A Component Selection Framework for COTS Libraries

Bart George, Régis Fleurquin, Salah Sadou

A Component Selection Framework for COTS Libraries

Bart George, Régis Fleurquin, and Salah Sadou

VALORIA Laboratory, University of South Brittany, 56017 Vannes, France
{george,fleurqui,sadou}@univ-ubs.fr
http://www-valoria.univ-ubs.fr/s

Abstract. Component-based software engineering proposes building complex applications from COTS (Commercial Off-The-Shelf) organized into component markets. Therefore, 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 non-functional properties. This mechanism has been validated on an example using the *ComponentSource* component market.

1 Introduction

Component-Based Software Engineering allows developers to build a system from reusable pre-existing commercial off-the-shelf (COTS) components. The two immediate potential benefits for such an approach are reduced development costs and shorter time-to-market [1]. For this reason, more and more software applications are built using COTS rather than being developed from scratch, as this is something that fewer and fewer companies can afford [2]. However, due to the intrisic nature of COTS as "black-box" units put into markets by third party publishers, software development life-cycle must be rethought in depth [3,4]. In fact, COTS-based software development leads to constant trade-offs between requirement specification, architecture specification and COTS selection [5]. In this context, it becomes impossible to specify requirements without asking if the marketplace provides COTS that can satisfy them. And one cannot specify an architecture without asking if there are COTS to integrate it.

In such a context, COTS selection becomes particularly important [6]. So important that a bad requirements definition associated to a poor selection of COTS products can lead to major failures [7]. There are also extra costs due to the investigation of hundreds of candidates disseminated into several different markets and libraries, not to mention the diversity of components' description formats. Finally, this phase can become so time-consuming that it may annihilate the initial promise of cost and time reductions [6]. Therefore, the only solution to maintain these gains is to have a selection process [1] that would be well-defined, repeatable, and as automated as possible.

In this paper, we propose a mechanism that allows application designers to select, among a vast library of candidates, the one that best satisfies a specific

M.R.V. Chaudron and C. Szyperski (Eds.): CBSE 2008, LNCS 5282, pp. 286–301, 2008.

need, modeled by a virtual component called a "target component". Section 2 will detail existing approaches, as well as their limits. In section 3 we will present our own approach. Then, before concluding, in section 4 we will present a validation of this approach using *ComponentSource* [8] as a component marketplace.

2 COTS Selection Techniques

The issue is the following one: given a vast number of COTS components from all origins, disseminated in several different markets, how can the one that will best satisfy an application's specific need be chosen ?

2.1 Presentation of Current Selection Processes

Works in the field of component selection are trying to answer this fundamental question. C. Güngör En and H. Baraçli [6] listed many of these works. This study shows that most selection processes provide at least the three following phases: evaluation criteria definition, prioritization of these criteria, and COTS candidates' evaluation according to these criteria. Usually, in order to achieve these phases, selection processes use multi-criteria decision making techniques (MCDM). The most used MCDM techniques are Weighted Scoring Method or WSM [9] and Analytic Hierarchy Process or AHP [10]. WSM consists in using the following formula: $score_c = \Sigma_{j=1}^{n}(weight_j * score_{cj})$, where weight $weight_j$ represents the importance of j-th criterion compared to the n-1 other evaluation criteria, and local score $score_{cj}$ evaluates the satisfaction level of the j-th criterion by candidate c. Thus, total score $score_c$ represents the global evaluation value for candidate c. Therefore, the best candidate is the one that has the highest total score. AHP is a technique that organizes the definition and prioritization of evaluation criteria. It consists in decomposing a goal in a hierarchical tree of criteria and sub-criteria whose leaves are available candidates. Inside each criteria-node, the importance of each sub-criterion is estimated compared to others. For example, the criterion "performance" can be divided in two sub-criteria "response time" and "resource consumption", the first sub-criterion having a weight twice higher than the second one. Then, one can use a formula such as WSM to evaluate each candidate c by aggregating local scores $score_{cj1}, ..., score_{cjn}$ inside each node j, and propagating all these sums to the root of the tree to get c's total score.

Now, let us take a look at the contributions made by main works in the field of component selection. OTSO [11] is considered as one of the first selection processes dedicated to COTS components. In addition to the three phases described above, it adds other ones such as pre-selection of COTS to identify potentially relevant candidates and limit their number (therefore it acknowledges the difficulty to manually evaluate too many candidates). PORE [7] is a selection process that pleads in favor of a progressive selection. Candidates are filtered and their number decreases while the description of the needs becomes more accurate. DEER [12] is aimed at selecting single components, or assemblies

of components, which satisfy requirements while minimizing costs. Other processes propose new steps in order to facilitate selection. For example, PECA [13] adds an extra phase : evaluation planning. It consists in choosing the people responsible for the evaluation of candidates and the techniques they will use. Other approaches focus on the definition of evaluation criteria [14]. For example, STACE [15] proposes taking into account "socio-technical" criteria. Such criteria can be, for instance: product quality, product technology, business aspects (for example, supplier reputation on the marketplace), etc... BAREMO [16] adapts AHP to COTS by defining a set of specific criteria and sub-criteria dedicated to these kind of components. COTSRE [17] proposes to create reusable "criteria catalogs". And CAP [18] proposes specific non-functional criteria inspired by ISO-9126 quality standard [19].

