AUGMENTATION OF COMPONENT RETRIEVAL USING CEREMONIAL METHODS

Vikas Verma
Associate Professor, CGC, Landran

Munish Mehta
Assistant Professor, CGC, Landran

Dr. Naveeta Adlakha
Assistant Professor, GCW, Karnal

ABSTRACT

Software reuse is the process of implementing or updating software systems by using existing software components. Software reuse is only effective if it is easy to locate and update a reusable component than to write it from scratch. There are two main problems in software reuse i.e. classifying software modules in a component and identifying relevant software components in a repository. Software component retrieval is an important task in software reuse as all the components must be found before they can be reused. Many researchers have proposed various techniques to search and retrieve components from software repository.

The existing component retrieval techniques can be divided into five main categories: Faceted search, Folksonomy classification, Keyword search, Signature matching and Ant colony optimization algorithm. In case we have exact queries belonging to single or restricted domain in that case Ant Colony Optimization algorithm [9] is much better than remaining four techniques because it is robust, scalable and expandable. This research paper suggests that ant colony optimization can be enhanced by using ceremonial methods in component representations so that components relevant to the requirements can be retrieved.

Keywords- Augmentation, component-based software engineering (CBSE), Ant Colony optimization (ACO), Ceremonial methods.
1. INTRODUCTION

The component-based software engineering (CBSE) lay emphasis on component based software development of applications so that the applications are easy to develop, maintain and expand. The main constituent of component-based development (CBD) is component. The term component has been defined in many different ways. We use the general definition of Brown [1], who defines that “a component is an independent and self-sufficient part of a system having complete functionalities”. When a component is developed it is intended that it can be integrated with other systems. These components get stored in software repositories. These repositories may contain thousands of components. A software component repository is storage location from which components may be retrieved and installed on a computer.

However, finding and reusing appropriate software components is often very challenging, particularly when faced with a large collection of components and little documentation about how they can and should be used[7]. The ant colony based optimization algorithm, can help re-user to identify and retrieve software component but only in a specific domain. This can be enhanced by using ceremonial methods in component representations so that components relevant to the requirements can be retrieved. The use of ceremonial methods for software and hardware design is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analysis can contribute to the reliability and robustness of a design. However, the high cost of using ceremonial methods means that they are usually only used in the development of high-integrity systems, where safety or security is of utmost importance [5]. Ceremonial methods mean the mathematics and modeling applicable to the specification, design, and verification of software and hardware systems.

2.0 ANT COLONY OPTIMIZATION TECHNIQUE

Ant colony optimization (ACO) is a population-based meta heuristic that can be used to find approximate solutions to difficult optimization problems. [2]

In ACO, a set of software agents called artificial ants search for good solutions to a given optimization problem. Ant colonies, are distributed systems that, in spite of simplicity of their individuals, present a highly structured social organization. And thus, ant colonies can accomplish complex tasks which can’t be handled by a single ant. Dorigo, M. et al. [3], with this observation proposed methodology based on the behavior of the real ant colonies.
The main idea is that the self-organizing principles which allow the highly coordinated behavior of real ants can be exploited to coordinate populations of artificial agents that collaborate to solve computational problems. Ants use a substance named pheromone for direct communication between them. While walking from food sources to the nest and vice versa, ants deposit pheromones on the ground, forming in this way a pheromone trail. Ants can smell the pheromone and they tend to choose, probabilistically, paths marked by strong pheromone concentrations.

Ant colony algorithm based technique generates rules to store and then identifies the component from software repository for possible reuse. This technique helps user in organizing and storing components in repository and later help in identifying most appropriate component for given context.

First step is to make use of keywords, their synonyms and their inter-relationships for searching of components. Then it makes use of ant colony optimization, initial pheromone of one is assigned to all domain representative terms of components. By updating pheromone for participating terms and non-participating terms iteratively and by calculating the quality of each rule generated, it leads to quality rules to retrieve the reusable components. As we analyze natural language based simple keyword, so the search suffers from inherited ambiguity problem. Because the search space is very large, in this work an ant colony optimization based component classification technique is proposed. Here the major goal is to generate rules for classifying components. For a particular context, various rules are generated and then their quality is checked. Various terms in form of ants are selected for a given context, and then their interrelationships are derived. Checking deposited pheromone on various terms generates rules. Pheromone is deposited on a particular path in here is term and their relationship, if it is contributing for current context otherwise pheromone is evaporated and that term will not play role in rule generation. Quality of a rule is calculated on the basis of various parameters.

3.0 STEPS OF COMPONENT RETRIEVAL USING CEREMONIAL METHODS

Ceremonial methods can be applied at various stages throughout the development process. i.e.

