Un mécanisme de sélection de composants logiciels

Bart George, Régis Fleurquin, Salah Sadou, Houari Sahraoui

Un mécanisme de sélection de composants logiciels

Bart George*, Régis Fleurquin*, Salah Sadou*, Houari Sahraoui**

**Laboratoire Valoria*
Université de Bretagne Sud
Campus de Tohannic
F-56000 Vannes Cedex
{george,fleurqui,sadou}@univ-ubs.fr

***Laboratoire GEODES*
Université de Montréal
C.P. 6128 Succursale Centre-ville
Montréal (QC) CANADA H3C3J7
sahraouh@iro.umontreal.ca

RÉSUMÉ. *Une approche à base de composants permet de construire des applications complexes à partir de composants « sur étagère » provenant de différents marchés. L'effort de développement réside désormais dans la sélection des composants qui répondent le mieux aux besoins apparaissant lors du développement d'une application. Dans cet article, nous proposons un mécanisme permettant la sélection automatique d'un composant parmi un ensemble de candidats, en fonction de ses propriétés fonctionnelles et non fonctionnelles. Ce mécanisme a été testé sur un cas concret utilisant le marché aux composants ComponentSource.*

ABSTRACT. *Component-based software engineering proposes to build complex applications from COTS (Commercial Off-The-Shelf) organized into component markets. The main development effort is required in selection of the components that fit the specific needs of an application. In this article, we propose a mechanism allowing the automatic selection of a component among a set of candidate COTS, according to functional and nonfunctional properties. This mechanism has been tested on an example using the ComponentSource component market.*

MOTS-CLÉS : *Sélection de composants logiciels, composants sur étagère, comparaison fonctionnelle et non fonctionnelle, composant recherché, indice de satisfaction*

KEYWORDS: *Software component selection, COTS components, functional and nonfunctional comparison, target component, satisfaction index*

RSTI - L'Objet - 14/2008. Architectures logicielles, pages 139 à 163

1. Introduction

Le paradigme composant propose de construire un système à partir d'éléments faiblement couplés et pour la plupart déjà existants. Le taux de réutilisation ainsi atteint entraîne une diminution des délais et des coûts de développement (Voas, 1998). Pour faire face à la complexité croissante des applications, les entreprises sont de plus en plus obligées d'avoir recours à des composants commerciaux « sur étagère », fournis par des tierces personnes, et dont la nature même impose de repenser profondément le cycle de développement d'un logiciel (Tran *et al.*, 1997). En effet, l'usage de tels composants entraîne de fréquents allers-retours entre les phases de définition des besoins, de spécification de l'architecture de l'application, et de sélection des composants à intégrer (Bornwsword *et al.*, 2000). Il n'est plus possible de spécifier un besoin ou une architecture sans se demander s'il existe sur le marché un composant capable de satisfaire le premier ou de s'intégrer dans la seconde.

Dans ce contexte, une activité voit son importance renforcée : la sélection de composants (En *et al.*, 2006). Cette activité est sensible : une mauvaise définition des besoins associée à une mauvaise sélection des composants peut conduire à des catastrophes comme celle du Service Ambulancier de Londres en 1992 (Maiden *et al.*, 1998). Elle est de plus très coûteuse car elle impose potentiellement le parcours de marchés comportant des centaines de composants, décrits avec des formats potentiellement très différents. La sélection devient au final très consommatrice en temps, au point de menacer les gains que conférait à l'origine ce type d'approche (En *et al.*, 2006). La seule solution pour espérer maintenir ces gains est de disposer d'un mécanisme de sélection (Voas, 1998) qui soit autant que possible automatisé.

Dans cet article nous proposons un mécanisme qui permet de sélectionner, parmi une vaste bibliothèque de composants, le candidat qui répond le mieux à un besoin spécifique. Ce besoin est modélisé par un composant virtuel appelé « composant recherché ». Cet article prend la suite d'un précédent travail sur la substitution de composants logiciels (George *et al.*, 2006; George *et al.*, 2007) et l'adapte au contexte des marchés aux composants. La section 2 détaille les approches existantes ainsi que leurs limites. La section 3 présente ensuite notre approche. Enfin, dans la section 4, nous présentons une étude de faisabilité de cette approche conduite dans le marché aux composants *ComponentSource* (ComponentSource, 2005), avant de conclure.

2. Les techniques de sélection actuelles

La problématique dans laquelle s'inscrit notre approche est la suivante : étant donné une multitude de composants sur étagère, répartis dans différents marchés, comment choisir celui qui répondra le mieux à un besoin découvert lors du développement d'une application ? Dans cette section, nous allons d'abord évoquer les travaux essayant de répondre à cette question, qui sont du domaine de la sélection de composants. Ensuite, nous allons détailler les inconvénients de ces techniques, et les moyens possibles pour y répondre.

2.1. *Présentation des techniques existantes*

(En *et al.*, 2006) ont listé un grand nombre de travaux portant sur la sélection de composants. Cette étude montre que les processus proposés comportent au moins trois phases : la description du besoin sous la forme de critères d'évaluation, la hiérarchisation de ces critères et l'évaluation des candidats sur la base de ces mêmes critères. Pour leur réalisation, ces processus préconisent en général l'usage de techniques dites de prise de décision multicritère (Ncube *et al.*, 2002). Les plus utilisées sont WSM, pour *Weighted Scoring Method* (Mosley, 1992) et AHP, pour *Analytic Hierarchy Process* (Saaty, 1990). WSM consiste à utiliser la formule suivante : $score_c$=$\Sigma_{j=1}^{n}(weight_j * score_{cj})$. Le poids $weight_j$ représente l'importance du j-ème critère par rapport aux n-1 autres critères d'évaluation. Le score local $score_{cj}$ évalue le degré de satisfaction du critère numéro j par le candidat c. Le score total $score_c$ représente la valeur globale d'évaluation de c. Le meilleur candidat est celui dont le score total est le plus élevé. AHP est une technique qui aide à décrire le besoin de manière organisée, en le décomposant en un arbre hiérarchique de critères et de sous-critères d'évaluation, dont les feuilles sont les candidats à évaluer. A l'intérieur de chaque critère-nœud, on détermine l'importance de chaque sous-critère par rapport aux autres. Par exemple, le critère « performance » peut se décomposer en deux sous-critères « temps d'exécution » et « consommation de ressources », le premier ayant un poids deux fois plus élevé que l'autre. Ensuite, on peut utiliser une méthode comme WSM pour évaluer chaque candidat c en additionnant les scores locaux $score_{cj1}, ..., score_{cjn}$ à l'intérieur de chaque nœud j, et en propageant les sommes obtenues jusqu'à la racine de l'arbre pour obtenir le score total du candidat.

Nous allons maintenant examiner les contributions des processus de sélection les plus représentatifs du domaine. OTSO (Kontio, 1996) fut le premier processus consacré aux composants sur étagère. Il ajoute notamment aux trois phases décrites précédemment celle de présélection, qui sert à identifier et limiter le nombre des candidats à évaluer. PORE (Maiden *et al.*, 1998) et CRE (Alves *et al.*, 2001) sont des processus qui plaident en faveur d'une sélection réalisée progressivement. Les candidats sont filtrés progressivement. Leur nombre décroît au fur et à mesure que la description du besoin se fait de plus en plus précise. D'autres processus innovent en proposant de nouvelles étapes qui facilitent la sélection. Par exemple, les processus PECA (Comella-Dorda *et al.*, 2002) et CEP (Phillips *et al.*, 2002) proposent d'introduire une phase de planification de l'évaluation. Cette phase consiste à choisir les personnes en charge de l'évaluation et les techniques à utiliser. Dans CEP, cette planification est effectuée en fonction du nombre de candidats potentiels à analyser, ce qui permet de mesurer « l'effort d'évaluation ». Le processus CISD (Tran *et al.*, 1997), quant à lui, classe les candidats en groupes de composants qui seront sélectionnés et intégrés ensemble. On évalue chaque candidat individuellement et à l'intérieur de son groupe, en fonction de l'architecture globale de l'application. D'autres processus, pour leur part, innovent dans la définition des critères d'évaluation (Carvallo *et al.*, 2007). Par exemple, STACE (Kunda *et al.*, 1999) affirme l'importance des critères « socio-techniques » : la qualité du produit, les technologies utilisées, la réputation du fournisseur sur le

marché, etc. BAREMO (Lozano-Tello *et al.*, 2002) adapte AHP aux composants sur étagère en définissant un certain nombre de critères et sous-critères qui leur sont dédiés. RCPEP (Lawlis *et al.*, 2001) propose de donner la description la plus complète du besoin par une approche collaborative entre les vendeurs des composants, les développeurs de l'application et les utilisateurs de cette application. COTSRE (Martinez *et al.*, 2008) propose de créer des « catalogues de critères » réutilisables. CAP (Ochs *et al.*, 2000) et DesCOTS (Grau *et al.*, 2004) proposent quant à eux des catalogues prédéfinis de critères non fonctionnels inspirés du standard ISO-9126 (ISO, 2001).