2.2 Limits of These Approaches

The main inconvenience of these processes is their lack of automation. Even if candidates' total scores are calculated with an automated formula such as WSM, local scores $score_{cj}$ are estimated manually by evaluators for each candidate c. Coming back to the example of sub-criteria "response time" and "resource consumption", it is clear that if we want a precise evaluation, it would be much better to measure them automatically with the help of metrics instead of letting a user enter arbitrary local scores. Furthermore, even if we limited ourselves to only one market, or a particular section of a market, we would face more than one hundred candidates anyway. For instance, the single *ComponentSource*'s "internet communication" section [8] contains more than 120 candidates. Therefore, it is important to automate local score measurements as much as possible. It is not only a matter of precision, but also an efficient way to deal with a huge amount of information. A high number of candidates becomes quickly fastidious in the case of a manual evaluation [20].

All local score calculations are not automatable the same way, though. Preselection phase usually uses a small number of general criteria, such as keywords. In this case, local score calculations are simple, but they can apply to a large number of candidates. Component search and retrieval techniques [21], whose goal is to formulate a specific query and then retrieve all the components matching this query, provide adapted algorithms for this kind of local scores, for instance, keyword search or facet-based classification [22]. However, during detailed evaluation phase, there can be many complex criteria, each one concerning a specific property (signature matching, metric value comparison...). In this case, local score calculations are much more complex because they require the aggregation of many values of different nature, but they apply to a smaller number of candidates. Fine-grained comparisons such as signature subtyping [23] fit this kind of comparison. And non-functional properties, in order to be evaluated, can be described with techniques such as quality of service contracts [24,25,26] or COTS-based quality models [27].

The mechanism we propose allows for the automation of these local score calculations by taking into account the need for flexibility on the criteria detail

level. The originality of our approach consists in automating COTS selection by using existing works from other domains. All these techniques cohabitate into a unique concept: target component.

3 Component Selection

Our selection approach takes place in a component-based software development context, as defined in [3]. In this context, a component-based application is built incrementally. When a component is added into the application, it brings its own constraints. Then, its required interfaces become part of the new requirements that must be satisfied by the next component to be integrated. Therefore, the application's current requirements are dictated, among other things, by components currently integrated in it.

We chose to model the application's requirements by virtual "target" components. A target component represents the "ideal" answer for a specific need, and has to be replaced by the closest "concrete" candidate component. Evaluation criteria are components' functional and non-functional properties. Such a representation allows the designer to have criteria that are closer to the application's true needs. It also allows for the use of many techniques dedicated to automatic component search and comparison. However, such a mechanism implies two problems: i) the choice of a description format for candidate components as well as target ones; ii) the definition of a comparison function for such component descriptions to measure their "similarity". In this section, we will successively present the solutions we propose to address these two problems.

3.1 Component Description Format

Nowadays, there is no consensus on component description format. Each market has its own way to document its components, often developed from several different models. For instance, *ComponentSource* stores ActiveX, JavaBeans or .NET components. However, all candidates must be compared to a target component according to a same description format. Furthermore, this format must be abstract enough to encompass concepts that are common to most existing models. This is why we defined our own format dedicated to COTS components. It is described with a UML model (figure 1), whose elements will be presented in the following pages.

Architectural artifacts. Three kinds of artifacts have been selected: components, interfaces, and operations. Components contain two sets of interfaces (provided and required), and interfaces are constituted by a set of operations. This representation is inspired by the standard definition used by many models such as UML 2.0 [28]. As COTS components are represented as "black-boxes", we will not take into account "composite" components.

For each (target) operation, we associate several signatures. It is useful to anticipate many to improve performance during the search for a specific service. A signature $S = ParamTypes \rightarrow ResultType$ details parameters' types,

