National Conference on Recent Research in Engineering and Technology (NCRRET-2015)
International Journal of Advance Engineering and Research Development (IJAERD)
e-ISSN: 2348 - 4470 , print-ISSN:2348-6406

Conceptual Case Based Approach to Retrieve Java Class Library

M. K. Patil, Dr. P. P. Jamsandekar
Research Student, BVU AKIMSS, Solapur
Professor, BVU IMRDA, Sangli

Abstract:
The CBR system is improved by using clustering algorithm with k – NN algorithm. Cases in the class libraries are clustered into smaller sub sets. The structure is represented by hierarchical manner. The similarity approach is examined by comparing the structures of retrieval of class libraries. The class library is maintained with the class, interface, and packages of JAVA programming language.
WE have proposed a model where each case in the repository is an active case and where a hierarchical structure provides an organization analogy useful to implement the retrieval mechanisms and rules.
Keywords: class library, clustering, retrieval

Introduction:

Case-based reasoning (Kolodner, 1992) means using old experiences to understand and solve new problems. In case-based reasoning, a reasoner remembers a previous situation similar to the current one and uses that to solve the new problem. Case based reasoning can mean adapting old solutions to meet new demands; using old cases to explain new situations; using old cases to critique new solutions; or reasoning from precedents to interpret a new situation or create an equitable solution to a new problem. The CBR (Aamodt, 1994) process can be represented by a schematic cycle, as shown in Figure (a). CBR typically as cyclical process comprising the four REs:

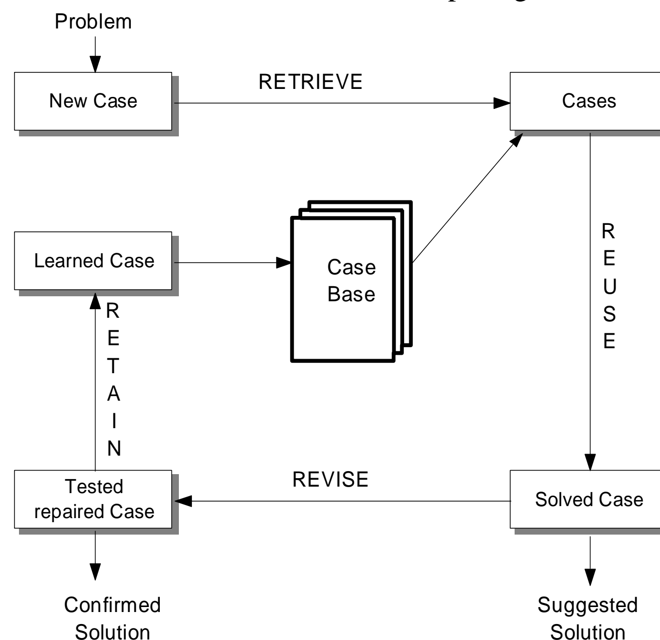

Figure (a): CBR Cycle

Retrieve the most similar cases; during this process, the CB reasoner searches the database to find the most approximate case to the current situation.

Reuse the cases to attempt to solve the problem; this process includes using the retrieved case and adapting it to the new situation. At the end of this process, the reasoner might propose a solution.

Revise the proposed solution if necessary; Since the proposed solution could be inadequate, this process can correct the first proposed solution.
Retain the new solution as a part of a new case.

This process enables CBR to learn and create a new solution and a new case that should be added to the case base. It should be noted that the RETRIEVE process in CBR is different from the process in a database. If you want to query data, the database only retrieves some data using an exact matching while a CBR can retrieve data using an approximate matching.

Case Retrieval
Case retrieval here represents the process of extracting the cases from the case base, which are closest to the current case of the initial problem. To extract similar cases and the best case there should be some selection criteria which determine the closeness of the current case to the stored cases. The case retriever generally searches the entire case to find the features of that case similar to the features of the current case however the entire case does not exists, the portion of a case matches. The retrieved case can also be modified by taking solution of another case. There are many case retrieval techniques to carry out searching some of them are the k-nearest neighbors (k-NN), decision trees, and their derivatives. These techniques use similarity metric that calculate the closeness among cases.