2.2. *Les limites de ces techniques*

L'inconvénient majeur de tous ces processus réside dans leur manque d'automatisation. Si le score total des candidats est obtenu par le biais d'un algorithme ou d'une formule automatisable comme WSM, les scores d'évaluation locaux $score_{cj}$ pour chaque critère d'évaluation j sont, eux, estimés manuellement par le développeur, et cela pour chaque composant candidat c. Or, même en se limitant à une seule bibliothèque, ou à une section particulière d'une bibliothèque, on se retrouve fréquemment avec un nombre de candidats dépassant la centaine. A titre d'exemple, la seule section « communication internet » de *ComponentSource* (ComponentSource, 2005) comporte plus de 120 composants. Il est donc important de chercher autant que possible à automatiser ces calculs de scores locaux à la fois pour gagner en précision mais également pour gérer de manière rentable la masse souvent très importante des informations à traiter. Dans le cas contraire, un nombre élevé de candidats et de critères devient vite rédhibitoire lors d'une évaluation conduite manuellement (Ncube *et al.*, 2002).

Lorsque l'on cherche à automatiser ces calculs, on constate qu'ils ne sont pas tous de même nature. La phase de présélection fait le plus souvent l'usage d'un nombre restreint de critères assez généraux (comme les mots-clés). Les calculs locaux sont simples, mais ils s'appliquent à un nombre très important de candidats. Les techniques de recherche de composants en bibliothèque, dont l'objectif, après avoir formulé une certaine requête, est de retrouver tous les composants qui correspondent à cette requête (Mili *et al.*, 1995), peuvent être adaptées pour réaliser des outils capables de calculer ce type de score. Par exemple, la recherche par mots-clés ou la classification par facettes (Príeto-Diaz, 1991) peuvent permettre de filtrer un grand nombre de candidats sur la base d'une requête gros-grain. A l'inverse, lors de la phase d'évaluation, les critères utilisés peuvent être très nombreux, complexes et axés sur des points de détail (comparaison de signatures de méthodes, comparaison d'interfaces, comparaison de valeurs de métrique, etc.). Les calculs sont beaucoup plus complexes, car ils nécessitent d'agréger de nombreuses valeurs de nature diverse, mais ils ne sont appliqués qu'à un nombre restreint de candidats. On pourrait s'inspirer dans ce cas de techniques de comparaison très fines, telles que le sous-typage (Cardelli, 1988), l'abstraction de services (Sadou *et al.*, 2001) ou la substitution orientée composant. Par exemple, la substitution contextuelle de P. Brada (Brada, 2003) permet de déterminer

si un composant peut se substituer à un autre en fonction du contexte d'utilisation de ce dernier. D'une part, le nouveau composant doit fournir tous les services du composant à remplacer qui sont réellement utilisés par les autres composants de l'application. D'autre part, tous les services requis par le nouveau composant doivent pouvoir être fournis par les composants déjà présents dans l'application. Enfin, pour évaluer les composants également selon leurs propriétés non fonctionnelles, on peut décrire celles-ci au moyen de techniques telles que les contrats de qualité de service (Defour *et al.*, 2004; Frolund *et al.*, 1998), ou user de modèles de qualité développés spécifiquement pour les composants sur étagère (Bertoa *et al.*, 2002; Alvaro *et al.*, 2006).

Le mécanisme que nous proposons permet d'automatiser ces calculs de score en tenant compte de cet impératif de flexibilité dans le niveau de détail. Ce mécanisme adapte les résultats de travaux provenant de domaines disjoints comme la recherche et la comparaison de composants. Le concept-clé de cette approche est celui de composant recherché.

3. Sélection de composants

Dans cette section, nous allons présenter un mécanisme permettant de décrire les composants sur plusieurs niveaux et d'automatiser leur comparaison. Tout d'abord, nous introduisons le concept de composant recherché. Ce concept consiste à modéliser un besoin d'une application par un composant virtuel. Répondre à ce besoin revient à remplacer ce composant virtuel par un candidat « concret ». Une fois que ce remplacement est fait, l'architecture concrète de l'application s'en trouve éventuellement modifiée. Le concept de composant recherché décrit non seulement les propriétés fonctionnelles et non fonctionnelles attendues, mais aussi la manière dont sont traités les écarts avec les propriétés des candidats. L'exploitation d'un tel concept suppose donc d'adresser trois problèmes. Le premier réside dans le choix d'un format de description pour les composants candidats et recherchés. Le deuxième réside dans le choix d'un mécanisme de pondération pour évaluer l'importance des écarts entre propriétés candidates et recherchées. Le troisième problème réside dans le choix d'une fonction de comparaison entre un composant candidat et un composant recherché pour obtenir une mesure de leur « similitude ». Nous allons présenter successivement les solutions proposées pour résoudre ces trois problèmes.

3.1. *Notion de composant recherché*

Lors de la conception d'une application, l'étape de modélisation architecturale fait émerger un certain nombre de composants. Quelques-uns de ces composants ont été placés dans le modèle d'architecture car le concepteur sait qu'une implantation concrète de ceux-ci sont à sa disposition. Les composants restants manifestent pour leur part des fonctionnalités dont il a besoin mais dont il ne sait pas si une implantation concrète existe. Ces derniers composants peuvent être qualifiés, à ce stade, de « conceptuels ». Comme le montre la figure 1, chacun d'eux va faire l'objet d'une

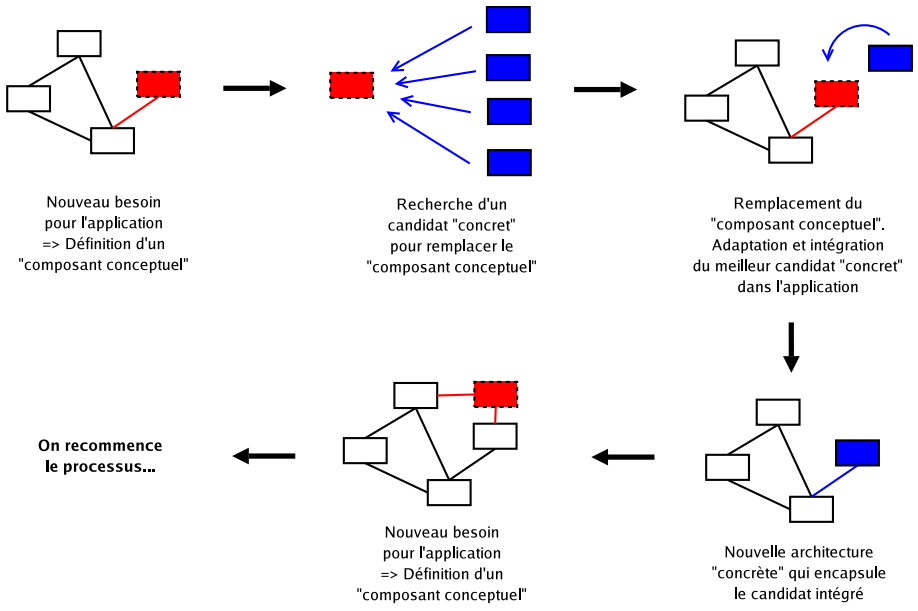

Figure 1. *Construction incrémentale de l'architecture*

recherche spécifique dont le but sera de sélectionner dans les marchés et bibliothèques un composant concret capable de se substituer intégralement à eux, ou tout du moins à même de constituer un remplaçant à coût d'adaptation et d'intégration le plus bas possible. L'intégration d'un composant concret à la place d'un composant « conceptuel » va peut-être imposer une modification du modèle d'architecture. Ces modifications peuvent concerner les composants conceptuels restants ou l'ajout de code supplémentaire pour permettre une collaboration effective avec des composants concrets. La construction d'une application à base de composants est donc incrémentale et itérative, ainsi que l'illustre la figure 1. Chaque composant conceptuel est partiellement déterminé par la partie courante de l'architecture qui est « concrète ». A chaque fois que l'un d'entre eux est remplacé par un composant concret, celui-ci modifie éventuellement par propagation l'architecture de l'application. C'est en fonction de cette nouvelle architecture que sont définis les composant conceptuels restants dont on va chercher à leur tour une implantation concrète. On est donc face à un processus de raffinement nécessitant la recherche d'un seul composant conceptuel à la fois.

Lors d'une recherche, la description du composant « conceptuel » concerné manifeste un besoin. Ce besoin est intégré dans une structure que nous appellerons « composant recherché ». Suivant le nombre des composants candidats à parcourir et/ou les résultats d'une précédente recherche, on peut être amené, soit à procéder par filtrages successifs, soit à élargir le champ de la recherche. Cette nécessité impose de disposer

pour le composant recherché d'une description potentiellement multiniveau. Au plus haut niveau, on évaluera les candidats sur la base de mots-clés contenus dans la documentation du composant recherché, ou sur la base de critères plus abstraits tels que la technologie utilisée. Au niveau de granularité le plus fin, on évaluera la signature des opérations contenues dans les interfaces des composants candidats. C'est pourquoi le composant recherché doit avoir une structure capable d'embrasser tous ces niveaux de description. Un composant recherché n'est donc pas un composant classique. Sa structure particulière est dictée par l'usage que l'on doit en faire : adapter la comparaison au nombre de candidats et au niveau de granularité exigé. Cette structure permet aussi d'utiliser dans une même comparaison des techniques issues de domaines divers.

Le simple calcul d'une différence ensembliste entre la liste des propriétés listées par le composant recherché à un certain niveau de granularité et celles offertes par un candidat concret est insuffisant pour permettre un classement pertinent. Se limiter à une telle approche revient à admettre que la présence, l'absence ou le respect partiel de deux propriétés ont les mêmes conséquences. Or l'absence ou le respect seulement partiel de certaines propriétés peut être plus ou moins pénalisant. Le cas le plus fréquent est d'ailleurs celui-là. Dans le monde des composants sur étagère en particulier, il est rare de trouver un composant candidat qui répond parfaitement à toutes les attentes du concepteur. Par conséquent, le composant recherché ne doit pas seulement englober une description de celui-ci, mais aussi une manière de traiter les disparités entre sa description et celles des candidats. Finalement, le concept de composant recherché doit présenter la structure en deux parties : la première partie offre une description à plusieurs points de vue des propriétés souhaitées ; la deuxième associe à chacune des propriétés du premier niveau une évaluation de la pénalité associée à son absence ou à son respect partiel vis-à-vis des autres propriétés.

