International Journal of Computing and Business Research (IJCBR)

ISSN (Online) : 2229-6166

Volume 2 Issue 1 2011

# MODEL FOR CONCERN DATA BASES

**Vishal Verma**

Kurukshetra University Post Graduate Regional Centre,

Jind, India

**Ashok Kumar**

Department of Computer Sc. & Applications,

Kurukshetra University, Kurukshetra, India

**ABSTRACT**: Large size complex software systems require the reuse of the code segment to a great extent to reduce the cost and time of development and to make the modification process as simple as possible. To make the reusability of code segment possible during engineering of complex software system the terms repository (secured collection) and mining [1] plays an important role. To implement the repository and mining of code segment we have to match our steps with the data base system for code segments. The data base of code segments must be implemented along with the development of the new system. This paper proposes a model which can help to do the repository, mining and hence to implement the reusability of code segments. This model explain the concept with the help of separation of concerns (code segments) used in Aspect Oriented Programming. The overall approach is based on the maintaining relationship between the concerns with relational data base. The approach can be used to describe the relationship among code segments in a better way than described by any other approach like concern graph [2].

**KEYWORDS:** Aspect, Concern, Data Base, Mining, Point Cut, Repository.

**INTODUCTION**: The repository, mining[1] and reusability of code segments are very important terms which should be implemented properly during engineering of new complex software system since reusability is important factor responsible for deciding the overall cost of development of the software system. Reusability is possible only if we do the proper repository of the code segments, & we have the proper method to do the mining of the desired code segment from the pool of available code segments, to do the mapping from one implementation(existing one) to another (desired one) and finally to reuse the code segment in the newly developing software. And hence the means/methods to support the overall reusability i.e repository, mining and reusability should be handled properly at the time of development of new software system.

The methodology of software development is advancing day by day from Procedure Oriented to Object Oriented to Aspect Oriented Programming [8].Now a days a lot of developments are going on with the Aspect Oriented Programming and hence we also use this method to be the base for new model. The basic terms used in Aspect Oriented Programming are: Aspect, Point Cut, Cross cut, Concern, Introduction etc. There are number of ways to implement the basic of AOP: AOP and AOD attempts to aid the programmer by the separation of concerns, specifically cross cutting concerns, as advancement in modularization. Separation of concerns entails breaking down a program into distinct parts that overlap in functionality as little as possible. AOP language like: AspectJ, HyperJ and Concern Manipulation Environment have their existence into the market. Important example which shows requirements of the cross cutting concerns among the software system are parallel processing of instructions, modules to find error, modules used for identification of user, enhancing the use of resources simultaneously among the number of user.

From the above examples it is clear that we cannot keep the code segments for the purpose in the simple single segments to be used only once, rather they are required to be available as and when needed with necessary modification (if required) at the time of execution and hence are kept in separate entity known as Aspect in AOP. It helps in increasing the control of local bodies on the code segments as per the requirements.

Some techniques like reification based models for storage and retrieval of aspects in object data bases [3] and explored composition of aspects with objects in such environment [4], some approaches like binary large object or character large objects exist which work in collaboration with metadata which is ultimately used for code mining and reusability. The basic problem with the above approach is that they are not implement able independently from the users point of view. Every time we have to be dependent on large amount of data to search a simple code module. Another problem related to these methods is the non independent modules which create constraints for us on searching a code segment of our requirement in the available mixed code segments of comparatively large size.

Main motive for the proposed model are:

1) To provide a relational data base model in which the developer can do entries along with development of code segments with and id and special supporting information which helps users at the time of mining and reusability. Each entry is available with information for its use and the purpose in the existing system.

2) To provide a method (interface) which supports the queries in simple GUI based environment.

3) To help the user to update the existing code segments with some easy steps if possible.

The basic anatomy of the model is based on the relationship of code segments into the tables of relational data base, which can be updated time to time.

**A TOUCH TO AOP**: Aspect Oriented Software Programming is about the separation of concern in software system and the encapsulation of those separated concerns in modules [4].From the above discussion and examples in AOP it becomes clear that in AOP we try to deal with some code segments which are scattered among the software system, making them difficult to understand and maintain.

Best example of concern is the code segments used for implementing security through logging process, the logging process is implemented (in OOP) with the help of classes and their member functions and hence any kind of modification require the change in all appearance of the logging code segments i.e all the tangled code segments are get effected because of single modification [7]. In AOP the classes are designed and coded separately from aspects encapsulating the cross cutting code.