Nearest-neighbor retrieval
The nearest-neighbor retrieval computes the similarity by calculating the weights of the features of the case retrieved with those of the current case. If the weighted sum of its features that match the current case is greater than other cases than that case is retrieved. Features that are considered important in a problem-solving situation are weighted heavily in the case-matching process.

Inductive approaches
An inductive approach creates the decision trees. This may reduce the query search time.

Knowledge-guided approaches
This approaches uses domain knowledge to determine the features of a case that are important for retrieving that case in the future. It is also considered an effective searching approach.

Validated retrieval
Validated retrieval has two phases. First involves the retrieval of all cases that match the important features of the current case. Another involves deriving more discriminating features to match the current situation from the group of retrieved cases. (Shiu, 2003)

Limitations of Existing Retrieval Techniques
Nearest-neighbor retrieval and inductive retrieval both have their strengths and weakness. The choice of retrieval techniques in CBR applications requires experience and experimentation. Nearest-neighbor retrieval is used without any pre indexing. If retrieval time becomes an important issue, inductive retrieval is preferable. Nearest-neighbor retrieval is a simple approach that computes the similarity between stored cases and new input case based on weight features.
A typical evaluation function is used to compute

$$similarity(Case_I, Case_R) = \frac{\sum_{i=1}^{n} w_i \times sim(f_i^I, f_i^R)}{\sum_{i=1}^{n} w_i}$$

nearest-neighbor matching (Kolodner, 1992)as
Where,
w_i is the importance weight of a feature,
sim is the similarity function of features, and
f_i^I and f_i^R are the values for feature i in the input and retrieved cases respectively.

Nearest-neighbor retrieval and inductive retrieval are widely applied in CBR applications and modules. Table shows the comparison between nearest-neighbor retrieval and inductive retrieval

Retrieval Techniques	Strength	Weakness
Nearest Neighbour Retrieval	Simple	Slow retrieval speed when the case base is large
Inductive	Fast retrieval	Depends on pre-indexing which is a time-consuming process

Retrieval	speed	Impossible to retrieval a case while case data is missing or unknown

In some CBR modules, both techniques are used: inductive indexing is used to retrieve a set of matching cases, and then nearest-neighbor is used to rank the cases in the set according to the similarity to the target case.

Work Domain: Java Class Library

The purpose of a case-based retrieval and reuse module is to help the developer to locate reusable code and to aid in program understanding and adaptation. The module matches Java classes from the class repository (base cases) to the target case (the class under construction) and then suggests similarities between them.

It enhances Java's reusability that it automates, ensures the quality of program. It will separate the Java's components as packages, classes, methods based on structure of the class and signature. Furthermore it ensures that the automated retrieval and adaption strategies will be immediately useful and work with existing software repositories.

Java Reflection enables Java code to discover information about the fields, methods and constructors of loaded classes, and to use reflected fields, methods and constructors to operate on their underlying counterparts at runtime (James Gosling, 1996). This capability allows us to extract feature descriptions from compiled classes without having access to the source code.

Java has a set of powerful mechanisms that directly support software reuse. However, the developer must have a sufficient knowledge of the language environment to be able to construct a mental mapping from existing object classes to the class that he wishes to construct. Java supports this to some extent in that it is possible to inherit the structure and functionality of an existing class and only specify new behavioral features in the new object class.

Proposed Work

A CBR system is said to be successful if it designs an efficient and effective case retrieval mechanism. K-Nearest Neighbor (KNN) search method searches the entire case base to retrieve K prior cases with minimal dissimilarities. One of the main drawbacks of the CBR is, the dimensionality problem: the uncontrolled growth of the case bases may result in the degradation of the performance of the system as a direct consequence of the increased cost in accessing memory.

The solutions of similar prior cases can be used to solve the problem of the new case. And to discriminate the similar cases from other cases we can cluster those cases that have similar solution parts. The Clustering techniques focus our search to the cluster that has similar case to that of our problem description. This technique deals with the Java's case library.

In the Java programming language the base case descriptions can be constructed from the software artifacts themselves using Java reflection.