3.2. *Le format de description des composants*

A ce jour, il n'existe pas de format de description de composants faisant consensus. Chaque bibliothèque ou marché possède sa propre manière de documenter les composants qu'elle héberge, et au sein d'une même bibliothèque peuvent cohabiter des composants décrits selon des modèles variés. C'est le cas par exemple de *ComponentSource*, qui possède des composants ActiveX, JavaBeans, .NET... Or, il est impératif que tous ces composants candidats puissent être comparés au composant recherché selon un même format de description. De plus, ce format doit être suffisemment abstrait pour englober les concepts communs à la plupart des modèles existants. C'est pourquoi nous avons établi notre propre format adapté aux composants sur étagère. Ce format est décrit au moyen d'un modèle UML (figure 2) dont nous allons présenter brièvement chacun des éléments.

3.2.1. *Les artefacts architecturaux*

Trois types d'artefacts ont été retenus : les composants, constitués de deux ensembles d'interfaces (fournies et requises), constituées chacune d'un ensemble d'opé-

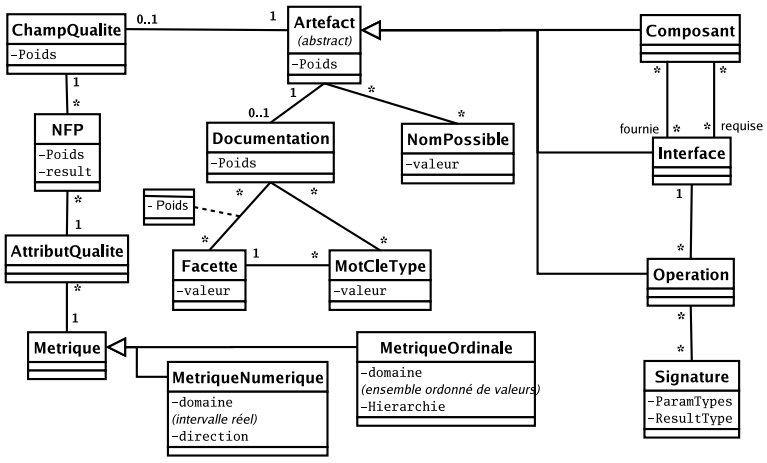

Figure 2. *Format de description pour les composants sur étagère*

rations. Cette représentation s'inspire de la définition communément admise des composants sur laquelle se base de nombreux modèles comme UML 2.0 (OMG, 2003). Comme ici nous ne nous intéressons qu'aux propriétés externes des composants et non à leur structure interne, nous ne prendrons pas en compte le cas des composants composites, c'est-à-dire contenant d'autres composants. Les interfaces fournies du composant recherché regroupent l'ensemble des services attendus par le concepteur pour les composants concrets déjà présents dans son application. L'ensemble des interfaces requises représente ce dont le composant a besoin pour fonctionner. Soit c'est déjà inclus dans l'architecture concrète de l'application, soit c'est quelque chose que le concepteur a prévu. En tout cas, il sait qu'il peut compter dessus.

A chaque opération est associée non pas une, mais plusieurs signatures. Il est en effet utile d'en prévoir plusieurs afin d'augmenter la performance lors de la recherche d'un service. Une signature $S = ParamTypes \rightarrow ResultType$ détaille les types des paramètres $ParamTypes = (\tau_1, ..., \tau_n)$ ainsi que le type du résultat $ResultType$. Si l'on prend l'exemple d'une opération de création de dossier, elle pourrait être proposée avec la signature $string \rightarrow void$. C'est le cas, par exemple, pour l'opération $MakeDirectory$ fournie par la version ActiveX du composant FTP de *PowerTCP*. Ce composant appartient à la section « communication internet » de *ComponentSource* (ComponentSource, 2005). Le paramètre $string$ est alors le nom du dossier à créer. Mais une autre signature pourrait être $string \rightarrow boolean$; le résultat de type $boolean$ indiquant si l'opération est un succès ou un échec. C'est le cas de l'opération $CreateDirectory$ du composant *FTPWizard*, lui aussi appartenant à la section « communication internet » de *ComponentSource*. Pour des raisons de cohérence, on imposera qu'une même opération ne contienne pas deux fois la même signature.

3.2.2. *Informations associées aux artefacts*

Deux ensembles d'informations sont communs à tous les artefacts. Le premier est l'ensemble de ses noms possibles. Cet ensemble est introduit car un même artefact peut être proposé sous des vocables parfois très différents. A titre d'exemple, une opération de téléchargement peut avoir pour nom $Download$ ou $GetFile$, comme c'est le cas, respectivement, pour les composants FTP de *ComponentSpace* et de *Xceed*, disponibles sur le site de *ComponentSource* (ComponentSource, 2005). Pour des raisons de cohérence, on imposera qu'un ensemble de noms possibles ne contienne pas deux fois le même nom. Le second ensemble d'informations tient lieu de documentation pour l'artefact. Chacune des informations contenues dans cette documentation est appelée « mot-clé typé ». Il s'agit d'une paire de chaînes de caractères ($facette$, $valeur$). La valeur du mot-clé typé est positionnée dans un certain domaine d'interprétation que l'on nomme « facette ». Il est possible d'associer à un artefact une documentation stipulant que son concepteur est la société NBOS et qu'il a été développé avec la technologie EJB. Il suffit pour cela de lui associer une documentation comportant deux mots-clés typés. Le premier aura pour facette « Publisher » et pour valeur « NBOS ». Le deuxième aura pour facette « Technology » et pour valeur « EJB ».

Il est possible d'associer d'autres informations aux artefacts, en particulier sémantiques. Cependant, l'objectif premier de notre approche était d'apporter une réponse concrète à une préoccupation industrielle. Pour cela, nous nous sommes placés dans le contexte des marchés aux composants tels que *ComponentSource*. Or, sur de tels marchés, la documentation des propriétés sémantiques des composants est très pauvre. C'est pourquoi nous n'avons pas encore abordé ces aspects pour l'instant.

3.2.3. *Propriétés non fonctionnelles associées aux artefacts*

Tout artefact peut se voir associer un champ qualité, qui regroupe l'ensemble de ses propriétés non fonctionnelles (NFP sur la figure 2). L'idée que tout artefact architectural peut avoir des propriétés non fonctionnelles est inspirée des langages de contrats de qualité de service QML (Frolund *et al.*, 1998) et QoSCL (Defour *et al.*, 2004). Ici, une propriété non fonctionnelle ou NFP représente une valeur (notée $result$) obtenue suite à la mesure du niveau d'un attribut qualité sur un artefact. Cette mesure est faite au moyen d'une métrique associée à l'attribut qualité. Cette structure s'inspire des modèles de qualité pour composants sur étagère comme CQM (Alvaro *et al.*, 2006) ou le modèle de (Bertoa *et al.*, 2002). Ces modèles étendent le standard ISO-9126 (ISO, 2001) en associant des attributs qualité et des métriques à ses caractéristiques et sous-caractéristiques. Nous avons choisi les métriques pour représenter et comparer les propriétés non fonctionnelles, car contrairement à d'autres méthodes qui se concentrent sur une propriété ou une famille particulière de propriétés comme la qualité de service, elles nous ont semblé l'outil d'évaluation le plus simple de la qualité dans son ensemble.

Il existe des standards pour les métriques, comme par exemple IEEE 1061-1998 (IEE, 1998), pour lequel une même caractéristique ou sous-caractéristique de qualité peut être mesurée par plusieurs métriques, et réciproquement. Cependant, il

se peut qu'un même attribut qualité soit mesuré par des métriques dont les types diffèrent d'un modèle à l'autre. Prenons l'exemple de deux modèles de qualité différents : celui de (Bertoa *et al.*, 2002) et CQM (Alvaro *et al.*, 2006). Les deux modèles associent parfois les mêmes attributs qualité aux sous-caractéristiques du standard ISO-9126 (ISO, 2001). Cependant, les métriques utilisées pour mesurer ces attributs sont parfois différentes d'un modèle à l'autre. Par exemple, l'attribut *Controllability* associé à la sous-caractéristique *Security* est mesuré par un pourcentage dans le modèle de Bertoa et Vallecillo, tandis qu'il est mesuré par un booléen dans CQM. Et pour mesurer l'attribut *Customizability* associé à la sous-caractéristique *Changeability*, le premier modèle utilise un entier, tandis que le second utilise un pourcentage. L'intégration du standard IEEE 1061-1998 dans le format de description suppose donc l'existence dans la littérature de données systématiques permettant de comparer les valeurs obtenues pour un même attribut qualité avec des métriques différentes. Ce n'est malheureusement pas le cas. En conséquence, dans ce format de description, nous considèrerons qu'un attribut qualité ne peut être mesuré que par une seule métrique, même si une métrique peut mesurer plusieurs attributs.