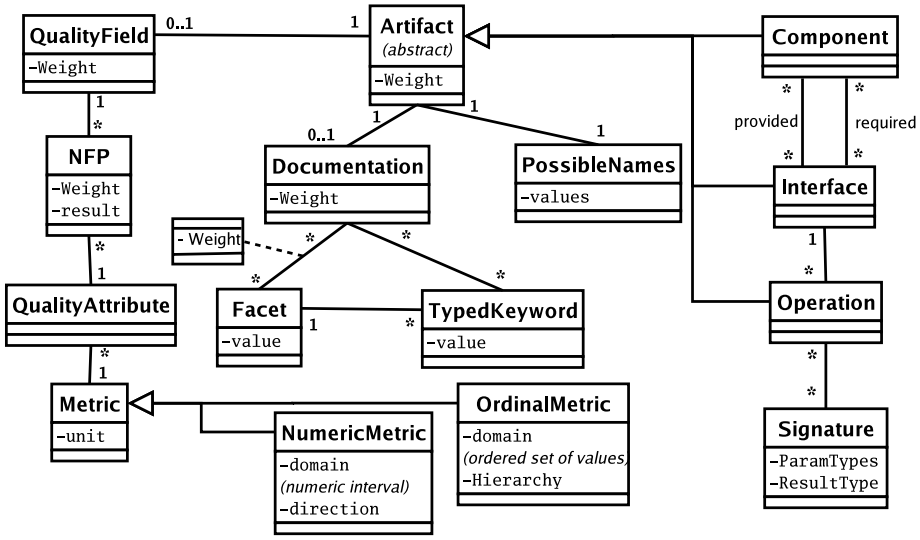

Fig. 1. Description format for COTS components

denoted $ParamTypes = (\tau_1, ..., \tau_n)$, and the result's type denoted *ResultType*. Let us take the example of an operation dedicated to folder creation. It could have a signature $string \rightarrow void$, like the *MakeDirectory* operation provided by *PowerTCP* FTP component, which can be found on the *ComponentSource* website. However, another signature for a folder creation operation could be $string \rightarrow boolean$, like for the *CreateDirectory* operation provided by the *FTPWizard* component, which can be found at the same place.

Information associated to artifacts. Two sets of information are common to all artifacts: a set of its possible names, and a documentation. The first set is here because a same artifact can be proposed under several different names. For example, a download operation can be named *Download* or *GetFile*. This is the case, respectively, for *ComponentSpace*'s FTP component and *Xceed*'s, both being available on *ComponentSource*'s website. The second set of information represents the artefact's documentation. Each one of the information elements included in this documentation is called "typed keywords". A keyword is typed because it positions its value in a specific interpretation domain called facet. For example, a component developed in EJB whose publisher is NBOS Inc. can be documented with two typed keywords: i) one that will have "Publisher" as facet, and "NBOS Inc." as value; ii) one that will have "Technology" as facet, and "EJB" as value.

It is possible to associate other information to artifacts, in particular behavioral information such as pre- and post-conditions. However, the primary goal of our approach is to bring a concrete answer to an industrial concern. To do so, consider the context of COTS component markets such as *ComponentSource*.

Unfortunately, in such markets, the documentation of components' behavior is very poor. This is why we currently do not address these aspects.

3.2 Non-functional Properties Associated to Artifacts

Each artifact can have a "quality field", i.e. a set of non-functional properties (NFP in figure 1). The idea that every architectural artifact can have non-functional properties is inspired by quality of service description languages such as QML [24] and QoSCL [26]. A non-functional property represents the result ($result_P$) obtained by measuring the "level" of a quality attribute on an artifact. This measure is made by a metric. Such a structure is inspired by quality models dedicated to COTS components [27,29]. Such models extend ISO-9126 quality standard [19] by associating quality attributes and metrics to its characteristics and sub-characteristics. We chose metrics to represent and compare non-functional properites, because contrary to other methods focusing on one specific property or family of properties [26], metrics seem to be the simplest evaluation tool for quality in the largest sense.

There are several standards for metrics, such as IEEE 1061-1998 [30], for which a same quality characteristic or sub-characteristic can be measured by several metrics, and conversely. However, there is a problem when a same quality attribute is measured by metrics of a different kind from one quality model to another. Let us take the example of two different quality models: Bertoa's and Vallecillo's model [29] and CQM [27]. Sometimes, both models associate the same quality attributes to ISO-9126's sub-characteristics, but measure them with different metrics. For instance, the *Controllability* attribute, associated to sub-characteristic *Security*, is measured by a percent value in Bertoa's and Vallecillo's model, whereas it is measured by a boolean in CQM. But even though metrics measuring the same attribute may have the same type, it does not mean they are semantically comparable. And there are no systematic methods allowing one to compare values obtained for a same quality attribute with different kinds of metrics. Consequently, we will consider for our description format that one quality attribute can be measured by only one metric, even though a same metric can measure several quality attributes. We can use attributes and metrics from one existing quality model. It can either be an academic one, or one provided by a component market such as *ComponentSource*. In any case, it must be the same for all components.

A metric can be numeric or ordinal. This distinction is inspired by the CLARIFI project [31]. The domain of a numeric metric is a subset of real numbers (integers, percent values...). As the quality models we surveyed do not propose metrics with negative values, we take as a hypothesis that the domain of a numeric metric is always positive. About the domain of an ordinal metric, it is a finite and totally ordered set. A numeric metric has a supplementary attribute called "direction". This direction allows for the interpretation of a metric's result. Available directions are *increasing* and *decreasing*. This distinction is inspired by quality of service contract languages [24,26]. An *increasing* (resp. *decreasing*) direction means that the higher (resp. the lower) the metric's value, the better the corresponding quality. For example, an operation's execution time has a

decreasing direction. An ordinal metric has one supplement attribute called "hierarchy". It gathers and ranks all the metric's possible values by associating a key to each one of them. This key, or rank, defines the total order relation on the metric's domain. When a value is "better" than another, the first one's rank is strictly superior than the second one's rank in the associated hierarchy. For example, if an ordinal metric M has {*very bad*, *bad*, *average*, *good*, *excellent*} as a domain, corresponding hierarchy is: $Hierarchy(M)$=[(0, *very bad*), (1, *bad*), (2, *average*), (3, *good*), (4, *excellent*)].

