SEC: A Search Engine for Component based software development

Sofien KHEMAKHEM
ISF of Sfax
DEP. of computer science
P.B. 88A-3099 Sfax, Tunisia
Khemakhem_sofien@yahoo.fr

Khalil DRIRA
LAAS-CNRS
7, Avenue du Colonel Roche
31077 Toulouse Cedex 4
Khalil@laas.fr

Mohamed JMAIEL
University of Sfax
Laboratory ReDCAD
P.B. 3038 Sfax, Tunisia
Mohamed.jmaiel@enis.rnu.tn

ABSTRACT
The successfulness of the component-based development(CBD) process relies on several factors including the structuration of the component repositories and the comparison procedures for interface exploring while comparing the expected and the provided services. Both functional and non-functional aspects should be considered. This paper presents a discovery ontology to organize components in a repository and an integration ontology to integrate a component into an application. In addition, we propose a search engine, called SEC for CBD, which uses the discovery ontology to automatically locates and presents a list of software components that could be used in the current development situation. This search engine consists of a persistent and an intelligent component which automatically generates a query from developer specification and indexes a repository of software components. SEC is not only suitable to discover components, but also able to automatically classify the selected components using the subsumption notion.

Categories and Subject Descriptors
D.2.13 [Software Engineering]: Reusable Software — Reusable libraries

Keywords
Software component, discover, ontology, non-functional aspect

1. INTRODUCTION
Component-based and service-oriented software architectures are likely to become widely used technologies in the future distributed system development. Component reuse is a critical requirement for the development process of component based and service-oriented software.

Component are developed as important and big autonomous and customizable software units. The successfulness of the reuse is important and depend on the efficiency of the search procedure. The search step will become an important step in the development process. The search step may fail if the explored component repositories are not appropriately structured. This step may also fail if the requirement service and the provided service are not appropriately compared.

During the development process, the developer is faced with a significant number of various component types. The use of a software components repository, having a clear structure, is crucial for the effectiveness of the CBD approach. This allows the developer to easily search and select the component which meets his/her needs perfectly.

Several approaches tried to improve software components classification to simplify the request specification and to obtain better results. In existing work, we distinguish five software component classification approaches.

The Adhoc technique, called also behavioral technique is based on the exploitation of results provided by the execution of the component. These results are collections of answers which describe the dynamic behavior of the component [24], [23], [3]. A relation of a behavioral nature must be used to classify the software components.

The second approach is based on the semantic characteristics of the component presented by a pair (attribute, value) [21], [25]. It represents the functional aspects of the software components. The identification of these characteristics and the components check classification procedure are fixed and verified by an expert of the domain. The similarity between two components is measured by the number of common characteristics. Research is based on a syntactic comparison of the set of the characteristics.

The third approach uses the facet classification. The facet classification approaches [5], [20], [26] represent the type of information to describe the software components. Each facet has a name which identifies it and a collection of a well-controlled terms known as vocabulary to describe its aspects. For example, the facet component-type can have the following values: COM, ActiveX, javaBean, etc. In search procedure, the user query is specified by the selection of a term for each facet.

The fourth technique of classification is based on the lattice. The concept of lattice was initially defined by R. Wille [27]. This concept is the representation of a relation, R, between a collection of objects G (Gegendale) and a collection of attributes M (Merkmale). The triplet (G, M, R) is called concept. Wille considers each element of lattice as a formal concept and the graph (Hasse diagram) as a re-
2. ONTOLOGY SUPPORT FOR COMPONENT DESCRIPTION

We have described the semantics of components means to express knowledge about the behavior, functional aspect and non-functional aspect of a component or component process. This knowledge comprises:

- Structural features: they specify the component’s internal structure. The developer uses these features to determine if interaction exists between component operations and the other components used to build the current project.

- Functional aspect: it identifies the functionalities of the component is expected to provide through many features. These features include methods that are used to adapt the behavior of the component to his context. The adaptation is made by specializing and customizing. The other kind of features are used by the application specific part of a component based software.

Generally this type of information is specified by the component’s methods.

- Non-functional aspect: it specifies the component constraints related to communication or computation. The non-functional aspect includes features such as performance, availability, reliability, security, adaptability and dependability. We distinguish the static and the dynamic categories of non-functional features. Static features, such as security-related constraints, do not change during component execution. Dynamic features, such as performance-related properties, depend on the deployment environment.