Une métrique peut être numérique ou ordinale. Cette distinction est inspirée à la fois des métriques utilisées dans les modèles de qualité pour composants (Bertoa *et al.*, 2002; Alvaro *et al.*, 2006), et du projet CLARIFI (Boegh, 2006) qui propose de différencier les différents types de métriques correspondant à leur domaine de valeurs. Le domaine d'une métrique numérique est un sous-ensemble des nombres réels (entiers, pourcentage...). Comme les modèles de qualité que nous avons étudiés ne proposent pas de métriques à valeurs négatives, nous prenons comme hypothèse que le domaine d'une métrique numérique est toujours positif. Le domaine d'une métrique ordinale est, pour sa part, un ensemble fini et totalement ordonné. Par exemple, $\{mauvais, moyen, bon, excellent\}$, ou bien le domaine booléen $\{faux, vrai\}$. Une métrique numérique possède un attribut supplémentaire appelé direction, qui permet d'interprèter le résultat d'une métrique. Les directions utilisables sont $croissante$ ou $decroissante$. Cette distinction est inspirée des langages de contrats de qualité de service tels que QML (Frolund *et al.*, 1998) ou QoSCL (Defour *et al.*, 2004). Une direction croissante signifie que la qualité est d'autant meilleure que la valeur de la métrique est élevée. Un exemple de métrique numérique de direction croissante est le nombre de messages stockés et délivrés en une seconde par un composant servant de file de messages. A l'inverse, une direction décroissante signifie que la qualité est d'autant meilleure que la valeur de la métrique est basse. Un exemple de métrique numérique de direction décroissante est le temps de réponse d'une opération. Une métrique ordinale possède un attribut supplémentaire appelé hiérarchie. La hiérarchie regroupe toutes les valeurs possibles de la métrique en leur associant une clé qui symbolise leur rang. C'est ce rang qui définit la relation d'ordre total sur le domaine de la métrique. Quand une valeur est « meilleure » qu'une autre, son rang est strictement supérieur dans la hiérarchie associée. La valeur de rang le plus élevé est égale au nombre de valeurs possibles moins 1. La valeur de rang le plus faible est égale à 0. Par exemple, si une métrique ordinale M a pour domaine {*très mauvais*, *mauvais*, *moyen*, *bon*, *très bon*}, alors la hiérarchie correspondante est $Hierarchie(M)$=[(0,

très mauvais), (1, *mauvais*), (2, *moyen*), (3, *bon*), (4, *très bon*)], et le rang de la valeur *bon* est égal à $rang(bon) = 3$.

3.3. *Traitement des disparités entre les composants candidats et le composant recherché*

Nous venons de définir le premier aspect de la structure d'un composant recherché, c'est-à-dire le format de description de ses propriétés fonctionnelles et non fonctionnelles. Nous allons maintenant aborder le deuxième aspect, qui traite des discordances entre les propriétés d'un composant recherché et celles d'un composant candidat. En effet, il faut prévoir le cas où une propriété spécifiée dans le composant recherché est absente d'un composant candidat, ou partiellement présente. Il faut également classer des candidats qui pour certains satisfont pleinement la propriété A mais pas la propriété B, et pour d'autres se retrouvent dans le cas inverse. Ce qui pose la question de savoir quelle propriété, de A ou de B, est la plus importante.

Dans le cas du format défini précédemment, nous avons vu qu'un composant décrit dans ce format contient des propriétés fonctionnelles et non fonctionnelles. Supposons que l'on compare deux composants candidats à un composant recherché selon ce format. L'un des candidats possède toutes les fonctionnalités attendues, mais est de piètre qualité. L'autre n'a pas toutes les fonctionnalités demandées dans le composant recherché, mais il est de bien meilleure qualité que le premier candidat. Lequel faut-il choisir dans ce cas ? La question se pose même quand on se limite aux seules propriétés non fonctionnelles. Par exemple, dans le contexte de la compression vidéo, il faut tenir compte à la fois de la qualité de l'image, de la qualité du son, et du taux de compression. En accordant plus d'importance à cette dernière propriété au détriment des deux autres, les candidats possédant un taux de compression élevé seront privilégiés par rapport aux candidats dotés d'une meilleure qualité d'image et de son.

Pour résoudre ce problème, nous avons choisi d'associer au format de description une pondération permettant de décrire pleinement le composant recherché. Elle consiste à attacher un poids aux éléments du format de description afin d'exprimer l'importance de chacun par rapport aux autres éléments de même niveau. Par exemple, à l'intérieur d'un champ qualité, le poids de chaque NFP représente son importance par rapport aux autres NFP. Le sens qu'on donne à ces poids ainsi que la manière de pondérer varient selon le mécanisme de pondération choisi. Lorsque chaque élément d'un composant candidat est comparé à un élément correspondant dans le composant recherché, le poids de ce dernier élément influe sur la comparaison. Il permet d'estimer l'importance réelle de cette comparaison locale dans l'évaluation globale du composant candidat.

3.4. *Indice de satisfaction entre composants*

Maintenant que nous savons décrire aussi bien le composant recherché que les composants candidats dans un format commun, nous pouvons aborder le problème de la comparaison entre deux composants décrits selon ce format. Sachant que les calculs de score total dans les techniques multicritères considèrent que le meilleur candidat est celui qui a le score le plus élevé, nous avons choisi de définir un indice de satisfaction basé sur le même principe. Cet indice est calculé de manière récursive, au moyen de plusieurs techniques de comparaison différentes. Il permet de quantifier en termes mesurables jusqu'à quel point un composant candidat est proche du composant recherché. C'est-à-dire qu'il permet de mesurer combien de propriétés fonctionnelles et non fonctionnelles le premier composant partage avec le second, dans le respect des poids assignés aux propriétés de ce dernier.

Nous allons tout d'abord présenter le principe de l'indice de satisfaction. Ensuite, nous introduirons les différentes formules utilisées pour le calcul récursif de cet indice. Nous commencerons par la formule générale utilisée au niveau composant, jusqu'aux fonctions de comparaison utilisées pour les éléments de plus bas niveau.

3.4.1. *Principe général et pondération associée*

L'indice de satisfaction renvoie une valeur réelle allant de 0 (le candidat ne possède aucune propriété recherchée) jusqu'à 1 (le candidat satisfait complètement le composant recherché). Comme nous cherchons uniquement à savoir si le candidat satisfait ou non le besoin, nous avons choisi de ne pas nous occuper des éventuels « bonus » accordés aux candidats. En d'autres termes, si un composant candidat fournit des propriétés supplémentaires non prévues dans le composant recherché, nous ne les prendrons pas en compte. De même, si le niveau d'une propriété candidate est supérieur au niveau attendu par la propriété recherchée équivalente, nous ne tiendrons pas compte de ce dépassement, et l'indice de satisfaction entre ces deux propriétés restera égal à 1. Pour sélectionner un composant candidat parmi tous ceux qui sont disponibles, on procède de la manière suivante. Tout d'abord, on calcule l'indice de satisfaction entre chaque candidat et le composant recherché. Ensuite, on choisit le candidat dont l'indice de satisfaction est le plus élevé.

Nous associons à l'indice de satisfaction une pondération par distribution. D'une part, cette pondération consiste à assigner à chaque élément un poids entre 0 et 1. D'autre part, elle consiste à partager la totalité du poids de chaque élément entre tous ses fils directs. La distribution se fait à partir du composant, qui a par défaut un poids de 1, jusqu'aux éléments de plus bas niveau. Par exemple, une interface doit partager son poids entre ses noms possibles, sa documentation, ses opérations et son champ qualité. La somme des poids de tous ces éléments est donc égale à 100 % du poids de leur interface parente. Il y a deux cas particuliers à ce partage des poids : les noms possibles et les signatures d'opérations. En effet, on cherche une seule signature ou un seul nom correct parmi tous ceux que l'on propose. Un « poids d'ensemble » sera donc assigné à l'ensemble des noms possibles de chaque artefact et l'ensemble des

signatures de chaque opération. Et chacun des noms possibles d'un même artefact se verra attribuer 100 % de ce poids d'ensemble. Il en va de même pour les signatures d'une même opération.

3.4.2. *Formule générale pour les composants*

Pour calculer l'indice de satisfaction entre un composant candidat C_1 et un composant recherché C_0, il faut d'abord calculer récursivement les indices locaux entre chacun de leurs éléments fils respectifs. Ensuite, on pondère chacun de ces indices locaux au moyen du poids associé à l'élément recherché. Enfin, on agrège ces indices locaux pondérés. Le poids assigné à un élément E est noté $Poids(E)$. Les éléments fils qu'il faut comparer sont : les noms possibles, les documentations, les interfaces fournies, les interfaces requises et les champs qualité. Les noms possibles candidats et recherchés sont regroupés dans les ensembles respectifs N_1 et N_0. La documentation candidate et la documentation recherchée sont notées respectivement D_1 et D_0. Les champs qualité candidat et recherché sont notés respectivement QF_1 et QF_0.