3.3 Satisfaction Index between Components

Using the same description format given above for candidate and target components, we can address the problem of component comparison. In selection processes and multi-criteria decision making techniques, after total score calculations are performed, the candidate with the highest total score is selected as the best one. This is why we chose to define a satisfaction index based on the same principle. This index allows one to determine how much a candidate component fits the target one. That means, how many functional and non-functional properties this candidate has in common with the target component. First, we will present the principle and the general formula for this satisfaction index, before giving the details for some elements of the description format.

General definition. A careful analysis of description format allows us to distinguish a hierarchical description. In this format, a component is described by a tree whose root is a component artifact and child nodes are potentially: $Interface$, $Documentation$, $PossibleNames$ and $QualityField$. Among them, an $Interface$ node can have the following child nodes: $Operation$, $Documentation$, $PossibleNames$ and $QualityField$. Therefore, the satisfaction index must compare recursively two nodes from different trees by comparing their respective child nodes pair by pair, then "aggregate" the result of sub-nodes' comparisons to measure similarity score. This calculus is the same whatever the nature of the compared nodes is (as long as they are both of the same nature). Therefore, we will give a generic description of this calculus independently of their nature.

To each node, we associate a type and a weighting function. For example, a node can have $Interface$, $Operation$ or $Documentation$ as a type. Only two nodes of a same type can be compared, otherwise the satisfaction index between them will return 0. On the opposite, the maximum value for a satisfaction index is fixed to 1, which means the candidate element completely fits the target one. Weighting function $Weight(E)$ allows the designer to associate to each node E a numeric value called "weight", which gives its importance compared to other nodes. General satisfaction index calculus for a target node E0 and a comparable candidate node E1 is described in figure 2. For each node e0, child of E0, we measure satisfaction indices with each child node of E1 that is comparable to e0 (respectively, index1, index2 and index3). The best result is the highest satisfaction index among them. This measurement is repeated for all E0's other child nodes. Finally, all these best indices, with their corresponding weight, are added

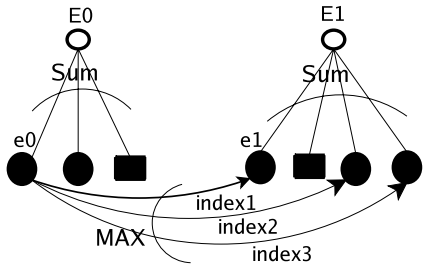

Fig. 2. Satisfaction index between two elements

to obtain a total satisfaction index between E1 and E0. In order to compare leaf nodes, we use a specific function to each kind of them.

Formally, the satisfaction index between a candidate element E_1 and a target one E_0, denoted $Index$, is defined as follows:

$$Index(E_1, E_0) = \begin{cases} \bullet\ 0 \text{ if } Type(E_1) \neq Type(E_0). \\ \bullet\ Comp(E_1, E_0) \text{ if } E_0 \text{ is a leaf node.} \\ \bullet\ \Sigma(\{Weight(e_0) * MAX(\{Index(e_1, e_0) \mid e_1 \in E_1\}) \mid e_0 \in E_0\}) \\ \text{ if } E_0 \text{is an inner node.} \end{cases} \tag{1}$$

Selection is performed by calculating satisfaction indices between each available candidate component and the target one, then choosing the candidate whose satisfaction index is the highest one.

Chosen weighting and comparison functions. Now that the general satisfaction index formula has been defined, we have to detail comparison functions we have chosen for each type of leaf node (NFPs, sets of possible names, typed keywords and operation signatures), as well as the weighting function we have chosen for every type of node.

Comparison function between NFPs: Let A_0 be a target artifact, P_0 be an NFP belonging to A_0's quality field, A_1 be a candidate artifact having the same type as A_0, and P_1 be an NFP belonging to A_1's quality field. P_1 is comparable to P_0 only if they measure the same quality attribute. In this case, the metric they both use will be denoted M. If M is numeric, comparison function will measure the similarity of P_1's result value with P_0's, with respect to M's direction. If M is ordinal, the comparison function will measure the similarity of P_1's result's rank with P_0's result's rank, with respect to M's hierarchy.

Formally, the comparison function between P_1 and P_0 is defined as follows:

$$Comp(P_1, P_0) = \begin{cases} \bullet\ 0 \text{ if } P_1 \text{ and } P_0 \text{ do not measure the same quality attribute.} \\ \bullet\ Comp_{inc}(P_1, P_0) \text{ if } M \text{ is numeric with an } \textit{increasing} \text{ direction.} \\ \bullet\ Comp_{dec}(P_1, P_0) \text{ if } M \text{ is numeric with a } \textit{decreasing} \text{ direction.} \\ \bullet\ Comp_{ord}(P_1, P_0) \text{ if } M \text{ is ordinal, and if } rank(result_{P_0}) > 0. \\ \bullet\ 1 \text{ if } M \text{ is ordinal, and if } rank(result_{P_0}) = 0. \end{cases} \tag{2}$$

With:

$$Comp_{inc}(P_1, P_0) = MIN(\frac{result_{P_1}}{result_{P_0}}, 1) \quad (3)$$

$$Comp_{dec}(P_1, P_0) = MIN(\frac{result_{P_0}}{result_{P_1}}, 1) \quad (4)$$

$$Comp_{ord}(P_1, P_0) = MIN(\frac{rank(result_{P_1})}{rank(result_{P_0})}, 1) \quad (5)$$

Let us suppose that P_0 and P_1 both measure the "quality level" of a particular attribute with an ordinal metric whose domain is {*very bad*, *bad*, *average*, *good*, *excellent*}. If P_0's result value is *good*, its rank is 3. If P_1's result value equals *bad*, its rank equals 1 and the comparison function between P_1 and P_0 gives 1/3.