All of this features represent different and complementary views of a component. The feature set used to describe a component, depends on the developer action: discovery and integration. The discovery of a component is made by sending a query to the repository manager. Once a set of components have been selected, additional features are specified to select a component before integration. For the discovery action, the query must include functional and/or non-functional features. For integration action the structural features have to be specified.

Our approach is based on the following ontologies:

- The discovery ontology: it specifies the functional and non-functional features.

- The integration ontology: it describes the PSMs (Problem-Solving Methods) used to specify a component’s structural features.

The integration ontology’s purpose is to decompose component’s functional features from its internal specification. Many approaches have proposed process-based languages such as BPEL (Business Process Execution Languages) [2] and OWL [7]. These languages describe a component’s internal structure using a predefined set of workflow-like patterns (sequence, parallel split, choice, etc.). But these languages lack an explicit, declarative decoupling between a component’s functional features (what) and its structural description (how) [10].

In integration ontology we try to divide the process or component into tasks. Tasks are either solved directly (by means of primitive methods), or are decomposed into sub-tasks (by means of decomposition methods) whose interaction can be modeled as workflow pattern [1]. As result we obtain a set of PSM’s which are mapped to knowledge elements. We use The Unified Problem-Solving Method Language (UPML) [8] to describe the components of PSMs.
For our present approach, we suppose that such PSMs are to be provided by component suppliers. For our future work, we will address the problem of deducing such description from component source code.

The discovery ontology describes the subject matter using the notions of concepts, instances, relations, and axioms [12].

- Concepts are organized in taxonomies through which inheritance mechanisms can be applied. A concept contains slots which are restricted by facets. In our case we specify a component as a concept that contain many slots such as the performance slot that describes the component performance by one of the three values (low, medium, high).

- Relations represent a type of interaction between concepts. They are formally defined as any subset of a product of n sets: R: C1 x C2 x ... x Cn. Examples of binary relations include: subclass-of and connected-to. For example the relation between component and method is a connected-to relation. The concept method contains the input types, output type, precondition and signature slots.

- Functions are a special case of relations in which the n-th element of the relationship is unique for the n-1 preceding elements. Formally, functions are defined as: F: C1 x C2 x ... x Cn-1 x Cn. Examples of functions are father-of and rank-of-a-component that calculates the rank of a component depending on the “used rate” and kindness match. The “used rate” computes the rate of the component utilization [14]. The kindness(goodness) of match is related to the subsumption notion [17]. The subsumption idea has to be related to suitable matching notions for components provided and query specification in order to refine the selection result.

- Instances are used to represent elements.

The discovery ontology is developed with PROTEGE2000 [13] and mapped through the Resource Description Framework which is an XML-based language [16] and which also represents the component descriptions in the repository. We have chosen XML because of its expressive power which allows representing complex information structures easily. We use the emerging XML Schema standard as a metalanguage for specifying the syntax and structure of component descriptions. The XML schema consists of a set of core schema type definitions. Our XML document, generally, contains the following key concepts: Component, Interface, Dynamic_NF_Aspect that represents the component dynamic aspect, and Static_NF_Aspect that represents the component static aspect. Each concept contains many slots which describe component features. The slot, generally, contains the following key elements: mincardinality, maxcardinality, label, domain and range. as indicated in the figure 2 the slot input has as mincardinality value “1” and as label the value “input”. The domain represents the concept of the slot, in our case the concept of input is “interface”. The range indicates the value type of the slot, in our case the input value(s) is/are one or more instances of the concept Type. These instances are: integer, double, float, date, string, class, etc.,

The two ontologies are under development, we plan to elaborate and instantiate them in the cooperative work domain. The actual component description includes functional aspect which can be automatically exploited to generate the elements of the discovery ontology. However the component supplier must supplement this description by adding the non-functional aspect.

3. DESIGN AND IMPLEMENTATION OF THE SEC ENGINE

Our system provides a persistent component, called SEC, that can be loaded in the development environment during the project creation. SEC is persistent because it is always ready for execution by the developers. It contains the search process and it can access the repository of component descriptions.