En ce qui concerne les interfaces fournies, on va comparer chaque interface fournie recherchée if_0 avec l'union ensembliste de toutes les interfaces fournies candidates, notée IF_1. En effet, considérons le cas où les opérations recherchées sont réparties entre plusieurs interfaces candidates. Au lieu de choisir entre ces interfaces et se priver d'une partie de ce qu'on recherche, il vaut mieux calculer un indice unique sur l'union ensembliste des interfaces candidates. En ce qui concerne les interfaces requises, elles nécessitent un traitement particulier. Nous avons expliqué précédemment que les services requis du composant recherché sont quelque chose qui se trouve déjà dans l'architecture concrète de l'application, ou que l'utilisateur a anticipé. Mais il se peut qu'un composant candidat requière plus que prévu, c'est-à-dire des services qui ne se trouvent pas dans le composant recherché. Le traitement appliqué aux services requis consiste donc, à l'inverse des services fournis, à savoir si un élément requis candidat trouve un équivalent dans le composant recherché. Entre eux, on ne mesure plus un indice de satisfaction, mais une pénalité spécifique, notée $Penalite$. Comme pour les poids et les calculs d'indice, cette pénalité est comprise entre 0 et 1 afin de garder un calcul global cohérent. Mais contrairement aux poids et aux calculs d'indice, 0 représente la « meilleure » valeur (le composant candidat ne requiert rien de plus que le composant recherché), tandis que 1 représente la « pire valeur » (le composant requiert beaucoup trop de choses par rapport à ce qui est prévu pour le composant recherché). Pour des raisons de commodité, on imposera que tous les services requis d'un composant recherché soient regroupés dans une interface unique. Au niveau composant, on va donc calculer la pénalité entre l'unique interface requise du composant recherché (notée ir_0) et l'union ensembliste des interfaces requises du composant candidat (notée IR_1). Puis on retranchera cette pénalité à l'indice de satisfaction entre les composants. Plus cette pénalité entre interfaces requises sera élevée, moins le composant candidat sera « satisfaisant ».

L'indice de satisfaction, noté $Indice$, entre un composant candidat C_1 et un composant recherché C_0 est formellement défini par la fonction :

$$Indice(C_1, C_0) = \begin{cases} Poids(N_0) * Indice(N_1, N_0) \\ + Poids(D_0) * Indice(D_1, D_0) \\ + \Sigma(\{Poids(if_0) * Indice(IF_1, if_0) \mid if_0 \in C_0\}) \\ - Poids(ir_0) * Penalite(ir_0, IR_1) \\ + Poids(QF_0) * Indice(QF_1, QF_0) \end{cases} \quad [1]$$

Comme le composant distribue son poids entre ses éléments fils, on a : $Poids(N_0)$ + $Poids(D_0)$ + $\Sigma(\{Poids(if_0) \mid if_0 \in C_0\})$ + $Poids(ir_0)$ + $Poids(QF_0)$ = $Poids(C_0)$ = 1.

Nous allons maintenant donner le principe et la formule utilisée pour le calcul récursif qui s'applique aux éléments fils du composant. D'abord, nous détaillons l'indice de satisfaction pour les services fournis. Ensuite, nous détaillons le calcul de pénalité pour les services requis.

3.4.3. *Calcul récursif pour les éléments du composant*

Une lecture attentive du format de description proposé permet de distinguer une description de nature fortement hiérarchique. Dans ce format, un composant est décrit au moyen d'un arbre dont la racine est un artefact de type composant et dont les nœuds fils sont potentiellement : NomPossible, Documentation, Interface (fournie ou requise) et Champ qualité. Parmi eux, un nœud de type Interface peut à son tour disposer de nœuds fils de type NomPossible, Documentation, Opération et Champ qualité. L'indice de satisfaction doit donc de façon récursive comparer deux nœuds a et b d'arbres distincts en comparant leurs nœuds fils respectifs deux à deux, puis « agréger » le résultat des comparaisons des sous-nœuds pour calculer le score de similarité entre a et b. Cette démarche est la même quel que soit le type des deux nœuds à comparer. On suppose qu'à chaque élément E de l'arbre est associé un type (noté $Type(E)$) et un poids $Poids(E)$. Un élément de l'arbre peut être par exemple de type « interface », « opération », « documentation »... Pour deux éléments de type différent, l'indice de satisfaction sera nul.

Pour comparer les éléments feuilles, nous utilisons une fonction $Comp$, qui est propre à chaque type de feuille. En ce qui concerne les nœuds, le calcul d'indice est effectué de manière générique, indépendamment du type des nœuds concernés. Nous allons maintenant détailler ce calcul, d'abord pour les nœuds descendants d'interfaces fournies, ensuite pour les nœuds descendants d'interfaces requises.

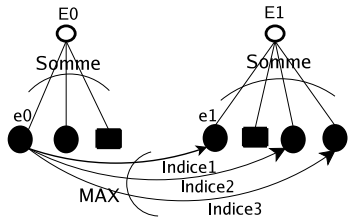

Figure 3. *Principe de l'indice de satisfaction entre deux éléments*

3.4.3.1. Indice de satisfaction pour les éléments d'interfaces fournies

Pour les éléments de l'arbre dont l'ancêtre est une interface fournie, on obtient le mode de calcul décrit à la figure 3. Pour chaque élément fils e0 de E0, on calcule l'indice de satisfaction avec chacun des fils de E1 comparable à e0 (respectivement, Indice1, Indice2 et Indice3). Le meilleur résultat est l'indice de satisfaction le plus élevé. On multiplie ce meilleur résultat par le poids de e0, puis on recommence l'opération pour tous les autres fils de E0. Enfin, on additionne ces meilleurs résultats ainsi pondérés pour obtenir l'indice total de satisfaction entre E1 et E0.

L'indice de satisfaction $Indice$ entre un élément fourni candidat E_1 et un élément fourni recherché E_0 est formellement défini par la fonction :

$$Indice(E_1, E_0) = \begin{cases} \text{. } 0 \text{ si } Type(E_1) \neq Type(E_0). \\ \text{. } Comp(E_1, E_0) \text{ si } E_0 \text{ est une feuille.} \\ \text{. } \Sigma(\{Poids(e_0) * MAX(\{Indice(e_1, e_0) \mid e_1 \in E_1\}) \mid \\ e_0 \in E_0\}) \text{ si } E_0 \text{ est un nœud.} \end{cases}$$

[2]

Selon ce calcul récursif, un élément candidat peut satisfaire plusieurs éléments recherchés différents. Par exemple, une interface fournie peut être la meilleure candidate pour deux interfaces fournies recherchées. Cependant, si un concepteur veut un composant avec un nombre précis d'interfaces ou d'opérations, il peut modéliser cette propriété par une NFP. Cette NFP serait associée à une métrique numérique telle qu'il en existe dans la littérature (Alvaro *et al.*, 2006).

3.4.3.2. Pénalité entre interfaces requises

Considérons un élément de l'arbre candidat E_1 et un élément de l'arbre recherché E_0, ces deux éléments ayant pour ancêtre une interface requise. Si E_0 et E_1 sont des nœuds, alors pour chaque fils e_1 de E_1, on va rechercher le fils e_0 de E_0 pour lequel leur pénalité multipliée par le poids de e_0 est la plus petite. On additionne

ces résultats minimaux pour obtenir la pénalité entre E_0 et E_1. Si en revanche E_0 et E_1 sont des feuilles, on mesure l'écart entre ces feuilles en calculant 1 - $Comp$. La pénalité maximale autorisée pour chaque élément requis candidat est de 1, afin d'éviter les débordements et de garantir que la pénalité totale pour l'ensemble des services requis sera comprise entre 0 et 1.

La pénalite entre un élément requis candidat E_1 et un élément requis recherché E_0, notée $Penalite$, est formellement définie par la fonction :

$$Penalite(E_0, E_1) = \begin{cases} \text{. 1 si } Type(E_0) \neq Type(E_1). \\ \text{. } 1 - Comp(E_0, E_1) \text{ si } E_1 \text{ est une feuille.} \\ \text{. } MAX(1, \Sigma(\{MIN(\{Poids(e_0) * Penalite(e_0, e_1) \mid \\ e_0 \in E_0\}) \mid e_1 \in E_1\})) \text{ si } E_1 \text{ est un nœud .} \end{cases}$$

[3]

3.4.4. *Fonctions de comparaison choisies*

Maintenant que l'indice de satisfaction a été défini pour les nœuds de l'arbre, il ne reste plus qu'à détailler les fonctions de comparaison que nous avons choisies pour chaque type de feuille, à savoir les NFP, les noms possibles, les mots-clés typés et les signatures d'opération.

3.4.4.1. Fonction de comparaison entre NFP

Soit A_0 un artefact recherché, P_0 l'une des NFP de son champ qualité, A_1 un artefact candidat de même type que A_0, et P_1 une des NFP de son champ qualité.

P_1 n'est comparable à P_0 que si les deux NFP mesurent le niveau du même attribut qualité, et dans ce cas, la métrique qu'elles utilisent sera notée M. Si M est numérique, la fonction de comparaison mesurera la correspondance entre la valeur de P_1 et celle de P_0 en fonction de la direction de M. Si M est ordinale, la fonction de comparaison mesurera la correspondance entre les rangs des valeurs de P_1 et P_0. La fonction de comparaison entre P_1 et P_0 est formellement définie comme suit :

$$Comp(P_1, P_0) = \begin{cases} \text{. 0 si } P_1 \text{ et } P_0 \text{ ne mesurent pas le même attribut qualité.} \\ \text{. } Comp_{inc}(P_1, P_0) \text{ si } M \text{ est numérique croissante.} \\ \text{. } Comp_{dec}(P_1, P_0) \text{ si } M \text{ est numérique décroissante.} \\ \text{. } Comp_{ord}(P_1, P_0) \text{ si } M \text{ est ordinale, et si} \\ rang(result_{P_0}) > 0. \\ \text{. 1 si } M \text{ est ordinale, et si } rang(result_{P_0}) = 0. \end{cases}$$

[4]

Avec :

$$Comp_{inc}(P_1, P_0) = MIN(\frac{result_{P_1}}{result_{P_0}}, 1) \quad [5]$$

$$Comp_{dec}(P_1, P_0) = MIN(\frac{result_{P_0}}{result_{P_1}}, 1) \quad [6]$$

$$Comp_{ord}(P_1, P_0) = MIN(\frac{rang(result_{P_1})}{rang(result_{P_0})}, 1) \quad [7]$$

Supposons par exemple que P_0 indique le temps d'exécution d'une opération recherchée, et que P_1 est une NFP d'une opération candidate. Si P_1 ne mesure pas le même attribut qualité que P_0, la fonction de comparaison rend 0. Sinon, elles partagent une même métrique numérique et décroissante. On applique donc $Comp_{dec}$ aux deux NFP. Supposons que le temps d'exécution recherché pour P_0 soit de 200 millisecondes. Si le temps d'exécution mesuré pour P_1 est de 300 millisecondes, la fonction de comparaison entre P_1 et P_0 rend 200/300. On estime donc que P_1 satisfait P_0 à 66,67 %. Si en revanche le temps d'exécution mesuré pour P_1 est inférieur à celui attendu pour P_0 (par exemple, 100 millisecondes), on considère que P_1 satisfait pleinement la propriété recherchée. La fonction de comparaison entre les deux NFP rend donc 1. Comme nous partons de l'hypothèse que le domaine d'une métrique numérique est toujours positif, le résultat minimum d'une comparaison entre métriques numériques est toujours 0.