Comparison function between sets of possible names: For simplicity reasons, we consider that for one candidate set of possible names N_1 and one target set possible names N_0, $Comp(N_1, N_0)$ equals 1 if one of N_1's names is contained in N_0 (i.e. there is at least one common element between N_1's values and N_0's), regardless of differences between uppercases and lowercases. In any other case, $Comp(N_1, N_0)$=0.

Comparison function between typed keywords: For a candidate typed keyword K_1 associated to a facet F_1 and a target typed keyword K_0 associated to a facet F_0, $Comp(K_1, K_0)$=1 if F_1's value equals F_0's (unless F_0's value is "", in this case facets are not compared) and if K_1's value equals K_0's. In any other case, $Comp(K_1, K_0)$=0.

Comparison function between operation signatures: To automate comparison between operation signatures, we chose to use signature subtyping, in particular contravariance and covariance rules [23]. Therefore, we consider that a candidate signature S_1=$ParamTypes_1 \rightarrow ResultType_1$ is subtype of a target signature S_0=$ParamTypes_0 \rightarrow ResultType_0$ if $ParamTypes_0$ is subtype of $ParamTypes_1$ and if $ResultType_1$ is subtype of $ResultType_0$. Consequently, $Comp(S_1, S_0)$=1 if and only if S_1 is subtype of S_0. Otherwise, $Comp(S_1, S_0)$=0.

Let us consider, for example, target signature S_0: $float \rightarrow int$. If $S_1 = float \rightarrow float$, it will be a subtype of S_0 and the comparison function will give 1. However, if $S1$: $boolean \rightarrow int$, the comparison function will give 0.

There are other existing techniques to compare operation signatures, in particular signature matching as defined by A. Zaremski and J. Wing [32] or S. Sadou *et al.* [33,34]. However, all matching rules are not fully automatable, because they require a collaborative approach.

Weighting function: For every type of node, we will use "weighting by distribution". It consists, for the application designer, in giving a percent weight and sharing the totality of each node's weight between its direct child nodes, from the root (the component whose weight is 1) to the leaves. For example, an interface will share its weight between its set of possible names, its documentation, its operations and its quality field, so that the sum of all these nodes' weights will equal 100% of the interface's weight. The only exception is the set of possible

signatures for an operation. As we look for only one correct signature among all the ones we propose, all of them will count for one. Let us suppose we described a target operation with three possible signatures and a quality field. If the quality field takes 40% of the interface's weight, then each signature will have a weight equal to 60% of the interface's weight.

4 Selection in *ComponentSource*

In this section, we present an experiment conducted on a concrete component market. This experiment shows practical feasability and interest of an automatic, multi-level, selection approach. We consider the following context: a designer needs for her/his application a component dedicated to FTP (*File Transfer Protocol*) among all the candidates available in the *ComponentSource* component market's "internet communication" section[1]. For each candidate we produced a description in our format. In particular, the quality model we used corresponds to the non-functional information available on *ComponentSource* web pages for each of its components. Then, we tested the selection mechanism on these translated descriptions, and produced the results presented in this section. Translation from original components' descriptions to our description format has been done using model transformation techniques. Because of article size, this work will not be presented in this paper, but it will be the subject of a future publication.

4.1 Non-negotiable Requirements

Let us consider the development of a component-based application. Some constraints are specific to the development context. They are often imposed and non-negotiable. For example, if the chosen development tool is *Visual Studio*, then the candidate components must have *Visual Studio* as a compatible container[2]. Therefore, we must filter candidates to keep only the ones that fit these technological constraints. We can model this filter with our framework. To do so, we can use this constraint on compatible containers as a property whose values are either "True" or "False". Then, we can specify a target component with an NFP corresponding to the compatibility with the *Visual Studio* container. Therefore, with only one constraint whose value is "True" or "False", the possible satisfaction index values are 1 or 0. Only the candidates whose satisfaction index equals 1 will be pre-selected. Thus, only 35 "compatible" candidates from *ComponentSource* remain. In the following pages, measurements will be performed only on these compatible candidates.

4.2 Initial Requirements

The application being in development, its current "concrete" architecture has requirements. That means, the designer needs to find a FTP component that

[1] For more information: http://www.componentsource.com/index.html

[2] There may be other non-negotiable requirements and filters, but for reasons of simplicity and clarity, we will use only this one as an example.