Once SEC is running, the developer specifies the query by selecting the adequate criteria. An approximate comparison
between the specified query and the description of components in the component description repository is made by the \texttt{compare\_query\_description()} function. If there is a positive result, the system indexes the obtained description(s) to software component repository. So the \texttt{search\_component(ref\_component[])} function retrieves the appropriate component(s), where \texttt{ref\_component[]} is the list of the components references to retrieve. Then, the developer uses an application programme interface (API) to integrate the desired components in the current project. Finally to facilitate the component integration the developer can use the integration ontology.

The repository manager can also manipulate the SEC in order to manage the two repositories. He/she can add, modify and delete the component or/and the component description. The description of components is also managed by the repository manager.

SEC is a component responsible for the location of components. It executes the specified query and retrieves and presents relevant components. SEC requires no loading from software developers in development environments. In current development practices, the developer clicks on the SEC icons. Next, he chooses the non-functional features and the functional features which meets his needs. A dynamic query would be formulated and then executed automatically in order to deliver the adequate component.

Figure 3 shows the development of a simple chat application by using Visual BASIC. The Listbox component contains the identity of the recipient, the Text component contains the content of the message. The developer needs an ActiveX component that contains a method having two input parameters of string type that correspond to the address of the recipient, and the message. This component must also ensure the security of the message and establish the communication with the mailing server. To find this component, the developer selects the following search criteria: method and communicating objects as functional properties, component type as external property and security as static non-functional property. In the following step, a value is applied to each criterion previously selected. Thus, the developer can associates the label ’Send’ or any other synonym label to the criterion method. A query will be then formulated automatically when clicking on the search button. The components-result of the search will be sorted according to the degree of the similarity with the query.

There are four degrees of similarity:

- **Exact**: If component C and request R are equivalent concepts, we call the match Exact; formally, C \( \equiv \) R.
- **PlugIn**: If request R is sub-concept of component C, we call the match PlugIn; formally, R \( \subseteq \) C.
- **Subsume**: If request R is super-concept of component C, we call the match Subsume; formally, C \( \subseteq \) R.
- **Disjoint**: Otherwise, we call the match Disjoint; that is, C \( \cap \) R \( \subseteq \bot \).

In other words there is no element of the component description that corresponds to an element of the query.

The degrees of the similarity are organized in a discrete scale. Exact matches with a high level of used rate are clearly preferable than those with low level of used rate; PlugIn matches are considered the next best, Subsume matches are considered to be third best.

The developer can read the description of each component to understand their functionality details. Thus, the developer loads the adequate component in the chat application. The retrieved re-usable components are presented in the search result form in the decreasing order of similarity value. Each component is accompanied by its degree of similarity, name, access path and a short description. Developers who are interested in a particular component can load it in the tool bar. Finally, the developer can adapt it to the task or reuse it, as it is in the program, depending on the similarities degree. SEC provides also an interface for the repository manager in order to add, remove components and modify the components information. We formulate many queries without introducing the non-functional features. We notice that in many cases the delivery component does not meet
the developer needs. Actually, the non-functional features play a decisive role in the search quality. In the present implementation the repository is manually updated. Automating this step may be done on the base of techniques such as introspection [19] or invariant detection [22].

We developed two ontologies, called integration ontology and discovery ontology and also a search engine called SEC. The integration ontology describes the component internal structure in order to facilitate the integration of the desired component in the current project. The discovery ontology helps the developer to select the adequate component by using SEC.

Compared to other retrieval systems, our approach is unique in the following aspects:

(1) The first attempt to locating components is based on a software component.

(2) The developed system can be integrated in many software development environments.

(3) The whole system (The software development environment, the application and the search tools) is software component-oriented.

4. CONCLUSION

Software component retrieval research has been advancing very quickly over the past few decades. Researchers have experimented techniques ranging from probabilistic to learning techniques. At each step, significant insights regarding how to design and implement more useful software component search systems have been gained.

In this paper, we presented SEC, a persistent component in a repository. It delivers re-usable components and helps the developer to integrate the selected component into the current work. To ensure the success of these two tasks we developed two ontologies: A discovery ontology and an integration ontology.

The present implementation of SEC does not integrated the ontology based description technique. This point will be considered for future versions of this work.

In the future work, we plan also to compare our search engine with others and to extend our approach to exploit the search and the description of web services.

5. REFERENCES


