

Test of a Data Basis Oriented Object after Phase of Conception

Soumia Layachi and MT Laskri

Laboratory of Research in LRI Data Processing, University Badji Mokhtar Annaba

Abstract: The concepts of the approach object introduce cause a lot of interest and make a large topic of survey of it. These concepts applied to the data base are going to allow us to master the complexity of the data base and facilitate us their reuse. If one considers the conception of a data basis oriented object a lot of methods of conception make the object of survey; for the stage of test no strategy of test has been proposed however, except some works that propose specific tests in the used domains and it is especially applied at the time of the phase of validation of the data basis. In this study a strategy of test of the applications oriented object is adapted to the data base oriented object.

Key words: BDD, BDDOO, AOO, Test, strategy of test

INTRODUCTION

The modelling aims to réduire la complexité as isolating a petit numbers elements importing at the same time to examiner. A model constructs itself therefore from an abstraction of the studied problem. The abstraction is a human faculty that permits to separate what is important and of what is not. To a reality, there is an infinity of modèles. A good model is the one that describes a reality for a problem posé^[1].

The recent research at the level of SGBDs has the tendency to demonstrate that the most promising solution to remedy the hiatuses of the present SGBDs (basis of data for the domains industrial and big systems) consists in adding persistence to a programming language.

While insuring that this system possesses an important modeling power and offers all other functionalities of SGBDs.

SGBDOOs based on programming languages such as (Small Talk, C++, ...etc) correspond well to this reality, whereas traditional SGBDs and the programming languages constitute two distinct realities (forever compatible) SGBDs are the result of the integration of the classical functionalities of SGBDs and an Object Oriented Language^[2].

If we consider the conception of an object oriented database many methods of conception will be the object of surveys in studies^[3-10]; for the stage of test no strategy of test has been proposed however, except some works that propose specific tests in the used domains^[11-18] and it's especially applied in the phase of validation of the database.

If we consider AOO, a strategy of test has been proposed and can be adapted to the object oriented database^[19,20].

Our project consists in the realization of a data basis multi médiat that will allow us to stock medical

pictures of the sound and the text. His/her/its data will be used later in entry for applications of treatments of pictures and the sound.

This basis of data must be reliable and of quality to be able to use given them without none problems.

One will use the UML method at the time of the phase of analysis and cocéption of the data basis. The graphs results of this concéption will be used at the time of the stage of test. As for the programming the whole work will be achieved in an environnement of Java programming.

It is in this context and with the aim of proposing a general test strategy for BDDOO that our work is located. This strategy tests the database right after its conception, something which permits to the inventor to correct her (his) mistakes early before putting it at the disposal of the user.

METHODOLOGY OF THE APPROACH

The proposed strategy is essentially based on two steps: a static analysis and a dynamic analysis.

Static analysis: In a first step we do a code static analysis of different classes of the basis; the sought goal being:

- * The assessment of a set of quality indicators.
- * The construction of the graphs of controls of the class methods.
- * The construction of the graphs of dependence between classes.
- * The analysis and the edition of the different graphs (class, method)
- * The interdependence between the different constituents of the software.
- * The addition, deletion and the modification of a class of the database.

Dynamic analysis: In a second stage, we conduct the analysis of the dynamic behavior of the classes of the database. The sought goal is:

- * The assessments of the efficiency of the tests.
- * Help to the follow up and the pursuit of the tests.
- * The assessment of the stopping criteria.

Test strategies of the object oriented applications:

The strategy recommended for the unit tests and integration of the classes of the object oriented applications presents the advantage of being general and remains applicable in most part of the programming languages by object. It's based on the two approaches introduced earlier. The static analysis permits to evaluate an important number of quality indicators^[21-23] specific to the object oriented applications, starting at the first phases of the cycle, thus permitting, on one hand, the development control at different levels and on the other hand, to orient the test process^[24]. The dynamic analysis permits on one hand, to refine a part of these indicators and on the other hand, to evaluate the efficiency of the test through the retained rates of cover^[19,20].

Proposed strategy: After the conception of the database, the inventor applies the test strategy of object oriented applications, to his object oriented database since we retrieve the same concepts packets of classes, classes, methods....etc. We can construct all the graphs necessary to the test of our database (graph of inheritance, graph of call between classes, graph of control of a method, graph of call between methods).

We intend to evaluate, starting right at the conception phase of the database, an indicator characterizing the estimable reliability of a class noted IFP^[19-22,24].

It is in fact about an evaluation of the rate of mistakes that if we can think that it is often linked to the failing rates, doesn't permit however, to evaluate to assess the operational reliability^[21,23].