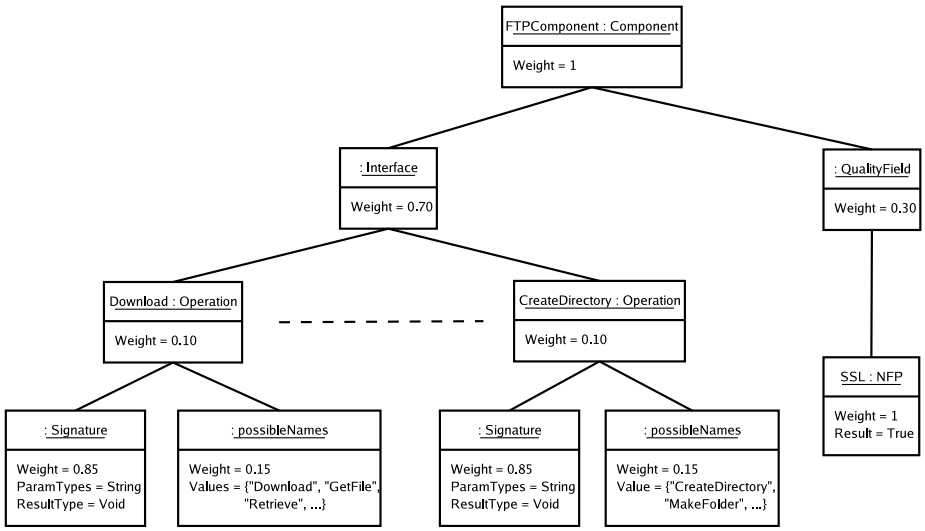

Fig. 3. Example of target component

provides operations whose signatures are required by "concrete" components already included in the architecture. Moreover, the application has security requirements, which imposes that the FTP component enables SSL protocol. It corresponds to quality attribute $SecuritySSL$, measured by an ordinal metric with a boolean domain (see figure 1). The target component that models all these requirements is shown in figure 3. From a functional point of view, it provides one interface containing 10 operations. Each one of them is dedicated to a specific FTP task (folder creation, login, download...), and has a signature imposed by the application's concrete architecture. However, it can have many possible names. From a non-functional point of view, the target component's quality field contains a unique NFP concerning the need of SSL protocol. This NFP represents quality attribute $SecuritySSL$ with value $True$. We estimate that the provided interface takes 70% of the component's weight, while the quality field takes the remaining 30%. Therefore, provided interface and quality field weights equal respectively 0.7 and 0.3 (see figure 3). We also consider that all 10 operations are of equal importance, so each one's weight equals 0.1 in the context of the provided interface. Of course, this is only an example of possible weighting: other weights can be estimated according to the context of the application.

4.3 Results for Initial Requirements

Satisfaction indices have been measured for each of the 35 candidates on our tool *Substitute*[3]. This tool takes as parameters XML files describing the chosen

[3] Because of article size, we cannot give details of *Substitute* tool. However, interested readers can download it at the following address: http://www-valoria.univ-ubs.fr/SE/Substitute/

Table 1. First satisfaction index measurements

Candidate name	Operations	NFP SSL	**Total**
IP*Works! SSL v6 .NET	0.73	1.0	**0.81**
IP*Works! SSL v6 ActiveX/VB	0.73	1.0	**0.81**
IP*Works! SSL v6 .NET Compact Framework	0.73	1.0	**0.81**
IP*Works! SSL v6 ASP/.NET	0.73	1.0	**0.81**
PowerTCP SSL for ActiveX	0.65	1.0	**0.75**
IP*Works! v6 .NET	0.73	0.0	**0.51**
IP*Works! v6 ActiveX/VB	0.73	0.0	**0.51**
IP*Works! v6 .NET Compact Framework	0.73	0.0	**0.51**
IP*Works! v6 ASP/.NET	0.73	0.0	**0.51**
PowerTCP FTP for ActiveX	0.65	0.0	**0.45**
Aspose Network .NET	0.65	0.0	**0.45**
IP*Works! SSL v6 C++	0.14	1.0	**0.39**
SocketTools Secure Visual Edition	0.12	1.0	**0.38**
SocketTools Secure .NET Edition	0.12	1.0	**0.38**
Xceed FTP Library	0.52	0.0	**0.36**

quality model and the set of all the component descriptions that will be used (the target component, and candidate ones). For target components, only needed properties are specified and weighted. Properties that are not specified take implicitly a null weight in our calculus. Once all XML component descriptions are loaded, *Substitute* returns satisfaction indices between each candidate of the library and the target component. Not only global indices, but also local ones for child nodes (interfaces, operations, NFPs...).

Table 1 shows the measurements for the 15 best candidates. First, there are 5 secure (SSL) and 4 non-secure (without SSL) versions of a FTP component provided by the *IP*Works!* component suite. These different versions are identified according to their language (C++), their framework (.NET, ActiveX) or the context they were developed for. For example, the .NET Compact Framework is made specifically for mobile phones, while the ASP/.NET version is better suited for Web applications. Then, there are the secure and non-secure ActiveX versions of a FTP component provided by *PowerTCP* component suite. There are also other FTP components provided by component suites *Aspose Network*, *SocketTools* and *Xceed*. At first, we ignored the candidates' development context. However, the first four candidates all have the same satisfaction index. The reason is that they all represent the same secure FTP component, but for different development contexts (ActiveX, .NEt...). Thus, they provide the same operations with the same signature. Then, we must consider a precise context, such as the development of a client/server application based on ActiveX. In order to choose between the remaining candidates, a more in-depth analysis of them is necessary.