The case-based reuse module supports retrieval and reuse of classes based on their signatures (methods return types and arguments etc.), which in this case is viewed upon as cases. From these signatures one may also extract some knowledge about what kind of component this is. The reuse component may suggest mappings between signatures of a retrieved case and the target, and the user may accept or discard the suggestions. In addition the reuse module suggests *how* to reuse a class, either by extension, or by lexical reuse of source code (if it is available).

Java's reflective capabilities are used to extract case descriptions from compiled Java classes, and case-based reasoning is applied to support retrieval and adaptation of reusable components. The purpose of the module is to localize potentially reusable code and to support the programmer in her program understanding and adaptation of the code.

(Bjørnar Tessem, 1999) It was researched that, the set of features that can be automatically extracted utilizing the Java reflective capabilities (e. g., method signatures, field types, inheritance

information, etc.) can be effectively used to retrieve components for subsequent reuse. It will decrease the effort required to retrieve the most plausible class in program by the developer.

Following figure (b) is a conceptual proposed work. It has the CBR phases such as Reuse Module, Retrieve Module and retain module where we have maintained the existing class library repository and new added one. To interact with system, we have a user interface where user can input the problem. The system admin module checks the problem and controls on entire module.

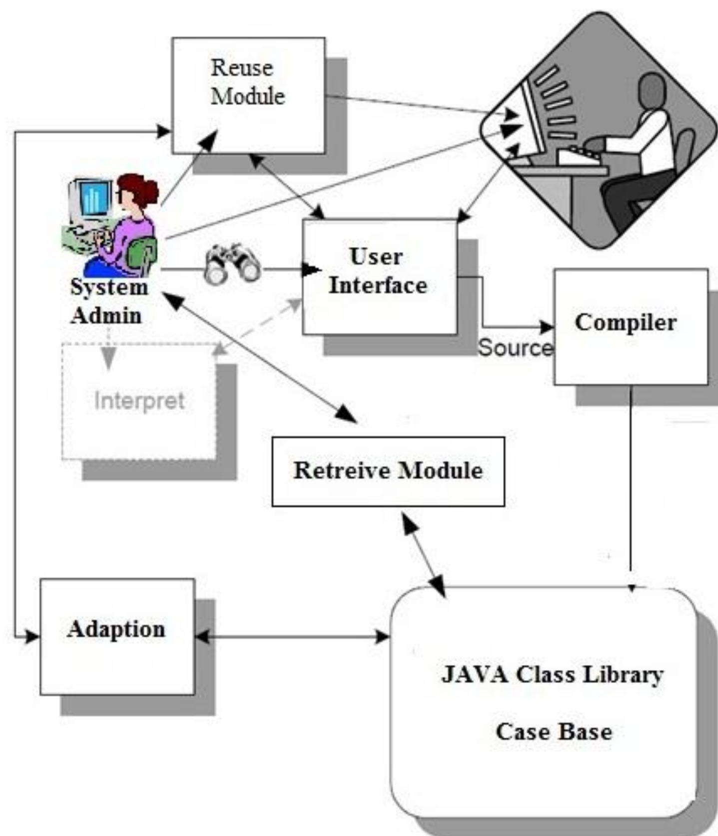

Figure (b): Proposed Module

System Admin Module

The System Admin Module monitors the programmer's implementation via the extracted signatures of the partial class specification and compiles the code when needed.

A *Compiler, interpreter* would allow to test, run un-compiled Java statements; thus check the syntax of the expressions. This is however not required but may be a great idea for further development.

The System Admin then passes the task on to the appropriate Retrieve Modules. Each cluster has more specific knowledge about a certain group of cases. The cluster represents a case base to the relevant to the package as suggested (James Gosling, 1996) and a single Case Base is in reality a single case description of a Java class.

The Manager also monitors the coding. The user specifies how often (a time interval) or under what condition (number of code lines in the editor) the Manager should interpret the code and make a target case for the retrieval process.

The Retrieve Module

Java programs are organized as sets of packages. Each set has its own set of names for types, which help to prevent name conflicts. The naming structure for packages is hierarchical which is convenient for organizing related packages in a conventional manner. A Retrieve Module represents a single package in the repository as a "package case". A package case consists of all the types (method return types, fields and argument lists) of all the classes in a Java package.