Supposons maintenant que P_0 et P_1 indiquent le « niveau » d'une qualité particulière, au moyen d'une métrique ordinale de type {*très mauvais*, *mauvais*, *moyen*, *bon*, *très bon*}. Si la valeur de P_0 est fixée à *bon*, son rang est 3. Si la valeur de P_1 est égale à *mauvais*, son rang est égal à 1 et la fonction de comparaison entre P_1 et P_0 rend 1/3. Pour que la fonction de comparaison entre P_1 et P_0 rende 1, il faut que la valeur de P_1 soit au moins de rang 3. En d'autres termes, la valeur de P_1 doit être *bon* ou *très bon*. Dans le dernier cas, le fait que la valeur P_1 soit supérieure à ce qui est attendu n'est pas pris en compte, et la fonction de comparaison rend 1. Comme nous comparons des rangs plutôt que des valeurs, et que le rang le plus faible du domaine d'une métrique ordinale est toujours 0, le résultat minimum d'une comparaison entre métriques ordinales est toujours 0.

3.4.4.2. Fonction de comparaison entre noms possibles

Pour des raisons de simplicité, nous considérons que la comparaison entre noms possibles est de type « tout ou rien ». Autrement dit, pour un nom possible candidat N_1 et un nom possible recherché N_0, $Comp(N_1, N_0)$ est égal à 1 si la valeur de N_1 est égale à la valeur de N_0. Dans tous les autres cas, $Comp(N_1, N_0)$ = 0.

Supposons que l'utilisateur ait besoin d'un composant particulier, dont il connaît le nom. Dans ce cas, le composant recherché aura celui-ci comme unique nom possible.

Il faudra donc que le nom du composant candidat contienne la même valeur (sans faire la distinction entre majuscules et minuscules, ou présence d'espaces et traits d'union).

3.4.4.3. Fonction de comparaison entre mots-clés typés

De la même manière que pour les noms possibles, nous considérons que la comparaison entre mots-clés de la documentation est de type « tout ou rien ». Autrement dit, pour un mot-clé typé candidat K_1 associé à une facette F_1 et un mot-clé typé recherché K_0 associé à une facette F_0, $Comp(K_1, K_0)$ est égal à 1 si la valeur de F_1 est égale à la valeur de F_0, et si la valeur de K_1 est égale à la valeur de K_0. Dans tous les autres cas, $Comp(K_1, K_0)$ = 0.

Supposons que l'utilisateur ait besoin d'un composant d'une technologie particulière, par exemple EJB, la documentation du composant recherché comportera un mot-clé typé contenant la facette « Technology » et la valeur « EJB ». Il faudra alors que la documentation du composant candidat contienne un mot-clé typé identique.

3.4.4.4. Fonction de comparaison entre signatures d'opération

Afin d'automatiser la comparaison des signatures d'opération, nous avons décidé d'adapter le sous-typage de signature (Cardelli, 1988). Par conséquent, nous considérons qu'une signature S_1=$ParamTypes_1 \rightarrow ResultType_1$ est sous-type d'une signature S_0=$ParamTypes_0 \rightarrow ResultType_0$ si et seulement si $ParamTypes_0$ est sous-type de $ParamTypes_1$ et $ResultType_1$ est sous-type de $ResultType_0$. Par conséquent $Comp(S_1, S_0)$ est égal à 1 si S_1 est sous-type de S_0, et à 0 sinon.

Considérons par exemple la signature recherchée $S_0 : float \rightarrow int$. Si S_1 a la même signature, la fonction de comparaison rend 1. Il en va de même si $S_1 : float \rightarrow float$, car dans ce cas S_1 est sous-type de S_0. Si par contre $S1 : boolean \rightarrow int$, cette signature n'est pas sous-type de S_0 et la fonction de comparaison rend 0.

D'autres techniques existent pour comparer les signatures d'opération, en particulier le matching de signatures défini par (Zaremski *et al.*, 1995), qui est plus flexible que le sous-typage. Cependant, toutes les relations de matching ne sont pas automatisables car elles nécessitent une approche collaborative.

4. Exemple concret

Dans cette section, nous présentons le résultat d'une expérimentation entreprise sur un marché aux composants réel. Cette expérimentation montre la faisabilité pratique et l'intérêt de notre approche de sélection ainsi que des outils qui la supportent. Nous nous sommes placés dans le contexte suivant : un concepteur recherche pour son application un composant dédié au FTP (*File Transfer Protocol*) parmi tous les candidats disponibles dans la section « communication internet » du marché aux composants *ComponentSource*[1]. Cette section contenait 131 composants au moment où

1. Pour plus d'informations : http ://www.componentsource.com/index.html

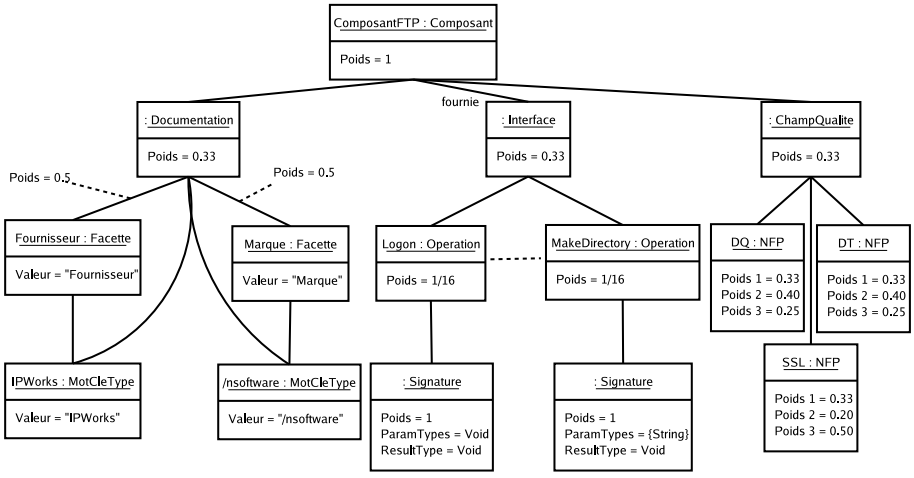

Figure 4. *Exemple de composant recherché*

nous avons effectué nos tests. Depuis cette base documentaire, nous avons produit pour chaque composant de cette section une description exprimée dans notre format. C'est sur cette documentation traduite que le processus de sélection dont nous présentons les résultats a été réalisé. Cette traduction dans notre format de description, que nous avons réalisée à la main, peut être elle-même automatisée en utilisant des techniques de transformation de modèles. C'est le sujet d'un travail en cours.

4.1. *Spécification du composant recherché*

La figure 4 montre le composant recherché entièrement spécifié selon notre format de description. Le concepteur a besoin d'un composant compatible avec la suite logicielle *IP*Works !* vendue par */n software*. D'un point de vue fonctionnel, cela suppose que le composant recherché fournisse une interface dotée de 16 opérations telles que le téléchargement, la création de dossiers, etc. Chaque opération possède une signature spécifique. D'un point de vue non fonctionnel, un certain nombre de qualités sont attendues. Le protocole SSL doit être autorisé, d'où une NFP SSL correspondant à l'attribut qualité $SecuriteSSL$ avec la valeur « True ». Ensuite, le composant doit avoir un degré de test élevé. On entend par « degré de test » le nombre de tests effectués sur le composant parmi les 8 tests reconnus par *ComponentSource* (installation, désinstallation, antivirus...). D'où une NFP DT correspondant à l'attribut $DegreDeTest$ avec la valeur « 8 ». Enfin, le composant doit également avoir un degré de qualité élevé. On entend par « degré de qualité » le nombre de propriétés de qualité que possède le composant sur les 9 attestées par *ComponentSource*. D'où une NFP DQ correspondant à l'attribut $DegreDeQualite$ avec la valeur « 9 ». $DegreDeQualite$ est mesuré par la

Version	Doc	Ops	SSL	DQ	DT	**Total 1**	**Total 2**	**Total 3**
ActiveX (S)	1.0	1.0	1.0	0.89	1.0	**0.988**	**0.985**	**0.991**
ActiveX (NS)	1.0	1.0	0.0	0.89	1.0	**0.877**	**0.919**	**0.824**
Delphi (S)	1.0	1.0	1.0	0.0	0.87	**0.875**	**0.850**	**0.906**
.Net (NS)	1.0	1.0	0.0	0.56	0.87	**0.826**	**0.857**	**0.786**
ASP (NS)	1.0	1.0	0.0	0.33	0.87	**0.801**	**0.828**	**0.767**
Java (S)	1.0	1.0	1.0	0.0	0.0	**0.778**	**0.733**	**0.833**
Delphi (NS)	1.0	1.0	0.0	0.0	0.87	**0.764**	**0.783**	**0.740**

Tableau 1. *Indice de satisfaction selon 3 distributions de poids différentes*

métrique ordinale $NeufDegres$, qui mesure le niveau d'une propriété de 0 à 9. De même, $DegreDeTest$ est mesuré par la métrique ordinale $HuitDegres$, qui mesure le niveau d'une propriété de 0 à 8. Enfin, $SecuriteSSL$ est mesurée par la métrique ordinale $Booleen$ qui indique la présence ou non d'une propriété particulière.