Table 2. New satisfaction index measurements

Candidate name	Operations	NFP SSL	NFP TD	**Total**
IP*Works! SSL v6 ActiveX/VB	0.73	1.0	1.0	**0.81**
PowerTCP SSL for ActiveX	0.65	1.0	1.0	**0.75**
IP*Works! v6 ActiveX/VB	0.73	0.0	1.0	**0.61**
PowerTCP for ActiveX	0.65	0.0	1.0	**0.55**
Xceed FTP Library	0.52	0.0	0.88	**0.45**

4.4 Requirement Evolution

By focusing on a precise development context, thus removing the versions of a same component made for a different context, further exploration becomes conceivable on a remaining candidate. Thus, we can notice they have some properties which were not considered first, but may be very interesting. A good example of such unexpected properties is the tests performed on these components before they were brought to the market. As the application has security requirements, it would be better if the candidates were tested before being integrated. Indeed, *ComponentSource* provides for each component some information about the tests performed on it: installation test, uninstall test, antivirus scan, sample code review, etc... Therefore, it would be interesting to check the "test degree" of each candidate, i.e. the number of tests, among the eight ones recognized by *ComponentSource*, which were performed on it.

This new requirement leads to a modification of the target component. Its quality field now contains a new NFP representing test degree and asking for the maximal value, 8. Provided interface and quality field weight do not change. However, inside the quality field, the weight of NFP representing SSL enabling must decrease a bit. It will equal 0.665 (two thirds of the quality field's weight), while the new NFP representing test degree will take the remaining 0.335.

4.5 Results for New Requirements

Satisfaction indices for the remaining candidates (i.e. those that are developed for ActiveX) have been calculated with *Substitute*. Table 2 shows the new measurements of satisfaction indices for the five best candidates. The new column, *NFP TD*, shows the results for NFP representing test degree. All the tests recognized by *ComponentSource* were performed on the *IP*Works!* and *PowerTCP* components in their secure and non-secure versions (index for NFP TD equals 1). The last candidate did not pass some of these tests, which decrease its satisfaction index. Finally, it seems obvious that, for the specified requirements, secure ActiveX/VB version of *IP*Works!* FTP component is the best candidate.

Usually, when we try to select the "best" candidate for an application's current requirements, we are limited by our initial knowledge. With a way to navigate through the library, it is possible to discover properties offered by the components which we did not originally think about. This is typically one of the trade-offs

predicted by L. Brownsword *et al.* [5] between requirement specification and COTS selection. Our easy-to-use way to specify requirements and select good candidates makes this navigation possible.

5 Conclusion and Future Work

We proposed an approach that allows us to automate the component evaluation phase, including: i) a description format for COTS components' functional and non-functional properties; ii) a satisfaction index that measures the similarity level between a candidate component and a target one. This approach has been validated on *ComponentSource* component market with the help of a tool that measures local and global satisfaction indices for a whole library of candidate components. This study showed that an automated comparison improves the performance of selection process. It also showed the importance of weighting.

Such an automated mechanism is adapted to an incremental construction of a component-based software, because "back-tracking" is possible. Each target component's specification depends on the components already integrated into the application. So if the situation is blocking (i.e. there is no candidate that can satisfy current target component), we can go back to previous ones and choose other candidates for them. It will lead to modified requirements, which may be better satisfied by candidates.

This paper follows and improves a previous work [35,36], whose goal was to find how a component could substitute another one. At that time, we considered no particular context. Since then, we adapted and improved our framework by considering an industrial problem, such as selection in COTS markets. Therefore, the work we present in this paper is better suited to a concrete component-based development context. Considering this context, our description format is inspired by what we do (and do not) find in documentation provided by COTS publishers. For this reason, we have not yet dealt with some component properties, particularly behavioral ones. Because of the importance of these aspects, we plan to take them into account in future versions of our framework. However, in order to achieve this goal, these properties should be documented more explicitly in component markets.

References

1. Voas, J.: COTS software - the economical choice? IEEE Software 15 (3), 16–19 (1998)
2. Ye, F., Kelly, T.: COTS product selection for safety-critical systems. In: Proc. of 3rd Int. Conf. on COTS-Based Soft. Systems (ICCBSS), pp. 53–62 (2004)
3. Crnkovic, I., Larsson, S., Chaudron, M.: Component-based development process and component lifecycle. In: 27th International Conference on Information Technology Interfaces (ITI), Cavtat, Croatia. IEEE, Los Alamitos (2005)
4. Tran, V., Liu, D.B.: A procurement-centric model for engineering CBSE. In: Proc. of the 5th IEEE Int. Symp. on Assessment of Soft. Tools (SAST) (June 1997)