The IFPs are used in the test strategy of databases to inform us about the classes of the most critical databases as well as the critical methods on which it is necessary to concentrate more the test. Thereafter, all along our work, they will help us to update the database (when adding, deleting and modifying)

Indeed, for each class of the graph of calls, we insure though these rates of cover, that the set of messages that is addressed to it as well as the messages that it sends to the other classes have been executed at least once (global communication of the class) (Fig. 1). The elimination of an arc (C_i, C_j) of the graph of calls, cannot be done, unless the set of messages between C_i and C_j have been executed at least once corresponds to the matching degree $C(C_i, C_j)$. We can hold this same reasoning for each class in the graph of calls. The node corresponding to a class C_i can't be deleted from the graph of calls with the set of arcs leaving from and

arriving to that node, unless the set of messages of the calling classes C_k and the set of messages addressed to the called classes C_j have been executed at least once (Fig. 1).

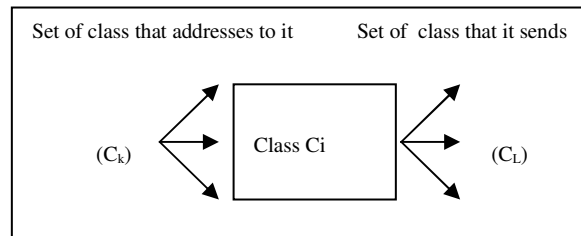


Fig. 1: Global communication of the class

The computed rate of test along the test will help the developer to decide if he got to a satisfactory rate, in other words, have a database well tested. The developer should be able to add, suppress or modify a class.

Addition of a class: We indicate the super class of the class to add, in other words, the code of the class to add. We don't accept the addition before the root class of the graph. We display the initial inheritance graph and the modified inheritance graph by marking the added class Fig. 2.

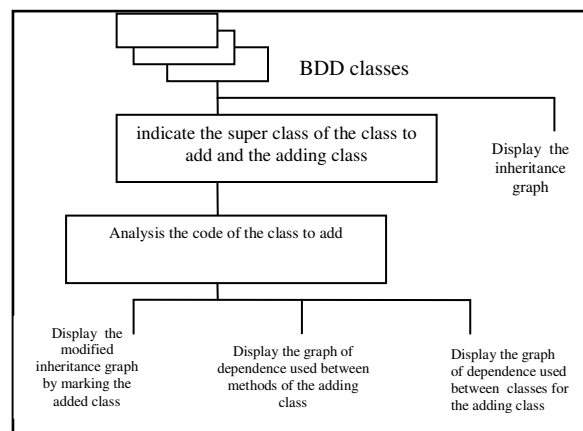


Fig. 2: General principle of the addition of a class

The analysis of the code of the methods of the class will allow us to know the communication of this class with the other classes (through these methods) of the BDD Fig. 3.

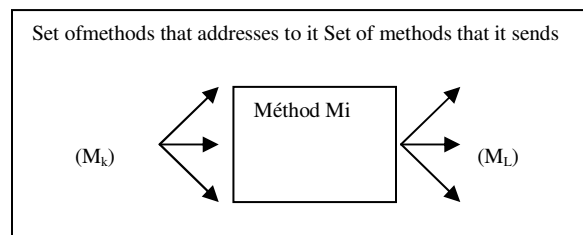


Fig. 3: Global communication of the class

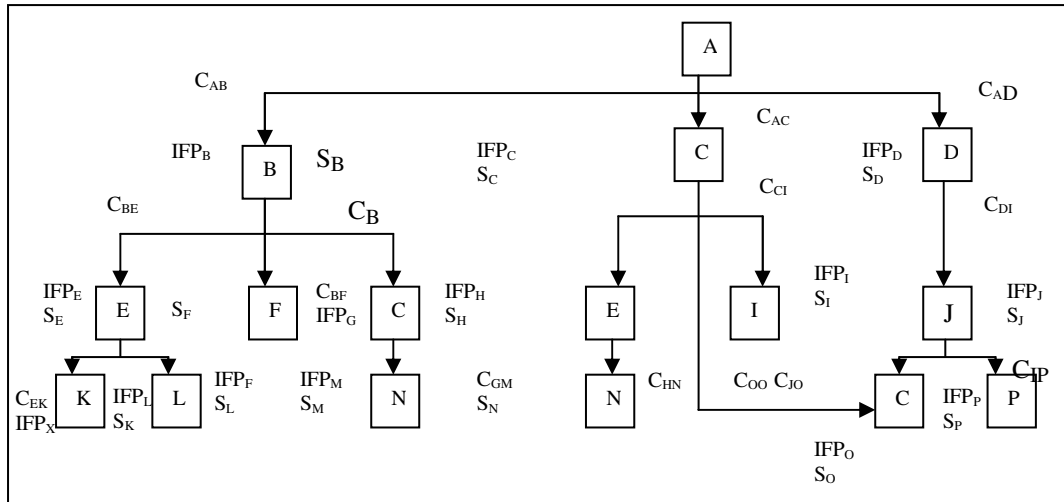


Fig. 4: Graph of dependence used between classes of the application

```

Display the inheritance graph and the graph of dependance used between classes of the application informed by the
informations (IFP,C,Si)
if (décision =tue) then
  repeat
  Select the class to delete
  Display the graph of dependance used between classes of the application with marquage of the suppress classes
  repeat
    display the set of methodes that addresses to it
    Up date of the classes(up date the method of the classes)
    Throw back the unitary test of the modifying methodes
    Throw back the integration test of the modifying classes
  Until the last method
  Reduction the call graph
  Until the last class
  Reduction the inheritance graphe
  Display the new inheritance graph
    
```

