Classification and Retrieval of Software Components Using UML Descriptions

Pedro Reis dos Santos
Technical University of Lisbon
1049-001 Lisboa Codex
prs@digitais.ist.utl.pt

Abstract

As the number and complexity of components available for reuse increases, more accurate methods are need to classify the small differences among components. To provide better accuracy, the retrieval methods need to exploit those differences in a user controlled manner. UML structural information provide a rich set of primitives that describe an implementation independent component with significant detail. We propose a representation model for UML structural information that classifies each white box component. A fuzzy selection process retrieves a set of components that matches some of the desired criteria. Then, a set of mechanisms is then used to increase precision by gathering relevant information from each retrieved component in order to highlight differences among them.

Keywords: reuse software process, software components, component classification, component selection, fussy matching.

1 Introduction

The advances in computer technology and software development methods allow the construction of larger and more complex applications. The reuse of software components permits the development of such applications in a reasonable time with limited resources [11]. A high quality component may be reused over and over, reducing significantly the application's development cost, since it provides at almost no cost a documented, well designed, efficient, thoroughly tested and easy to adapt software block. These components are frequently available in library packages for a specific application domain distributed commercially at high cost. Other components of lower quality, generally corresponding to components not specifically designed for reuse, can still provide considerable benefits when compared to development from scratch. These component may be freely available, or at a very low cost, from private developers, universities and even commercial companies. During the last decade, millions of freely available applications have been tested, improved, documented and regularly used by many anonymous programmers, making such applications, and the components that form them, more tested and efficient than many commercially available equivalents.

2 Classification and selection methods

The reuse of a software component requires methods to retrieve useful components from a very large universe of available components. The search of a components through the exhaustive iteration through all components is only possible when the number of available components is very small. An enumerative classification mechanism is, therefor, required to group components in related areas of application, making the number of components to search much smaller. This technique is possible when dealing with comprehensive libraries covering a complete taxonomy

of concepts in a given domain since the characteristic of each component in the library do not overlap significantly with the others [9]. However, when there are many components providing different solutions to the same problem the classification mechanism will insert them all in the same group, making the exhaustive search within the group difficult and time [11]consuming. In order to make the classification process more precise hierarchical and faceted methods were developed [7, 18, 17]. Hierarchical classification methods are difficult to change and limits the kinds of relationships that can be represented. Faceted classification methods allow one facet to be changed without affecting others and can create complex relationships by combining facets and terms from a fixed vocabulary.

Although, when the number of similar components is very large, the used of a fixed vocabulary, common to all the above mentioned methods, proves to be insufficient. The stereotyping introduced by fixed vocabulary methods groups a significant number of similar components in the same category. For such cases the creation of the stereotype can not be performed on classification process but rather on the selection process when the requirements are specified. If the selection process is based on a small number of stereotypes then a large number of components will be retrieved resulting in method identical to a fixed vocabulary approach. However, when the requirements include a large number of characteristics it is possible to reduce the similarity evaluation and retrieve a small number of components that match very closely the requirements supplied. If the number of retrieved components is large then the requirements can be refined to include new characteristics that exclude some components until one, of just a few, are retrieved. On the other hand, if no component is retrieved, the requirements can be relaxed by excluding very demanding characteristics so that the closest match can be found.

The widespread use of public licensed free software components implies that components are available not only from comprehensive libraries covering a complete taxonomy of concepts in a given domain. This fact makes the classification process a recording of the relevant characteristics of the component, since the stereotyping occurs later when the components is being inspected by the selection process. The relevant characteristics may be manually inferred by the component developer or the catalog builder as is the case most fixed vocabulary methods. If no stereotyping is needed at the classification level then the classification process can automatically extract the relevant information from the component, generally in its source code form. The component's characteristics may be obtained from signatures, metrics or structural information. The description can be very detailed resulting in large memory recording requirements for classification and big computational overload for the selection process.

The work described in this article uses UML structural description information to record the characteristics of each component. These characteristics are included in a hierarchical description of identifiers that is more complete than signatures and can be complemented with information obtained from other sources such as metrics or facet classification methods. The selection process uses a set of operations on hierarchies to select the parts of the hierarchy that contain information related to the query performed. Then a weighted fussy comparisons of the identifiers in the sub-hierarchy is performed in order to evaluate the degree of similarity between the query and the retrieved components.