5. Brownsword, L., Obendorf, P., Sledge, C.: Developing new processes for COTS-based systems. IEEE Software 34 (4), 48–55 (2000)
6. 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. In: Proc. of the 5th Int. Symp. on Intelligent Manufacturing Systems (May 2006)
7. Maiden, N., Ncube, C.: Acquiring cots software selection requirements. IEEE Transactions on Software Engineering 24 (3), 46–56 (1998)
8. ComponentSource: Website (2005), `http://www.componentsource.com`
9. Mosley, V.: How to assess tools efficiently and quantitatively. IEEE Software 8 (5), 29–32 (1992)
10. Saaty, T.: How to make a decision: The analytic hierarchy process. European Journal of Operational Research 48, 9–26 (1990)
11. Kontio, J.: A case study in applying a systematic method for COTS selection. In: Proceedings of International Conference on Software Engineering (ICSE) (1996)
12. Cortellessa, V., Crnkovic, I., Marinelli, F., Potena, P.: Driving the selection of COTS components on the basis of system requirements. In: Proceedings of ACM Symposium on Automated Software Engineering (ASE) (November 2007)
13. Comella-Dorda, S., Dean, J., Morris, E., Oberndorf, T.: A process for COTS software product evaluation. In: Proc. of 1st Int. Conf. on COTS-Based Soft. Systems (ICCBSS), Orlando, Florida, USA, pp. 46–56 (2002)
14. Carvallo, J.P., Franch, X., Quer, C.: Determining criteria for selecting software components: Lessons learned. IEEE Software 24 (3), 84–94 (2007)
15. Kunda, D., Brooks, L.: Applying social-technical approach for COTS selection. In: UK Academy for Information Systems Conf. (UKAIS 1999) (April 1999)
16. Lozano-Tello, A., Gómez-Pérez, A.: Baremo: How to choose the appropriate software component using the analytic hierarchy process. In: Proc. of Int. Conf. on Soft. Eng. and Knowledge Eng (SEKE), Ischia, Italy (July 2002)
17. Martinez, M., Toval, A.: COTSRE: A components selection method based on requirements engineering. In: Proceedings of the 7th Int. Conf. on COTS-Based Soft. Systems (ICCBSS), February 2008, pp. 220–223 (2008)
18. Ochs, M., Pfahl, D., Chrobok-Diening, G., Nothelfer-Kolb, B.: A COTS acquisition process: Definition and application experience. In: Proceedings of the 11th European Software Control and Metrics Conference (ESCOM), pp. 335–343 (2000)
19. ISO International Standards Organisation Geneva, Switzerland: ISO/IEC 9126-1:2001 Software Engineering - Product Quality - Part I: Quality model (2001)
20. Ncube, C., Dean, J.: The limitations of current decision-making techniques in the procurement of COTS software component. In: Proc. of the 1st Int. Conf. on COTS-Based Software Systems (ICCBSS), Orlando, Florida, USA, pp. 176–187 (2002)
21. Mili, H., Mili, F., Mili, A.: Reusing software: Issues and research directions. IEEE Transactions On Software Engineering 21(6), 528–562 (1995)
22. Príeto-Diaz, R.: Implementing faceted classification for software reuse. Communications of the ACM 34(5), 88–97 (1991)
23. Cardelli, L.: A semantics of multiple inheritance. Information and Computation 76(2), 138–164 (1988)
24. Frolund, S., Koistinen, J.: QML: A language for quality of service specification. Technical report, Hewlett-Packard Laboratories, Palo Alto, California, USA (1998)
25. Beugnard, A., Sadou, S., Jul, E., Fiege, L., Filman, R.: Concrete communication abstractions for distributed systems. In: Object-Oriented Technology, ECOOP 2003 Workshop Reader, Darmstadt, Germany, November 2003, pp. 17–29 (2003)
26. Defour, O., Jézéquel, J.M., Plouzeau, N.: Extra-functional contract support in components. In: Proc. of 7th Int. Symp. on CBSE (May 2004)

27. Alvaro, A., de Almeida, E.S., Meira, S.: A software component quality model: A preliminary evaluation. In: Proc. of the 32nd EUROMICRO Conf. on Soft. Eng. and Advanced Applications (SEAA) (August 2006)
28. OMG: UML 2.0 superstructure final adopted specification, document ptc/03-08-02 (August 2003), http://www.omg.org/docs/ptc/03-08-02.pdf
29. Bertoa, M., Vallecillo, A.: Quality attributes for COTS components. I+D Computación 1(2), 128–144 (2002)
30. IEEE: IEEE Std. 1061-1998: IEEE Standard for a Software Quality Metrics Methodology. IEEE computer society press edn (1998)
31. Boegh, J.: Certifying software component attributes. IEEE Software 40(5), 74–81 (2006)
32. Zaremski, A., Wing, J.: Signature matching: a tool for using software libraries. ACM Trans. On Soft. Eng. and Methodology (TOSEM) 4(2), 146–170 (1995)
33. Sadou, S., Mili, H.: Unanticipated evolution for distributed applications. In: 1st Int. Workshop on Unanticipated Software Evolution (USE) (June 2002)
34. Sadou, S., Koscielny, G., Mili, H.: Abstracting services in a heterogeneous environment. In: IFIP/ACM International Conference on Distributed Systems Platforms, Middleware 2001, Heidelberg, Allemagne (November 2001)
35. George, B., Fleurquin, R., Sadou, S.: A component-oriented substitution model. In: Proceedings of 9th Int. Conf. on Software Reuse (ICSR 9) (June 2006)
36. George, B., Fleurquin, R., Sadou, S.: A methodological approach for selecting components in development and evolution process. Electronic Notes on Theoretical Computer Science (ENTCS) 6(2), 111–140 (2007)