Journal of Computer Sciences 2 (1): 29 -32, 2006
ISSN 1549-3636

JBOOM: Java Based Object Oriented Model of Software Configuration Management

[1]Bhavya Mehta and [2]S.K.Muttoo
[1]Department of Computer Science, College of Vocational Studies, University of Delhi, India
[2]Department of Computer Science, University of Delhi, India

Abstract: Most of the present Software Configuration Management systems deal with version and configurations in the form of files and directories, the need today is to have a Software Configuration Management system that handles versions and configurations directly in terms of functions (program module). A major objective of this research is the use of Java in the Software Configuration Management systems. An object-oriented language provides both design and implementation in an integrated manner. We have proposed a model that expresses change evolution in terms of class hierarchies. As the changes evolve so does the class hierarchy, it can be further extended and existing classes can be extended.

Key words: Software configuration management, object oriented modeling, versions, configurations, change control, Java, objects, classes, interfaces, methods

INTRODUCTION

An object-oriented model provides an abstraction for modeling SCM entities and relationships. An object-oriented model provides both design and behavior in an integrated manner. An ideal way of representing growing changes is to use objects and classes. A major emphasis of this study is to represent the changes in terms of an a class and its instances. Our proposed model uses Java objects and classes to correspond to changes that are made in the system components[1]. The classes selected in our model reflect the most common changes that would arise in the normal evolution of software systems[2].

Acronyms

SCM	Software Configuration Management
API	Application Programming Interface
IT	Information Technology
RND	Research and Development

An IT consultancy organization example: Consider an IT organization, IT Consultancy Services (ITCS).

It has a centralized In-House Division for maintaining a centralized control over the various departments in the firm. Various other departments in ITCS like Admin (for administration), Finance, Rand D (for research and development), Networking, Products have their own internal IT divisions. The developers at In-House create a library ver1.0 and distribute it to all the other departments in the organization. Now after some time In-House department feels some new features may be added to the same library making it ver1.1, at this stage the changes are minor so do not affect the application users. Later on the library developers at In-House department add some security final checks, deporting and archival features to the library API, proposing a new version of the library ver1.2, in doing so they had to change few methods and fields that are already in use. Now the application users cannot use this library and an inconsistency arises.

The library developers at In-House department must provide a solution to help avoid such situations. Library developers must consider the following:

* Changes should be noticed, recorded and published properly.
* Redundancy of code must be avoided while creating new library APIs.
* A simple and an efficient way of recording these changes must be employed.
* In a distributed system some coordination must be maintained once the testing of the changed code has been completed so that the changes can be implemented into the new library and API can be released and published.

If we apply our JBOOM model to the above mentioned scenario. Firstly, our model is an object oriented SCM model and any distinguishing entity in the organization can be simulated as an object and further such related objects can be grouped into classes. Extending the existing classes can create more classes. Our SCM model has at the top of its inheritance hierarchy a class that reflects the most common changes that evolve during the development of a software library. Then there are two subclasses of this class, which again reflects changes that have been made in the members of a class and the classes in the inheritance chain hierarchy, respectively.

Corresponding Author: Bhavya Mehta, Department of Computer Science, College of Vocational Studies, University of Delhi, India

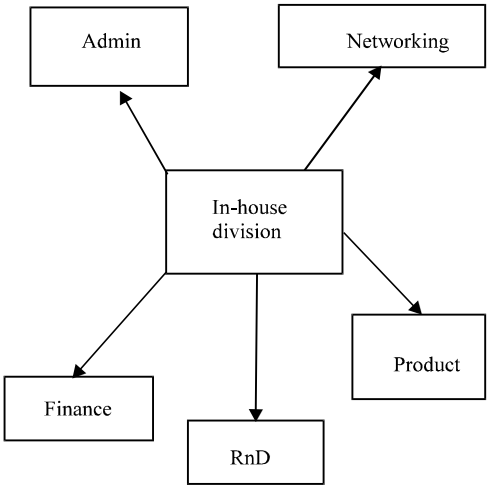


Fig. 1: Company organization

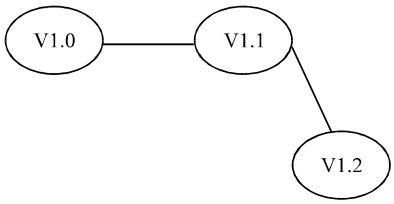


Fig. 2: Version tree for the library

The problem stated above can be efficiently handled by applying this model and its various classes we can extend these classes to an extent we want even the functionality of the library can be successfully altered and later recorded by using our SCM model[3]. As stated earlier in this section in the ver1.2 of the developed library, changes are made to the methods and members.

