Zbornik konference

Razvoj in prenovitev IS

Četrta mednarodna multi-konferenca

Informacijska družba – IS2001

Proceedings of the Conference

Development and reingeneering of IS

Fourth International Multi-Conference

Information Society – IS2001

Uredil / Edited by

Ivan Rozman

25. oktober 2001 / 25th October 2001

Ljubljana, Slovenija

CLASSIFICATION AND SELECTION OF SOFTWARE COMPONENTS USING UML DESCRIPTIONS

Pedro Reis dos Santos
Department of Electrical Engineering and Computation
Instituto Superior Técnico
Av. Rovisco Pais,1 1049-001 Lisboa, Portugal
Tel: +351 21 8419340; fax: +351 21 8417499
e-mail: prs@digitais.ist.utl.pt

ABSTRACT

The number and complexity of components available for reuse is becoming very large requiring more accurate classification methods. Object-oriented languages produce, due to extensive use of inheritance and overloading, a large number of components with small differences that are difficult to cope with existing classification and selection methods. To provide better accuracy, the selection method needs to exploit those differences in a user controlled manner. UML structural information provides significant detail enabling a distinctive classification of similar components. A selection process retrieves a first set of components that match the developer requirements criteria and, a set of mechanisms is then used to increase precision.

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 [7].
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 can be available in library packages for a specific application domain distributed commercially at high cost, be freely available, or have not been specifically designed for reuse and still provide considerable benefits when compared to development from scratch. During the last decade, a large number of components and decomposable applications have been tested, improved, documented and regularly used by many developers, making such components valuable for reuse. These component may be freely available, or at low cost, from private developers, universities and commercial companies.

In order to reuse a component, the developer must know of its existence and recognise the application context. However, as the number of available components increases, the developer requires tools to assist him in the retrieval of the component that better matches the requirements. The retrieval process is based on a selection process that chooses, from a set previously classified component characteristics, a component from the reuse library that requires the smallest amount of adaptation effort in order to be reused in the new context.
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.

2 RELATED WORK

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 small [7]. An enumerative classification mechanism, that groups components in related areas of application, 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 [5]. In order to make the classification process more precise hierarchical and faceted methods were developed [14,13]. 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.
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 [1]. While faceted based classification methods must choose a value that best matches the component behaviour 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. 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 [15, 9]. Other methods of structural classification include the use of source code identifiers and data structures [3, 2] and pattern based methods [4, 11]. The first are similar to signature matching but use data structures and other information instead of just using functions.
Finally, formal methods use mathematical descriptions of components and theorem proving to determine similarity between component descriptions and queries [12,16]. 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.

3 COMPONENT CLASSIFICATION

The purpose of the classification process is to gather information relevant to the selection process. The proposed system builds an hierarchical data structure of attributes from UML structural information that can be used in the selection process. An Unified Modeling Language -- UML -- description is a precise notation and well defined semantics of a design level component representation excluding language dependencies and implementation details [6]. An important aspect of UML is its ability to interface with programming languages. The UML is capable of producing code scheletons in many programming languages and capturing the design information present in the source code. In this way, instead of dealing with the peculiarities of each programming language, a single uniform, concise and compact description of the component is obtained.
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 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.
An hierarchical description of a class includes the names of each element and the associated type in the respective branch. The separation by 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 implies 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. Therefore, 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 COMPONENT SELECTION

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 [8, 9]. 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 [10].

4.1 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 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 [8]. 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. 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.

4.2 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. 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.

4.3 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 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 [8].

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.

5 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] Ernesto Damiani, Maria Grazia Fugini, and Enrico Fusaschi. A description-based approach to OO code reuse. *IEEE computer,* 30(19):73-80, October 1997.

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

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

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

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

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

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

[8] 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.

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

[10] 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.

[11] 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.

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

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

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

[15] 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.

[16] 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.