4.2. *Outils utilisés et résultats obtenus*

Comme nous voulions mesurer l'indice de satisfaction uniquement sur les candidats susceptibles de traiter de FTP, nous avons effectué une présélection limitée à la documentation contenant le mot-clé « FTP ». Sur les 131 candidats que contenait la section « communication internet » de *ComponentSource*, seulement 55 composants se sont révélés « pertinents ». Ensuite, les indices de satisfaction pour chaque candidat ont été mesurés sur la version 3.1 de notre outil appelé *Substitute*[2], qui prend en paramètre des fichiers XML décrivant le modèle de qualité choisi et l'ensemble des descriptions des composants utilisés (le composant recherché comme les composants candidats). Cet outil peut retourner les indices de satisfaction locaux pour les éléments fils (opérations, NFP...), ainsi que l'indice global de satisfaction de chaque composant candidat. Pour pondérer les différents éléments du composant recherché, *Substitute* utilise le mode de pondération par distribution.

Nous avons calculé les indices de satisfaction sur les 55 composants candidats restants, en distribuant les poids dans le champ qualité de 3 manières différentes, comme le montre la figure 4. La première distribution choisie est telle que chacune des 3 NFP se voit attribuer 1/3 du poids du champ qualité. La deuxième distribution choisie met en avant les NFP DQ (degré de qualité) et DT (degré de test) par rapport à la NFP SSL. Les deux premières NFP se voient attribuer chacune 2/5 du poids du champ qualité, tandis que la troisième récupère le 1/5 restant. A l'inverse, la troisième distribution choisie privilégie la NFP SSL par rapport aux NFP DT et DQ. La première NFP se voit donc attribuer la moitié du poids du champ qualité, tandis que les deux

2. Le lecteur voulant en savoir plus sur l'outil *Substitute* peut le télécharger à l'adresse suivante : http ://www-valoria.univ-ubs.fr/SE/Substitute/

autres récupèrent chacune 1/4 de ce poids. Pour les trois distributions, la documentation, l'interface fournie et le champ qualité se voient attribuer tous les trois 1/3 du poids total. De plus, chacune des 16 opérations de l'interface fournie se voit attribuer 1/16 du poids de celle-ci.

Le tableau 1 montre les résultats du calcul des indices de satisfaction pour quelques candidats significatifs parmi les 20 meilleurs, c'est-à-dire ceux qui possèdent exactement la documentation (Doc) et les opérations (Ops) demandées dans le composant recherché. Ce sont tous des composants FTP de marque *IP*Works !*. La véritable différence réside dans le degré de qualité DQ ainsi que dans le degré de test DT et dans l'autorisation de SSL. Sur le tableau 1, ces candidats sont désignés par le langage pour lequel ils sont conçus (version Java, version ActiveX, etc.). Les mentions « (S) », pour « version sécurisée », et « (NS) », pour « version non sécurisée », indiquent respectivement si le candidat autorise ou non SSL. Les trois totaux correspondent chacun à un indice de satisfaction total obtenu avec une distribution de poids particulière. Par exemple, « Total 1 » correspond à la première distribution, et ainsi de suite. Les calculs ont montré que quelle que soit la distribution choisie, c'est la version ActiveX sécurisée qui est incontestablement le meilleur candidat, loin devant les autres. En effet, ce candidat possède 8 des 9 propriétés de qualité attestées par *ComponentSource*. La valeur de la NFP correspondante est donc « 8 », le rang de cette valeur est « 8 », et l'indice de satisfaction avec le degré de qualité recherché est de 8/9 = 0.89. En ce qui concerne le degré de test, tous les tests reconnus par *ComponentSource* ont été effectués sur ce candidat.

En revanche, l'influence de la distribution des poids se fait sentir sur le classement des autres candidats. Selon que l'on privilégie DQ et DT par rapport à SSL ou l'inverse, certains candidats remontent dans le classement au détriment des autres. Prenons l'exemple des versions ActiveX non sécurisée et Delphi sécurisée. Avec la première distribution, qui donne des poids équivalents à toutes les NFP, ce sont les deuxième et troisième meilleurs candidats, à seulement deux millièmes d'écart dans leur indice de satisfaction. Avec la deuxième distribution, qui favorise DQ et DT, la version ActiveX non sécurisée est encore deuxième meilleur candidat, et de loin. En effet, elle possède un degré de qualité élevé et un degré de test maximal. A l'inverse, la version Delphi sécurisée possède un degré de test élevé, mais un degré de qualité nul. Elle descend donc dans le classement derrière la version .Net non sécurisée. En revanche, avec la troisième distribution, qui met en avant SSL au détriment de DQ et DT, la version Delphi sécurisée devient deuxième meilleur candidat. On peut observer les mêmes écarts et le même changement dans le classement entre la version .Net non sécurisée et la version Java sécurisée. La manière de distribuer les poids a donc une influence majeure dans le classement des candidats.

Toutefois, ces calculs ont été obtenus en imposant une méthode d'agrégation et une pondération particulières. Une agrégation plus « naturelle » ou intuitive peut être obtenue grâce à l'outil de visualisation de (Langelier *et al.*, 2005). Cet outil permet de représenter chaque candidat par une boîte 3-D, dont la forme est obtenue par agrégation de divers attributs graphiques. Les principaux attributs qui peuvent être affichés

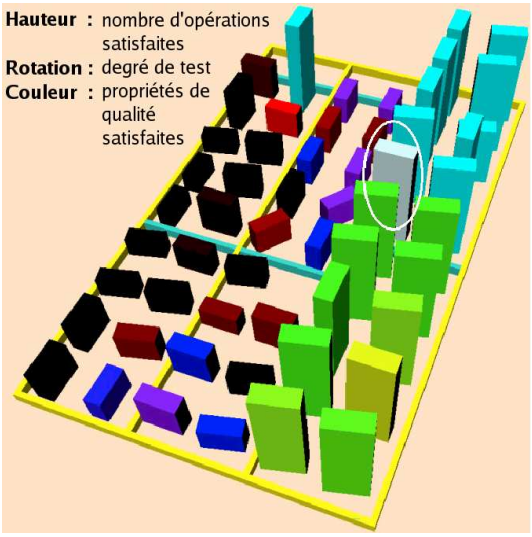

Figure 5. *Visualisation des résultats (le meilleur candidat est entouré en blanc)*

pour chaque boîte sont : la taille, la rotation et la couleur. Chaque attribut correspond à un critère de mesure particulier. Dans notre cas ce critère représente l'indice de satisfaction local pour l'un des éléments du composant. 5 critères de mesure ont été retenus, auxquels on a associé un attribut graphique particulier (non visible sur la version papier) : la documentation est représentée en vert, la NFP SSL en bleu et la NFP DQ en rouge, tandis que la taille représente « les opérations » (combien d'opérations candidates correspondent à une opération recherchée) et la rotation représente la NFP DT. Plus l'indice de satisfaction pour un élément particulier du candidat est élevé, plus l'attribut graphique correspondant influera sur la forme de la boîte représentant ce candidat. Par exemple, plus il possède d'opérations demandées, plus sa taille augmentera. Et comme les couleurs se mélangent, les candidats accumulant les indices locaux élevés ont une boîte dont la couleur se rapproche davantage du blanc[3].

Après avoir calculé les indices de satisfaction avec *Substitute*, l'outil de visualisation a ainsi permis de montrer (figure 5) une vue globale des 55 composants candidats affichés et classés selon les critères de mesure initiaux. Le but était d'identifier selon ces critères les boîtes les plus grandes, dont la couleur était la plus proche du blanc, et dont l'angle était le plus proche de 90 degrés. La boîte qui correspond le plus à cette description est encore une fois la version ActiveX sécurisée du composant FTP d'*IP*Works !* ; ce composant est entouré sur la figure. D'autres boîtes de grande taille, qui possèdent aussi la plupart des opérations demandées, correspondent à de bons candidats, mais soit leur degré de qualité n'est pas bon, soit SSL n'est pas autorisé.

3. Vu le nombre de couleurs utilisées, il est recommandé de lire cet article en ligne.

Cet exemple met en évidence le principal avantage de l'automatisation de la sélection : les indices de satisfaction ont été calculés sur des dizaines de composants en quelques secondes, alors qu'une évaluation manuelle de chaque candidat pour chaque critère aurait pris plusieurs jours. De plus, l'interprétation graphique des résultats à l'aide de l'outil de visualisation offre une autre méthode d'agrégation. Cette méthode permet de repérer intuitivement et en un coup d'oeil les meilleurs candidats. Elle permet également à l'utilisateur d'effectuer ses propres compromis de sélection en ayant accès aux indices locaux de tous les candidats en même temps. Cet exemple met également en évidence l'importance de la pondération dans le résultat final. En effet, une distribution particulière des poids peut rendre certains candidats meilleurs que d'autres, tandis qu'une autre distribution peut provoquer l'effet inverse. Cet étude montre enfin que notre approche de sélection multiniveau permet des comparaisons fines, qui exploitent dans une large mesure les propriétés fonctionnelles et non fonctionnelles des composants.