The links between classes and aspects are expressed by explicit or implicit join points. An aspect weaver is used to merge the classes and the aspect with respect to the join points. [6] One important outcome of introduction of AOP is the use of term concerns for code segments. Some users have a bit wider meaning of the term concern. Formally the term concern can be described as: “The part of a software system relevant to particular concept, goal or purpose [4]” or “A concern is anything a stake holder may want to consider as a conceptual unit including features, non-functional requirements and design idioms.”[5] Consider the embodiment of a concern to be the collection of all types and type members relevant to a particular concept, goal or purpose addressed by the software, coupled with any input or output data sets (or parts thereof) relevant to that same concept.[4] In many cases , the source code implementing a concern is not encapsulated in a single programming language module and is instead scattered and tangled throughout the system [5]. A variety of concerns were separated using AspectJ, including both homogeneous crosscutting concerns and heterogeneous concerns. By homogenous cross cutting concerns we mean a concern in which same or very similar behavior need to occur at multiple points in the control flow of a software system. By heterogeneous cross cutting concerns, we mean a concern that impacts multiple points in a software system but where the behavior that needs to occur at each of those points is different.[4] Scattered concerns make it difficult for programmer/developer to reason about which part/pieces of code interact to implement a concern and about how different concern interact with each other.[5] Scattering and tangling can be controlled up to a great extent by implementing the inheritance among the code segments or rearranging the code so that repeated code can be included in the base class and is retrieved/referred when needed in derived classes, but the ultimate result is the shift in angle from which we see the problem. The increase in overhead on the classes and methods of invocation gets complex and hence the overall reusability get implemented with some cost and no better benefits.

The basic anatomy of interrelationship of code segments and their implementation within aspect of AOP can be shown with the help of an figure as shown below (using UML notation).

```
aspect  <name of aspect>

{

        Point cut P1

         before(argument list)

        {

          // Some code

        }

        after(argument list)

        {

          //some code

        }

        introduction(argument list)

        {

        //some code
        }

}
```

Fig. 1 Anatomy of Aspect

The classes their member function and data members are designed and coded separately than the code which is responsible for crosscutting them from the rest of the part. The join points are used to make/implement the links between the aspect (collection of cross cutting code and method of implementation). The combination of class's code and aspect is combined together by using the Aspect weaver so that they can be get execute as per the requirement at the join points.

**TO ESTABLISH RELATIONSHIP BETWEEN CODE SEGMENT AND DATA BASE**: To implement the relationship between the code segments and their storage in data base we have to be clear about the basic element's structure of AOP

*Aspect*: An aspect is a modular unit designed to implement a concern. An aspect definition may contain some code (advice) and the instruction on where, when and how to invoke it. Depending on the aspect language, aspects can be constructed, hierarchically and the language may provide separate mechanism for defining an aspect and specifying its interaction with an underlying system.

*Advice*: Advice is a behavior to execute at a join point. Many aspect languages provide mechanisms to run advice before, after, instead of, around join points of interest. An advice is oblivious in that there is no explicit notation at join point that the advice is to be run here - the programmer of original base code may be oblivious to the evolving requirements.

*Join points*: Join points are well defined places in the structure or execution flow of a program where additional behavior can be attached. The most common element of a join point model are method calls, though aspect language also defined join points for a variety of other circumstances, including field definition, access and modification, exception and execution events and states.

*Point cut designator:* A point cut designator describe a set of join points. This is an important feature of AOP because it provides a quantification mechanism- way to talk about doing something at many places in a program with a single statement.

*Concern:* Any engineering process has many things about which it cares. These range from high level requirement to low level implementation issues, some are localized to a particular place in the emerging system and some refer to measure able property at a whole some are aesthetic and some involve systematic behavior.[7] From the above discussion and basic definitions of Aspect and its related term it is very much clear that an entity named advice in Aspect is totally independent from its other related parts. It is basically code segment which is designed to be executed at the "before" or "after" conditions of a part of code segment. The advice entity may be co related with named point cuts since they give a clear cut point of insertion/execution for the advices. The implementation of new feature/operation is done with the help of introduction of "Introduction" of the new code segment into the existing code segment.