3 Related Work

The increasing number of components available for selection and the overlapping of many of their characteristics discourages the use of fixed vocabulary methods. Structural classification methods gather large quantities of unadulterated information from the original component allowing for more precise selection methods. In pair attribute-value methods, the component is classified using a set of predefined attributes. Every component must find a suitable value for each attribute [4, 7]. While faceted based classification methods must choose a value that best matches the component behavior from a small number of previously defined choices, in pair attribute-value methods the matching is done in the selection process and can

be controlled from query to query. In the system proposed in this article we extend pair attribute-value methods by structuring attributes in an hierarchy according to their scope and generality. Furthermore, the name of many of the attributes are not predefined but rather extracted from the component allowing comparisons between components and against the keywords in the query.

Methods of classification and selection of components based on signatures extend the principles of binding and linking of the programming languages and compilers to fussy cases [19, 13]. A routine to be reused does not need to have the same name, number and type of arguments. Name overloading and sub-typing allow several routine candidates, but when the call is made there only one implementation that matches the query. Component reuse based on signatures extends such binding mechanism to fussy cases where candidates with similar signatures might be considered good candidates. This implies that some adaptation, manual or automatic, must be performed before the component can be reused. The adaptation may require to change the order of the arguments, perform type conversions, or infer the value of some arguments based on other arguments or global data. The system described in this article uses the signature information provided by UML and is capable of many signature based matches by controlling the degree of similarity on the selection query.

Other methods of structural classification include the use of source code identifiers and data structures [6, 5] and pattern based methods [8, 2, 15]. The first are similar to signature matching but use data structures and other information instead of just using functions. Although, these methods are sensitive to the choice of names for the identifiers and data types, but can still be used in the proposed system by setting a small degree of similarity. Finally, structural classification based on formal methods uses mathematical descriptions of components and theorem proving to determine similarity between component descriptions and queries [1, 16, 3, 20]. Some of the drawbacks of formal based methods include the need for a formal description of the component which is not common, the difficulty to compute and control similarity and huge computation effort required even for a small number of components. The system proposed in this article does not address formal based descriptions, although, UML may include formal descriptions in a sub-language called OCL.

4 Component classification

The purpose of the classification process is to gather information relevant to the selection process. While the classification is performed once for each component, the matching is performed on every every query of the selection process. Therefor, the information should be then organized from the selection point of view. The proposed system builds an hierarchical data structure of attributes from UML structural information that can be used in the selection process.

4.1 Why UML ?

Many classification mechanisms are based on textual descriptions or source code. In either case the information relevant for the classification of the component constitutes a small part of all available data. An Unified Modeling Language – UML – description is a design level representation of the component excluding language dependencies and implementation details. At the design level, UML includes and extends most of the other design languages and provides a precise notation and well defined semantics. It is able to represent the information present in many other design languages with no loss and captures the essential characteristics of a component [10]. However, the current implementation of system described in this article uses only a partial UML description ignoring the information present in some diagrams and formal OCL descriptions.

An important aspect of UML is its ability to interface with programming languages. The UML is capable of producing a description of the UML information in many languages, that must then be filled with implementation details not present in UML. However, the ability to built an UML description from source, capturing the design information present in the source code, represents the most important UML characteristic for this work.

In this way, instead of parsing each source file and taking into consideration the peculiarities of each programming language, a single uniform, concise and compact description of the component is obtained. Furthermore, since almost every component must be subjected to some form of adaptation, the changes can be described in UML and then transported to the programming language in use. Even if no source code is available, although a manual UML description must be built from some form of documentation, the resulting UML description can then be used to compare components in equal term and plan changes and integration procedures.

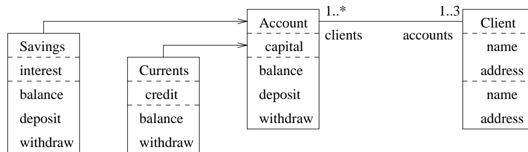

Figure 1: Simplified UML `accounting` example.

4.2 Representation model

The representation model maps almost all of the information contained in a UML class diagram into an hierarchy of attribute-value pairs. Since UML class diagrams are collection of classes, the first level of the hierarchy is represented by the names of each class. In the second level, each class is described in three branches representing the operations, attributes and associations of the class. The operations of a class are essentially signatures of procedures with a name, a return type and a set of named and typed arguments. The attributes of a class are typed data structures and are treated the same way as operations without arguments except that are placed on a different branch. Finally, a branch of associations that includes inheritance, aggregation and composition associations. Inheritance associations are described by the name of the super class, while aggregations and compositions are modeled by an association name, a class name and cardinality information. The UML example in figure 1 produces the hierarchical representation of figure 2.

