 2012.

[61] YAHLALI.M , CHOUARFIA Abdallah ,"**Component-Based Software Application Quality** ", Conférence International sur les modèles, Optimisation et Sécurité des Systèmes , Tiaret.Algérie,2010.

[62] YAHLALI.M , CHOUARFIA Abdallah, **"Software Component Quality Evaluation ",** International Conference on Advanced Aspects of Software Engineering (ICAASE'14),Université de Constantine-Algerie,2014.

[63] Pascal André ,Gilles Ardourel ,Christian Attiogbé, **"Spécification d'Architecture en Kmelia : Hiérarchie de connexion et composition",** 1re Conférence francophone sur les Architectures Logicielles CAL 2006, Nantes. France, 2006

[64] YAHLALI.M , CHOUARFIA Abdallah, « **SCAE: Software Component Assembly Evaluation** », International Journal of Software Engineering and Its Applications, Vol.8, No.3, pp.255-268,2014.

[65] YASSAD Lamia « **Sélection Sémantique de composants logiciels basées sur des critère de qualité de service** », Thèse de doctorat, Université de Constantine-Algerie,2012

[66] YAHLALI.M , CHOUARFIA Abdallah, "**Towards a software component assembly Evaluation**", IET software, doi: 10.1049/iet-sen.2014.0001,2014.