**DATA BASE MODEL**: The proposed model for explaining the relationship between actual code and its related data base can be designed by using the fig .1 as its base for the coding. From fig.1 we can easily observe that the element Aspect contain some basic entities as point cuts ,before, after, introduction etc All of these entries though represented collectively but still are independent from each other. All of these elements are used to take a decision regarding the point of applicability of the Aspect in the application, before and after give the information about the applicability of condition where the code related with the Aspect will execute when called. Graphically the aspect's part can be shown with the help of fig.2. If we consider the fig.1 as the smallest unit to be used as the code segment then the anatomy of the related data base can be represented by tables from table – I to table -IV

Program

Aspect

International Journal of Computing and Business Research (IJCBR)

ISSN (Online) : 2229-6166

Volume 2 Issue 1 2011

| Advice | Point Cut | Introduction |
|---|---|---|
| Argument | Argument | Argument |
| Condition | before | code |
| Code | after | return type |
| Return type | some code | |

Fig 2. Graphical representation of aspect

Program Table Table-I

| Program Id | Aspect Id (I) | Aspect Id(n) |
|---|---|---|
| 1 | As1 | Asn |
| 2 | As2 | asn |

Aspect Table Table-II

| Aspect Id | No of Arg. | Some Code | Return Type | Description |
|---|---|---|---|---|
| As1 | 2 | // | Int | // |
| As2 | 4 | // | Float | // |
| | | | | |

Point Cut Table Table-III

| Point Cut Id | Aspect Id | Type | Corelation |
|---|---|---|---|
| P1 | As1 | Before | // |
| P2 | As2 | After | // |

Introduction Table Table-IV

| Introduction Table Id | Aspect Id | Code | Discription |
|---|---|---|---|
| I1 | As1 | // | // |
| | | | |

The above discussed combination of tables for implementation of the data base model for saving the code segment (with reference to AOP) represent exactly how the various items of the aspect/concern can be stored independently in the form of tables and can be collected together to make a data base. The important point to be noted form the entries in the tables i.e from Table-I to Table-IV is that all the entries in the upper tables (from top to bottom) are independent on the entries in the lower tables which ultimately provides the immunity to the user to make any number of entries in the lower tables as per the requirement. Each and every table as well as all the elements can be assigned a unique id which help to do the hurdle free entries in the data base. The concept of assigning id to the data elements in the table helps in mining of data from the data base as simple as possible. The little description about each element helps the user to select the code segments which fulfill his requirement. Once the user has the data base structure of the aspects/concerns of the program he/she can develop the front end application for easy access of the entries from the data base.

**APPLICABILITY OF MODEL**: This model can be used to make the data base for Aspect's of any Aspect Oriented Programming based software. In the step by step by manner the programmer with the help of design of the software system decide the number of Aspects/ Concerns required to be stored in the data base along with the little description of each. As the coding and relevant information becomes available with the phases of software development the data base design is finalized and the entries are made into the data base. The method of maintaining the uniqueness of the tables as well as various data elements is decided at the time of data base design. Number of tables in the data base will depend on the number of Aspects and their interrelationship among each other.

**FUTURE ENHANCEMENT IN THE MODEL**: Model discussed in this paper suggests the way of saving the code segments of a program with reference to AOP in the data base. This model also allows the user to perform all the basic operations like addition of new record, deletion of existing record and searching of a record from the data base. The proposed model can be enhanced so that it may support the entries in the tables which show the relationship among entries and how they are used in various programs simultaneously. The another modification can be done in it so that this model may be able to support the basic query structure of the sql or of any other query language which can help the user to retrieve the record from the data base. The implementation of a graphical user interface for the writing and execution of queries can be done. The graphical interface helps the novice programmer to work with it and will be useful for the training of the new user of the software.

## REFERENCES

1. Loughran, N., A. Rashid (2002) *Mining Aspects*. Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design (with AOSD 2002)

2. Rashid. A On to Aspect Persistence. 2nd International Symposium on Generative and Component based Software Engineering (GCSE), 2000: Springer-Verlag, Lecture Notes in Computer Science 2177: 26-36
3. Rashid. A, Weaving Aspects in a Persistent Environment. ACM SIGPLAN Notices.Feb 2002
4. Colyer, A., A. Rashid, G. Blair (2004) *On the Separation of Concerns in Program Families*. Technical Report Number: COMP-001-2004(1)
5. Matin P. Robillard, Gail C. Murphy, Representing Concerns in Source Code, ACM Transactions Software Engineering and Methodology, Vol 16. No. 1 Article 3. Publication Feb. 2007
6. Rashid, A. and P. Sawyer (2000) *Object Database Evolution using Separation of Concerns*. ACM SIGMOD Record. Volume 29(4), Pages 26-33.
7. Referenced from wikipedia free website for education material.
8. I. Krechetov, B. Tekinerdogan, A. Garcia, C. Chavez, U. Kulesza (2006) *Towards an Integrated Aspect-Oriented Modeling Approach for Software Architecture Design*. Workshop on Aspect-Oriented Modeling held AOSD'06, March 20-24, 2006, Bonn, Germany