An hierarchical description of a class includes the names of each element and the associated type in the respective branch. The separation by

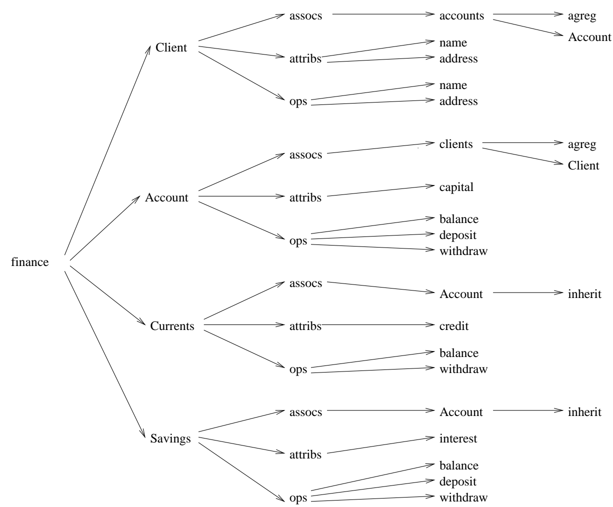

Figure 2: Hierarchical representation of the `accounting` example from figure 1.

branches is very important since each branch has a very different implication on adaptation cost of a selected component. For instance, a missing operation is not important since it can be seamlessly added, while adding an attribute might force the change of the constructors and imply changes on many function to keep the attribute consistent with the other internal state of the class. On the other hand changes on inheritance dependencies might be almost impossible to perform, and changes on some compositions and aggregations propagate changes to other classes.

The hierarchical model described tries to capture the compartments types of UML descriptions and explore the importance of their role in the component. The identifiers used to describe attributes, operations, arguments or data types are retained since the name of the identifier will be used to characterize its semantics. Therefor, a good choice of names can influence the selection or rejection of a component, even when using a thesaurus as a basis for weighted fussy comparisons.

4.2.1 Description enrichment

An UML description includes most of the information used to characterize a component at a design level. So, it will quite rare for two different components to have the same UML description, even on an abundance of similar compo-

nents. However, these descriptions can be long when compared to faceted classifications making the selection process time consuming in presence of a large number of components. A simple way to speedup the selection process is to include a facet branch where a small number of stereotyped attributes provides a fast way to exclude the components that do not match closely the query criteria. In the current implementation the faceted branch is inserted manually in each component. The selection process is then performed in two phases: a first selection and an iterative refinement. The first selection is, after all, a faceted based method that produces a number similar components of components that must then be processed by the iterative refinement. Since the majority of the components will fall in this category only a small fraction will be subjected to the hierarchical matching process.

Besides allowing the inclusion of a faceted branch, the hierarchical model allows the insertion of other branches containing information considered useful to some selections like the target language of the component, architecture or operating system dependencies, as well as, metric information such as complexity or performance. This additional information, even if not used in the selection process since it may be absent, can be decisive when a final decision must be made by direct inspection of a very small number of retrieved components.

5 Component retrieval

The selection process should retrieve a small number of candidate components that match the query criteria imposed by the selection requirements, in a small amount of time. A good selection process should not ignore components that match the requirements – recall – and should not retrieve components that do not match the requirements – precision [12, 13, 7]. In order to achieve high recall and high precision rates the selection process must create a query that rigorously formulates the requirements. Then the query must be able to produce an ordered list of matches that reflects the degree of similarity between the elements in the query and every component. The degree of similarity should reflect the adaptation effort needed to transform the retrieved component into an usable component for the target application [14].

5.1 Degree of similarity

The computation of the degree of similarity is the evaluation of a similarity function that returns a number representing the proximity between the query and a particular component. The numbers returned from the evaluation of every component against the same query are then sorted. Since the component is represented by an hierarchy of identifiers, the similarity function compares the identifiers used to represent the component with every query identifier. A fixed vocabulary approach may use boolean comparisons, where the number of exact matches is the returned value. However, if the name of each identifier can be selected from an very large set, there is a small probability of using the same name in the query and in the component representation. The use of a thesaurus allows the matching of synonymous names, enabling the use of the same fixed vocabulary techniques by performing the stereotyping in every query[12]. Therefor, the computation time is bigger than in fixed vocabulary techniques but the results are the same.