The following set of changes may occur giving rise to inconsistencies in the system:

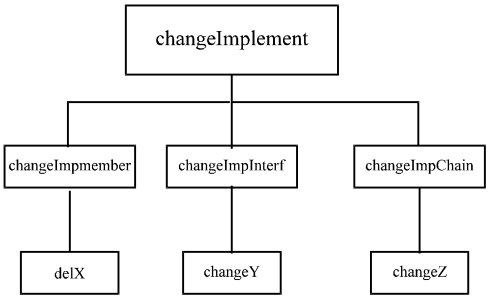
C1: Any public classes if deleted/added may be referenced by other classes in the hierarchy.
C2: Any public data members have been altered (deleted or added)
C3: Public methods have been changed/deleted or added
C4: Final methods or members are changed.
C5: Any abstract classes and methods are deleted or added
C6: Any interface or its methods and data members have been added or deleted.

We propose a model that takes care of all the above-mentioned changes in a systematic manner.

Object oriented modeling using java: A model essentially is responsible for generating potential configurations of systems. Whenever a change occurs in a real world entity it's a two-part process firstly its state changes and the other part involves change in its implementation. A model provides an abstraction at some level; it reflects only the essential required features of a system in problem domain and avoids others. Since a model must highlight on "what" as well as "how" of system under development, an object oriented model based on Java reflects both, what a system does and how it accomplishes that. What refers to specifications and how refers to the implementation aspects of a system. An ideal model is one that separates both these aspects. Abstracting away from implementation is an important feature of modeling. It also allows a chain of several specification-implementation relationships, in which each implementation defines specifications for the next layer. Using Java as the basis for the creation of an SCM model provides all the above-mentioned features in addition to providing the design and the language constructs. Another feature of Java is generalization, which is a taxonomic relationship between a more general description and a more specific description that builds on it and extends it. More specific description is fully consistent with more general one and may provide extension to the functionality of generalized one. A sub class in Java is said to extend a class the word extends in itself signifies the fact that an extension to the existing class is provided. In Java if the need is to hide certain details from the end user interfaces are used. This way we can have client objects access services providing objects indirectly through an interface. A key property of Java is that objects are manipulated indirectly, through implicit references to explicitly allocated storage. The JVM implementation performs automatic garbage collection, as a background thread. Normally interfaces provide an indirection to the inheritance hierarchy. This indirection allows the clients to access the service-providing object without having to know what kind of objects they are. But if we need to design a set of related classes that provide similar functionality then organize common portions of their implementation into an abstract super class. Interface is used in place of an abstract class when there is no default implementation to inherit i.e. no instance variables and no default method implementation. One benefit of using an interface is that a class can implement as many interfaces as it needs an object oriented model is the basis of object oriented techniques. At its core it means to model the system being designed as a collection of objects. An object is a distinct self-contained entity that is an abstraction of some aspect of the system being modeled .It has a variable that defines its state and methods that define its behavior. The variables are usually private to the object, which means that the object encapsulates its internal state making it a black box (an abstraction). Its methods provide public access to this encapsulated unit. Individual objects are instances of a class .A class is a prototype, a blue print based on which objects are

created. Normally classes at the top of inheritance hierarchy are abstract super classes and interfaces are used at lower levels relatively.

JBOOM: Java based object-oriented model of SCM: For every change that is made in the code, our model has corresponding classes[4,5]. Java provides the programmer with a customized change representation in the form of classes. Our model provides a solution to the above-mentioned changes (C1-C6) in a systematic and efficient manner. Java provides an organized and a hierarchical way of simulating the above-mentioned changes in the form of class hierarchy. Class at the top of JBOOM is *changeImplement*, an abstract class that reflects the most common changes that evolve during the development of a software system. This is an abstract class and has been created so that can be easily extended (Fig. 3).



Where:
X: Fields, Methods InhFields and Constructor.
Y: Class, Chain, Abst , Final.
Z: InhIntData , InhIntMeth

Fig. 3: Class hierarchy of JBOOM

It contains members that are used by other classes down the SCM model class hierarchy. Sole purpose of this class is to provide an appropriate super class from which other classes may inherit interface and provide their own implementation. An important feature in Java is that related classes can be grouped together in entities called packages. Since every item in an SCM system has a history associated with it, complete information about items needs to be maintained. Our proposed model uses separate Java packages to store new and the old SCM items. Vector class contained in java.util package can be used to create a generic dynamic array known as a vector that can hold objects of any type and number.

Method, boolean contains (Object element),this method returns true if element is contained by the vector and else returns false. Another method, Object elementAt (int index), returns the element at the location specified by the index.

Immediate subclasses of *changeImplement* (Fig. 4) class are:

* Class changeImpMember
* Class changeImpChain
* Class changeImpInterf

Above-mentioned classes are abstract classes.