- Design
Ceremonial methods may be used to give a description of the system to be developed, at whatever level(s) of detail desired. This formal description can be used to guide further development activities, additionally, it can be used to verify that the requirements for the system being developed have been completely and accurately specified.

- **Development**

Once a formal specification has been produced, the specification may be used as a guide while the actual system is developed during the design process (i.e., realized typically in software, but also potentially in hardware). Like, if the formal specification is in an operational semantics, the observed behavior of the actual system can be compared with the behavior of the specification. Additionally, the operational commands of the specification may be open to direct translation into executable code. If the formal specification is in an axiomatic semantics, the preconditions and post conditions of the specification may become assertions in the executable code.

- **Verification**

Once a formal specification has been developed, the specification may be used as the basis for providing properties of the specification.

### 3.1 Component Specification

The specification for a software component corresponds to the specification of an abstract data type (ADT) and a set of methods that operate on that abstract data type. Each method is specified by an interface, type declarations, a precondition, and a post condition. The interface of a method describes the syntactic specification of the method. The typing information describes the types of input and output parameters and internal (local) variables. The precondition describes the condition of the variables prior to the execution of the method whose behavior is described by the post condition. An abstract data type (ADT) is a behavioral notion and can be implemented by many different classes. A class is a program module that implements an abstract data type. A subtype is also an ADT and each of whose objects behaves similarly to objects of its super types. A subclass is an implementation that is derived by inheritance from its super class. A subtype represents a behavioral relationship. When a subtype is derived from some super type the objects behavior with this subtype can be verified according to the objects with its super type instead of verifying this subtype again. In contrast a subclass relationship is a purely implementation
relationship. The generality relationship in our proposed work will similar to the super type-subtype relationship.

### 3.2 Building the Lower-level Organization

It is important to determine the generality relationship between any pair of components. The straightforward approach is to construct the lower-level organization by performing a pair-wise comparison between all components. The transitivity property of the generality relationship can be exploited in order to reduce the computational complexity of building the lower-level hierarchy. If A clause B and B clause C then the relation A clause C is automatically established without having to compare components A and C. Using the recursive operation to build the lower-level organization may reduce the computational time of construction.

### 3.3 Building Higher-Level Organization

Once we apply Minimum Spanning Tree the software components can be grouped into disjoint clusters in a set of graphs. To form a connected organization of software components, a conventional clustering algorithm [8] is applied to the most general components obtained from the Minimum Spanning Tree that is the roots of trees and the top elements of the lattices in set of graphs.

The objective of clustering is to form a set of clusters in such a way so that the intercluster similarity is low and the intracluster similarity is high. Applying a clustering algorithm to the most general components of the lower-level organization leads to the generation of the higher-level organization of the component library. The similarity between two components S and S’ is used as the basic criterion to determine clusters. In general, the criterion used to evaluate similarity determines the shape of the resultant clusters.

### 3.4 Reusable Candidates Searching

The construction of the hierarchy is performed in two stages beginning with the lower-level that further used in the construction of the higher-level organization. In contrast the search and retrieval process proceeds from the higher-level organization to the lower-level. At the higher-level organization a query is mapped to some index that indicates the starting nodes within the organization at which the searching algorithm have to begin. The remaining portion of the higher-level organization and the corresponding lower-level is searched using formal reasoning techniques thus providing an exact determination operation.
Three possible classes of existing specifications can be retrieved using logic reasoning techniques i.e. an exact match to the new specification, or a component more general than the current specification, or a component more specific than the current specification.

3.5 Repository Browser

The browser enables a user to graphically traverse the hierarchically organized specifications of software components by making information about class hierarchies. The browser facilitates the iterative process of developing and accessing reusable specifications. With the Repository Browser you can execute commands like copy, move, rename, directly on the repository. The repository browser looks very similar to the Windows explorer, except that it is showing the content of the repository at a particular revision rather than files on your computer. In the left pane you can see a directory tree, and in the right pane are the contents of the selected directory. At the top of the Repository Browser Window you can enter the URL of the repository and the revision you want to browse. Just like Windows explorer, you can click on the column headings in the right pane if you want to set the sort order. Here the context menus are also available in both panes. The context menu for a file allows you to:

- Open the selected file, either with the default viewer for that file type, or with a program you choose.
- Save an unversioned copy of the file to your hard drive.
- Show the revision log for that file, or show a graph of all revisions so you can see where the file came from.
- Censure the file, to see who changed which line and when.
- Delete or rename the file.
- Make a copy of the file, either to a different part of the repository, or to a working copy rooted in the same repository.
- View/Edit the file's properties.