The main advantage of retaining the original identifier names in the classification process is the possibility of evaluating the similarity in case by case basis. Two synonymous names have the same essential meaning, but one cannot be used in all senses the other can. Some names have a broader meaning, or more specific meaning. That is, a name with a broader meaning can be used instead of a more specific one, but not the contrary (see figure 3). The fact that two names are not perfect synonymous, generally, implies that in a thesaurus each name must associate a similarity weight with every synonymous. Then, the similarity function must evaluate the weighted similarity between the query name and the names describing the component. The returned value is the summation of the similarities of every name in the query. The proposed system enables each user to supply his own thesaurus, besides the global thesaurus. The personal thesaurus can include regular expressions to group names with similar

writing, like plurals and verb tenses. This characteristic is specially important in Latin languages where verbs can have dozens of terminations according to time and subject, as well as names that can be male or female, singular or plural, or even support declinations.

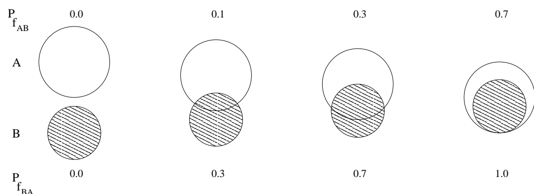

Figure 3: Weighted degree of similarity between the scope of different names.

Each identifier has a relative importance in the overall query, therefor, a factor must assigned to scale its influence in the value resulting from the query evaluation. When defining the requirements, the user must assign a large factor to increase the importance of an identifier, specify a small factor in order to decrease its importance, or associate a negative factor to exclude the components that include similar identifiers. Finally, the use of the same identifier in a different branch of the hierarchy can have a different meaning. Also the depth of the identifier in the hierarchy indicates its generality or even its function, for instance, as an argument name or as an operation name. For the above reasons, the evaluation of similarity can have a different meaning depending on the location of the matching identifier. Therefor, it is possible to restrict the matching algorithm to a particular area of the hierarchy by previously slicing the hierarchy through a set of operations.

5.2 Manipulation operations

Each component is represented by a single hierarchy containing all the information extracted from the UML class diagram. For many large components the generated hierarchies can be quite large increasing significantly the query time. Furthermore, by restricting the search to certain areas of the hierarchy more specific conclusion can be draw from the use of a particular identifier. The user can confine the search to certain areas of the hierarchy by slicing it with six manipulation operations. These operations can be combined by the query language to produce complex restrictions. The manipulation operations are:

granulation: limits the depth of the hierarchy. It allows for generic search and ignoring details, being useful in the first stages of the selection process to ensure a good recall.

specialization: limits the hierarchy to a single branch. It allows the search in specific contexts, being useful in the later stages of the selection process to ensure a good precision.

locate: limits the hierarchy to the branches that contain a specific name. It is able to characterize the use of a particular by providing the number of occurrences and the depth at which they occur.

qualification: limits the hierarchy to the branches with the same ancestor as a given name. It is useful to determine the presence of other identifiers in the same context.

union: groups two hierarchies in a single one. It is useful to determine, in the later stage of selection, if two or more components can be grouped in order to provide the requirements in the query.

differentiation: limits the branches of the hierarchy to those branches that are not present in another hierarchy. It is useful to evaluate what characteristics are absent from the selected component, or vice-versa, what characteristics does the component have that were not required. It is also very useful for comparing two selected components for a final choice.

5.3 Selection analysis

The selection process consists of series of queries that are expected to retrieve a smaller number of components on each query, since the number of restrictions can be tuned from the previous query. A restriction, or query element, is built by associating names and their associated factors to a subtree obtained by manipulating the component hierarchy with the operations above. The query

consists in the evaluation of each query element and adding the resulting values, for every component in the catalog. The query elements can be named and used repeatedly, allowing a successive refinement of the selection process combining different query elements in each query.

The process usually starts with a small number of, not too restrictive, queries to ensure a good recall. The first query can also be replaced by an auxiliary method like a faceted based method. When the number of components retrieved in the first choice is high, then the proposed process can perform a more precise selection since it records much more information from each component.

The refinement process uses the components retrieved from the first selection and through a series of more specific queries is, hopefully, able to elect the best candidate component for reuse. The refinement process is based on an analysis along three axis, in order to highlight small differences between the components and supply clues for the next query. The functionality of each of the three axis is:

descriptive: controls the amount of information supplied by the description of a component to the query. It relates to the depth of tree and the number of branches used when performing element queries. In some queries, excessive detail may produce erroneous results by focusing on too much detail, while other queries may require more information to find a proper match [12].

comparative: relates two or more queries and evaluates the comparative results of adding or removing query elements, or changing name factors. The use of an uncontrolled vocabulary is particularly important in this type of analysis since a different choice of names may produce divergent results.

development: addresses very similar and related components such as different production versions, and inherited or specialized versions of the same component. The development analysis studies more than one component by comparing the individual results of each one. This type of analysis is based on evaluating the cross differences produced by the differentiation operation, and is able to highlight small differences.