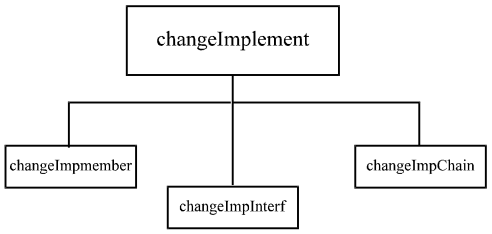


Fig. 4: Abstract classes of JBOOM

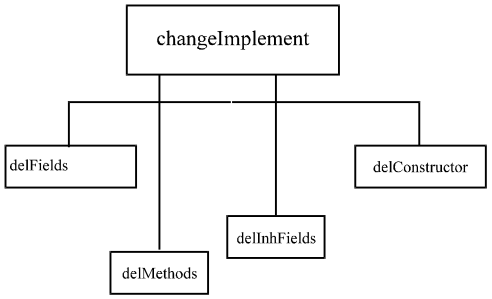


Fig. 5: The class hierarchy of changeImpMember class

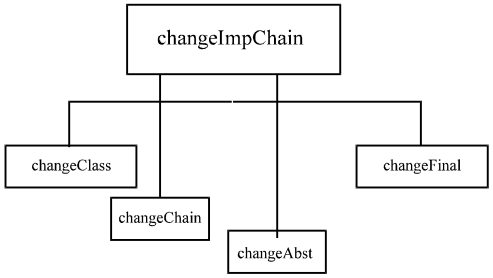


Fig. 6: Class hierarchy of changeImpchain class

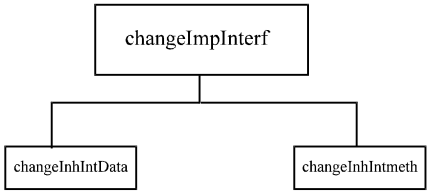


Fig. 7: Class hierarchy of changeImpInterf class

ChangeImpMember class contains method checkChangeImplement( )
That uses an abstract method Boolean implementChange( )

In changeImpChain class hierarchy implementChange( ) method is applied to every class. In chageImpMember class implementChange( ), returns

an integer for every member that has been changed since objects created are unique. These two classes have very little code so that further classes down the hierarchy can be created easily by extending the existing classes. Class changeImpInterf checks whether any changes have been implemented in interfaces used.

Derived classes of changeImpMember class (Fig. 5)
delFields class: Checks whether some field has been deleted from a class or not. If so, classes are checked where the changes have been made as a change in class's field's further affects rest of the classes in the hierarchy.

delMethods class: Checks whether any public methods have been deleted from a class under observation, if the members of a class are altered (changed /deleted /added).then it affects rest of the classes in the hierarchy.

delConstructors class: Checks for any constructors that have been removed.

delInhFields class: Checks whether the access privileages for any class have been altered.

Derived classes of changeImpChain class (Fig. 6)
changeClass class: Checks for any deleted public classes as these classes may have been used by other classes in the system.

changeChain class: Checks for any alteration in the inheritance chain hierarchy.

changeAbst class: Records changes that have been made to classes that have become abstract. Since the sole motive of declaring a class as abstract is to use it for extending other classes not for instantiating.

changeFinal class: Records changes made to those classes that have become final.

Derived classes of changeImpInterf class (Fig. 7)
changeInhIntdata: Checks whether any data conained in the interfaces have been changed/deleted or added, since interfaces facilitate multiple inheritance, so the classes that implements that interface needs to be checked .

changeInhIntmeth: Checks whether any methods contained in the interfaces have been altered.

Thus these above mentioned classes describe the structure of an object oriented Java based SCM model.

CONCLUSION

This study proposes an object-oriented model of an SCM system, JBOOM that simulates changes in terms of classes and their respective objects. Since the environment used is Java it eliminates the need of separately implementing these changes. In the proposed system design and implementation are integrated. Also an ideal way of representing evolving changes is a hierarchy. Our model is a hierarchical model, presents changes in a step-by-step manner[6-8]. Java is the de facto language for programming in the web and our model captures its features and arranges them in a hierarchy. The proposed model adopts a simple yet systematic approach towards managing evolving changes. The very fact that the model uses classes, abstract at the top of the hierarchy proves that it is simple to implement and extend which is a major requirement with any SCM system.

REFERENCES

1. Booch, G., J. Rum Baugh and I. Jacobson. The Unified Modeling Language. Addison -Wesley.
2. Cheon, Y. and G. Leavens, 2002. A Runtime Assertion Checker for the Java Modeling Language (JML).Software Engineering Research and Practice (SERP'02), CSREA Press, pp: 322-328.
3. Hamie, A., 2004. Translating the Object Constraint Language into the Java Modeling Language. SAC'04, Nicosia, Cyprus.
4. Korper, Elis and S.J. Ellis, 2000. The E-Commerce Book: Building the E-Empire, Orlando, FL: Academic Press.
5. Leon, A., 2000. A Guide to Software Configuration Management. Norwood, MA: Artech House.
6. Rational Software Corporation,, 2003. The Unified Modeling Language: UML Version 2.0, http://www.rational.com,'03
7. SEI., 2000. Capability Maturity Model. <http://www.sei.cmu.edu>.
8. Westfechtel, *et al*., 2001. A layered architecture for uniform version management. IEEE, TSE., 27: 12.