You can also use the repository browser for drag-and-drop operations. If you drag a folder from explorer into the repo-browser, it will be imported into the repository. Note that if you drag multiple items, they will be imported in separate commits. If you want to move an item within the repository, just left drag it to the new location. If you want to create a copy rather than moving the item, Ctrl-left drags instead. When copying, the cursor has a “plus” symbol on it, just
as it does in Explorer. If you want to copy/move a file or folder to another location and also give it a new name at the same time, you can right drag or Ctrl-right drag the item instead of using left drag. In that case, a rename dialog is shown where you can enter a new name for the file or folder. Whenever you make changes in the repository using one of these methods, you will be presented with a log message entry dialog. If you dragged something by mistake, this is also your chance to cancel the action. In searching for a reusable candidate the user may select the root node to begin the search, or the system will select the root node based on the syntactical components of the query. E.g. developing a multiple type Software component repository using ceremonial methods for an Operating system,

- Define which type of software we require like “educational software”, “science software”, “mathematical tools”, “web applications”, “games”, “sound and video” etc. Also providing each software a specific name like “electrical cad” in “educational software” type, “solitaire” in “games” etc.
- Define a simple data structure (Array) which is capable of holding searched software. All the searched software is placed in it by using an Insertion algorithm. Software names will be used as variables in it.
- Apply the sorting algorithm on this software array which sorts this software on the basis of their name and type i.e. the category they belong.
- Find how software in this sorted list is related with another. If there is any relation between them in that case they are grouped as a cluster else find their respective cluster.
- Create a category as per our requirements and place the related software cluster in that category.
- Finally we will develop a repository browser to provide the graphical view of this repository to the user.

4.0 LIMITATIONS OF CEREMONIAL METHODS

There are several reasons why ceremonial methods are not used as much as they might be, i.e. Expense: Because of the rigor involved, ceremonial methods are always going to be more expensive than traditional approaches to engineering. However, given that software cost estimation is more of an art than a science, it is debatable exactly how much more expensive formal verification is. In general, ceremonial methods involve a large initial cost followed by less
consumption as the project progresses; this is a reverse from the normal cost model for software development.

**Limits of Computational Models:** While not a universal problem, most ceremonial methods introduce some form of computational model, usually hamstringing the operations allowed in order to make the notation elegant and the system provable. Unfortunately, these design limitations are usually considered intolerable from a developer's perspective.

**Usability:** Traditionally, ceremonial methods have been judged on the richness of their descriptive model. That is, 'good' ceremonial methods have described a wide variety of systems, and 'bad' ceremonial methods have been limited in their descriptive capacities. While an all-encompassing formal description is attractive from a theoretical perspective, it invariably involved developing an incredibly complex and nuanced description language, which returns to the difficulties of natural language. Case studies of full ceremonial methods often acknowledge the need for a less all-encompassing approach.

### 5.0 ADVANTAGES OF CEREMONIAL MODELS

Ceremonial methods offer additional benefits outside of provability, and these benefits do deserve some mention.

**Discipline:** By virtue of their rigor, ceremonial systems require an engineer to think out his design in a more thorough fashion. In particular, a formal proof of correctness is going to require a rigorous specification of goals, not just operation. This thorough approach can help identify faulty reasoning far earlier than in traditional design.

The discipline involved in formal specification has proved useful even on already existing systems. Engineers using the PVS system, for example, reported identifying several microcode errors in one of their microprocessor designs.

**Precision:** Traditionally, disciplines have moved into jargons and formal notation as the weaknesses of natural language descriptions become more glaringly obvious. There is no reason that systems engineering should differ, and there are several ceremonial methods which are used almost exclusively for notation.

For engineers designing safety-critical systems, the benefits of ceremonial methods lie in their clarity. Unlike many other design approaches, the formal verification requires very clearly
defined goals and approaches. In a safety critical system, ambiguity can be extremely dangerous, and one of the primary benefits of the formal approach is the elimination of ambiguity.

6.0 CONCLUSION

The proposed system constructs a two-tiered organization repository from formal specifications. Thus the organization can help users store browse and retrieve existing reusable components. Proposed system will be extended in several aspects. Efficient techniques are being used to determine functional similarity between two software components. The abstraction scheme to form meta-nodes of software components will also be further investigated. An efficient searching algorithm that includes hashing and reasoning schemes will be developed. The specifications representing the inheritance relationship and the generality of software components, needs to be studied in order to exploit the properties of object-oriented development techniques. Our work provides a framework for a software reuse and retrieval system.

7.0 REFERENCES