Each value has a significance attribution. The significance of a type in a certain package is a calculation of its number of occurrences in a case in relation to occurrences in each other package and in the whole repository. The significance of a type is hence a value used in the matching with the target case's types. If it finds that classes in the package contain highly significant types for this particular target case it will pass the target case to the case base for further matching. The clustering algorithm won't store the relevant case which having good match instead of best match. It there is no significance it doesn't store in case base repository.

The Case Base Repository

The individual case, or Case Base, possesses its own case description. The descriptions are created using Java's reflective facilities. Java allows any class to be asked for its methods, fields, constructors, inheritance information, and other information at run time (Sun Microsystems, 1999). Java's syntactic reuse construct is the import statement. Java uses an environmental variable called CLASSPATH to establish where to search for classes that are mentioned as import statements.

The retrieve module supported case-based retriever traverses the directories on the CLASSPATH environmental variable, extracts all the feature information for each class in a pre-processing step and stores that information in a file associated with the class for later use.

Each file is associated with a Case Base. When a base case is matched with a target case it obtains a similarity value based on threshold. This value (between 0-1) determines if the case is a user for reuse. If the match is good (greater than a predefined threshold) the Case Base offers itself as a potential case for retrieval. The user specifies the threshold the case has to match to be considered as a potential case for reuse. If the match evaluates to half of the threshold, the Case Base continues to live in memory but does not send an event.

Similarity Matrix

The estimation of the similarity between the target and the base is developed by (Bjørnar Tessem, 1999). The Case Bases (base cases) estimate a similarity to the target class using similarities between pairs of methods, constructors, and data fields. To establish a similarity for a base case it does the following steps:

1. For each method, constructor, and data field in the base class use its signature to compute a similarity to each of the method signatures of the target.
2. For each method, constructor, and data field in the target select the most similar entry in the base class description and match it to this entry. As the entries in the base class are selected, mark them not-selectable.
3. The total similarity is the sum of the similarities of the selected matches in the target case.

For constructors only argument similarity counts, whereas for data fields type and name similarity counts. The similarities grouped into similar types of clusters using clustering technique.

At last, after retrieving a user's case the system provides the user with feedback about what it has carried out. The system gives information about which alternatives to the programmer has. The alternatives consist in either proposing **adaptation of the retrieved case**(s) with help from the reuse assistant, to continue the adaptation independently from the assistant, or to continue the programming with new searches for others and maybe more appropriate cases for potential reuse. The programmer is completely free to follow system's advice or to ignore it.

Conclusion:

Modern programming languages, especially object-oriented languages, make use of large libraries of reusable components (e.g. class definitions). We want to make it easier for programmers to make use of the resources contained in these libraries.

The System Admin Module collects all retrieved cases from the different packages. The best cases are sorted by how well they match the target case

and are presented to the user. The leftover cases (or Case Bases) are kept alive, as they may become potential cases for reuse in the further development of the target case. In the next round of matching these leftover cases will be re-matched without having to re-read their features and invoke them again. The System Admin Module responsibility in this environment is to independently execute the case-based matching cycle at the right time and with satisfying feedback.

References:

1. Aamodt, E. P. (1994). *Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches* (Vol. 7). AI Communications. IOS Press.

2. Bjørnar Tessem, R. A. (1999). *Case Based Support for RAD*. Skokie, Illinois: SEKE'99 Proceedings - 11th Conference on SE and KE.

3. James Gosling, B. J. (1996). *The Java Language Specification, The Java Series* (1 ed.). Addison-Wesley.

4. Kolodner, J. L. (1992). *An Introduction to Case-Based Reasoning*. College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280,: Artificial Intelligence Review.

5. Morisbak, S. I. (June 22, 2000). *The Road to ASCRARAD: The Development of Agent Support for a Case-based Reuse Application for RAD.*

6. Shiu, S. K. (2003). *Foundation of Soft Case Based Reasoning*. Wiley series on intelligent systems.

7. Sun Microsystems, M. V. (1999). *Java 2 SDK, Standard Edition Documentation*. CA.