6 Conclusion

The method proposed in this article addresses the abundance of components where classical selection methods are unable to distinguish from similar components. The classification process uses UML class diagram descriptions to extract the classification information. The precision of the process is limited, essentially by the amount of information available on UML structural descriptions. Since detail is added at a deeper level in the hierarchy it does not lead to misinterpretation whenever the additional information is not needed. The use of identifiers produces richer representations that can be used when necessary and ignored by the use of thesaurus. The control over the detail involved in the queries makes the method attractive to explore small differences between similar components.

The system is not as automatic as other methods but is still much better than manual inspection where other algorithms return dozens of components classified in the same category. It is targeted for situations where the number of available components is very high, which is generally the case if the required component cannot be found in the local catalog and wider search must be performed. Most WWW search engines perform searches by filename or a small number of keywords, making millions of components freely available. If the component proves to be worthwhile, then it can be manually classified and integrated in the local catalog, even if an equivalent component exists in the home catalog.

References

[1] Dean Allemang and Beat Liver. Functional representation for reusable components. In *Workshop on Institutionalizing Software Reuse*, 1995.

[2] Patrick Ateyaert, Carine Lucas, Kim Mens, and Theo D'Hondt. Reuse contracts: Managing the evolution of reusable assets. In *OOPSLA*, pages 268–285, 1996.

[3] Yonghao Chen and Betty H. C. Cheng. Formally specifying and analysing architectural and functional properties of components for reuse. In *8th Annual Workshop on Software Reuse*, March 1997.

[4] Ernesto Damiani, Maria Grazia Fugini, and Enrico Fusaschi. A description-based approach to OO code reuse. *IEEE Computer*, 30(10):73–80, October 1997.

[5] Pedro Reis dos Santos and Rui Gustavo Crespo. Assisted selection of components using classified identifiers. In *7th IPMU*, pages 740–747, Paris, France, July 1998.

[6] Letha H. Etzkorn and Carl G. Davis. Automatically identifying reusable OO legacy code. *IEEE Computer*, 30(10):66–71, October 1997.

[7] William B. Frakes and Thomas P. Pole. An empirical study of representation models for reusable software components. *IEEE Transactions on Software Engineering*, 20(8):617–630, August 1994.

[8] Koen De Hondt, Carine Lucas, and Patrick Steyaert. Reuse contracts as component interface descriptions. In *ECOOP, 2nd International Workshop on Component-Oriented Programming*, pages 43–49, 1997.

[9] Mehdi Jazayeri. Component programming - a fresh look at software components. In *European Software Engineering Conf.*, September 1995.

[10] Philippe Kruchten. Modeling component systems with the unified modeling language. In *International Workshop on Component-Based Software Engineering*. Software Engineering Institute, 1998.

[11] Hafedh Mili, Fatma Mili, and Ali Mili. Reusing software: Issues and research directions. *IEEE Transactions on Software Engineering*, 21(6):528–561, June 1995.

[12] Rym Mili, Ali Mili, and Roland T. Mittermeir. Storing and retrieving software components: A refinement based system. *IEEE Transactions on Software Engineering*, 23(7):445–460, July 1997.

[13] R. T. Mittermeir, H. Pozewaunig, Ali Mili, and Rym Mili. Uncertainty aspects in component retrieval. In *7th IPMU*, pages 564–571, Paris, France, July 1998.

[14] Eduardo Ostertag, James Hendler, Rubén Prieto Díaz, and Christine Braun. Computing similarity in a reuse library system: An AI-based approach. *ACM Transactions on Software Engineering and Methodology*, 1(3):205–228, July 1992.

[15] Santanu Paul and Atul Prakash. A framework for source code search using program patterns. *IEEE Transactions on Software Engineering*, 20(6):463–474, June 1994.

[16] John Penix and Perry Alexander. Component reuse and adaptation at the specification level. In *8th Annual Workshop on Software Reuse*, March 1997.

[17] Rubén Prieto-Díaz. Implementing faceted classification for software reuse. *Communications of the ACM*, 34(5):89–97, May 1991.

[18] Rubén Prieto-Díaz and Peter Freeman. Classifying software for reusability. *IEEE Software*, 4(1):6–16, January 1987.

[19] Amy Moormann Zaremski and Jeannette M. Wing. Signature matching: A tool for using software libraries. *ACM Transactions on Software Engineering and Methodology*, 4(2):146–170, April 1995.

[20] Amy Moormann Zaremski and Jeannette M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, October 1997.