5. Conclusion et perspectives

Nous avons proposé une approche qui permet l'automatisation de la phase d'évaluation des composants, et qui comprend : i) un format de description des propriétés fonctionnelles et non fonctionnelles des composants sur étagère ; ii) un indice de satisfaction qui évalue le niveau de ressemblance d'un composant candidat avec un composant recherché. La nature hiérarchique du format proposé permet une comparaison à plusieurs niveaux de granularité. Une étude de faisabilité de cette approche a été effectuée sur le marché aux composants *ComponentSource*, au moyen de deux outils. Le premier outil mesure les indices de satisfaction pour toute une bibliothèque, et le second permet d'avoir une vue d'ensemble graphique des candidats. Cette étude nous a permis d'illustrer le gain de temps que confère une sélection automatisée par rapport à des calculs locaux estimés manuellement, ainsi que l'importance de la pondération.

Notre approche se place dans le contexte des marchés aux composants « sur étagère » tels que *ComponentSource*. Notre format de description est donc inspiré de ce qu'on trouve (ou ne trouve pas) dans la documentation offerte sur ces marchés par les fournisseurs de composants. Pour cette raison et dans un premier temps, nous n'avons pas abordé certains aspects, en particulier sémantiques. Toutefois, compte tenu de l'importance de ces aspects, nous pensons prolonger notre approche afin d'en tenir compte, dans l'espoir qu'ils seront un jour documentés sur les marchés aux composants. Nous pensons également proposer d'autres niveaux de description, d'autres sémantiques de comparaison que la satisfaction (par exemple, l'estimation de l'effort) et d'autres mécanismes de pondération. En combinant tous ces éléments, cela permettrait d'obtenir une stratégie progressive de sélection des composants.

Afin de nous concentrer sur la comparaison automatique de composants, nous avons formulé plusieurs hypothèses de travail. En particulier, nous nous sommes placés dans un contexte de construction incrémentale de l'architecture, et nous avons considéré que les descriptions à comparer étaient toutes au même format. Il existe

cependant plusieurs bibliothèques contenant chacune des centaines de composants. Ceux-ci sont décrits et documentés selon des modèles potentiellement différents les uns des autres. Il est donc nécessaire d'extraire ces descriptions, puis de les convertir en un même format, afin que la comparaison avec le composant recherché se fasse sur une base commune. Nous travaillons actuellement à l'automatisation de l'extraction et de la transformation des formats de description pour les composants candidats. Pour ce faire, nous utilisons les techniques de transformation de modèles. D'un point de vue architectural, nous pensons étendre notre mécanisme à la sélection simultanée de plusieurs composants pouvant être intégrés ensemble. Cela permettrait de passer d'une notion de composant recherché à un concept d'architecture recherchée.

6. Bibliographie

Alvaro A., de Almeida E. S., Meira S., « Software Component Quality Model : A Preliminary Evaluation », *Proceedings of the 32nd EUROMICRO SEAA Conference*, August, 2006.

Alves C., Castro J., « CRE : A Systematic Method for COTS Component Selection », *Brazilian Symposium on Software Engineering*, Rio de Janeiro, Brazil, October, 2001.

Bertoa M., Vallecillo A., « Quality Attributes for COTS Components », *I+D Computación*, vol. 1, n° 2, p. 128-144, November, 2002.

Boegh J., « Certifying Software Component Attributes », *IEEE Software*, vol. 40, n° 5, p. 74-81, May, 2006.

Bornwsword L., Obendorf P., Sledge C., « Developing New Processes for COTS-Based Systems », *IEEE Software*, vol. 34, n° 4, p. 48-55, July-August, 2000.

Brada P., Specification-Based Component Substituability and Revision Identification, PhD thesis, Charles University, Prague, Czesch Republic, 2003.

Cardelli L., « A Semantics of Multiple Inheritance », *Information and Computation*, vol. 76, n° 2, p. 138-164, February-March, 1988.

Carvallo J. P., Franch X., Quer C., « Determining Criteria for Selecting Software Components : Lessons Learned », *IEEE Software*, vol. 24, n° 3, p. 84-94, May-June, 2007.

Comella-Dorda S., Dean J., Morris E., Oberndorf T., « A Process for COTS Software Product Evaluation », *Proceedings of 1st ICCBSS*, Orlando, Florida, USA, p. 46-56, 2002.

ComponentSource, « Website », , http ://www.componentsource.com, 2005.

Defour O., Jézéquel J.-M., Plouzeau N., « Extra-Functional contract support in components », *Proceedings of 7th Int. Symp. on Component-Based Soft. Engineering (CBSE 7)*, May, 2004.

En C. G., Baraçli H., « A Brief Literature Review of Enterprise Software Evaluation and Selection Methodologices : A Comparison in the Context of decision-Making Methods », *Proceedings of the 5th International Symposium on Intelligent Manufacturing Systems*, 2006.

Frolund S., Koistinen J., QML : A Language for Quality of Service Specification, Technical report, Hewlett-Packard Laboratories, Palo Alto, California, USA, 1998.

George B., Fleurquin R., Sadou S., « A Substitution Model for Software Components », *Proceedings of ECOOP-QaOOSE Workshop*, Nantes, France, July, 2006.

George B., Fleurquin R., Sadou S., « A Methodological Approach for Selecting Components in Development and Evolution Process », *ENTCS*, vol. 6, n° 2, p. 111-140, January, 2007.

Grau G., Carvallo J., Franch X., Quer C., « DesCOTS : A Software System for Selecting COTS Components », *30th EUROMICRO Conference*, Rennes, France, September, 2004.

IEE, *IEEE Std. 1061-1998 : IEEE Standard for a Software Quality Metrics Methodology*, IEEE computer society press edn. 1998.

ISO, *ISO/IEC 9126-1 Software Engineering - Product Quality - Part I : Quality model*. 2001.

Kontio J., « A Case Study in Applying a Systematic Method for COTS Selection », *Proceedings of International Conference on Software Engineering (ICSE)*, 1996.

Kunda D., Brooks L., « Applying Social-Technical Approach for COTS Selection », *UK Academy for Information Systems Conference (UKAIS 1999)*, University of York, UK, 1999.

Langelier G., Sahraoui H., Poulin P., « Visualization-based Analysis of Quality for Large-scale Software Systems », *Proc. of ACM Symp. on Automated Soft. Eng. (ASE)*, November, 2005.

Lawlis P., Mark K., Thomas D., Courtheyn T., « A Formal Process for Evaluating COTS Software Products », *IEEE Computer*, vol. 34, n° 5, p. 58-63, May, 2001.

Lozano-Tello A., Gómez-Pérez A., « BAREMO : How to Choose the Appropriate Software Component Using the Analytic Hierarchy Process », *SEKE 14 (ACM)*, July, 2002.

Maiden N., Ncube C., « Acquiring COTS Software Selection Requirements », *IEEE Transactions on Software Engineering*, vol. 24, n° 3, p. 46-56, March, 1998.

Martinez M., Toval A., « COTSRE : A COmponenTs Selection Method Based on Requirements Engineering », *Proceedings of the 7th ICCBSS*, Madrid, Spain, p. 220-223, February, 2008.

Mili H., Mili F., Mili A., « Reusing Software : Issues and Research Directions », *IEEE Transactions On Software Engineering*, vol. 21, n° 6, p. 528-562, June, 1995.

Mosley V., « How to Assess Tools Efficiently and Quantitatively », *IEEE Software*, vol. 8, n° 5, p. 29-32, May, 1992.

Ncube C., Dean J., « The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Component », *Proceedings of the 1st ICCBSS*, p. 176-187, 2002.

Ochs M., Pfahl D., Chrobok-Diening G., Nothelfer-Kolb B., « A COTS Acquisition Process : Definition and Application Experience », *Proc. of the 11th ESCOM Conference*, 2000.

OMG, « UML 2.0 Superstructure Final Adopted Specification, Document ptc/03-08-02 », , http ://www.omg.org/docs/ptc/03-08-02.pdf, August, 2003.

Phillips B., Polen S., « Add Decision Analysis to Your COTS Selection Process », *CrossTalk*, vol. 15, n° 24, p. 21-25, April, 2002.

Príeto-Diaz R., « Implementing faceted classification for software reuse », *Communications of the ACM*, vol. 34, n° 5, p. 88-97, May, 1991.

Saaty T., « How to Make a Decision : The Analytic Hierarchy Process », *European Journal of Operational Research*, vol. 48, p. 9-26, 1990.

Sadou S., Koscielny G., Mili H., « Abstracting Services in a Heterogeneous Environment », *Middleware 2001 (IFIP/ACM)*, Heidelberg, Germany, p. 559-563, November, 2001.

Tran V., Liu D.-B., « A Procurement-Centric Model for Engineering Component Based Software Engineering », *Proceedings of the 5th SAST Symposium (IEEE)*, June, 1997.

Voas J., « COTS Software - The Economical Choice ? », *IEEE Software*, vol. 15, n° 3, p. 16-19, March, 1998.

Zaremski A., Wing J., « Signature Matching : a Tool for Using Software Libraries », *ACM Trans. On Soft. Eng. and Methodology (TOSEM)*, vol. 4, n° 2, p. 146-170, April, 1995.