Fig. 5: Steps to follow for the suppression of a class

```

Display inheritance graph with marquage of the class where methode will be suppress
Display the set methods of classes that address to it (communication of the method)
Up date the classes (up date the methods of class)
Throw back the unitary test of modifying methods
Throw back the integration test of the modifying classes
    
```

Fig. 6: Steps used to suppress a method

We throw the unit test of the added methods as well as their integration test. Thereafter we throw the unit test of class and its integration test. Throughout this work we use the test that we have already used for the AOO, as well as the rates corresponding to each step of the test to be able to evaluate the IFPs of this added class.

Once this part is finished, we notice that in the methods of this added class we needed the other methods pertaining to the BDD class, but no class of the database needed the methods of the added class. This modification (add a call toward these methods) will be taken under account by the modification part.

Addition of a method: The developer indicates the class where he wants to add a method. Thereafter the method is added, we analyze the code of this method and we should display the method control graph and the graph of call between methods of the class and the graph is used since it's modified by the adding of that of that method. We throw the unitary test of the method and its integration test.

Deletion of classes: The developer indicates the class to suppress, we don't admit the suppression of the root class of the inheritance graph. We display the inheritance graph and the used graph informed of the information (IFP, Si, C)^[20] to indicate for it the importance of the class relatively to the set. The

developer can come back on his decision if he judges the class important otherwise he will have an idea on the modifications to bring thanks to the information on the machings indicated at the level of the used graph (C). Figure 4 gives an idea of that information:

If the developer indicates a class which has some son classes, all the class packet will be suppressed. We display the inheritance graph by marking the class or classes to suppress.

For each class to suppress we should consider its methods and indicate for each method the set of methods it calls and the set of methods calling it, like indicated in the Fig. 5.

For the methods which call the methods of the class to suppress, we should modify these methods and ignore their call toward these suppressed methods. We throw back the unitary test of these methods and their integration test. We should redo this same work for all the classes to suppress. During the suppression we should follow the following steps (Fig. 5)

Suppression of a method: If we want to suppress a method in a class we should first indicate the class to which the method belongs. Thereafter, we determine the communication of that class with the other classes (for that method). Then we should update the calls of the other methods toward that method. We should throw the unitary test of the updated methods and thereafter throw their integration test. Throw the unitary test of the class where we operated a modification, Fig 6.

Modification of a class: This modification consists on deleting, adding or modifying a method.

Modify a method: The developer brings the desired modification at the level of the method and then throws the unitary test of this one and then its integration test.

CONCLUSION

The strategy that we recommend for the unitary and integration tests for the classes of the object oriented databases has the advantage to be general. It is proposed to AOO and can be applied to BDDOO right after the conception phase.

It stands on two approaches: static and dynamic, it permits us to evaluate in the static analysis phase an important number of quality indicators specific to object oriented applications which will be adapted to BDDOO and the dynamic analysis permits us to refine these indicators, to follow their evolution during test and evaluate the efficiency of the test.

This strategy is the first part of a work on the test of BDDOO, the second part should take under account the dynamic test of the object oriented database to be able to verify the integrity constraints of the database.

REFERENCES

1. www.univ-reunion.fr/~panelli/enseignement/coo/td.html
2. www.syspotico.ca/mapoisson/sbgdoo/chap2.html.
3. www.Vision.
4. www.orsys.fr/cours/conception_orientee_objet.htm.
5. www.philippe.guezelon.free.fr/mcd/mcd.htm.
6. www.esz_metz.fr/metz/personnel/popineau/ML/UMLpresentationpdf.
7. www.ceation_site_internet.info/chap004.htm.
8. www.test.int-evry.fr:8088/biblio/livres/basesdonnees.php
9. www.chez.com/etranvouer/cours/S5-ProgMultimediat-1p.pdf
10. www.alrj.org/docs/langages/pooc.pdf
11. www.VerifySoft.com/fr_cmtjava.htm.
12. WWW.VerifySoft.com/fr-jcover.htm.
13. WWW.moteurprog.com/?url=article_Affiche.php&AD_article=68.
14. WWW.Papp.in2p3.fr/Evenement/Anssors/aussoicristal.pdf.
15. www.i3s.unice.fr/~mh/RR/2002/RR-02_01-F.MALLET.pdf
16. www.phpaie.net/archives/jan04/threads.html
17. www.microsoftemea.comea.com/desktopdeployment/download/BDDtechnicaltraining/FAQ%20section
18. www.news.webplanete.net/Optimiser_objet_messages.htm
19. Badri, M., L. Badri and S. Ferdenache, 1995. Towards quality control métrics for oriented objected systems analysis. Proc. of Tools Europe'95, Versailles, France, Prentice hall.
20. Badri, M., L. Badri and S. Layachi, 1995. Vers une stratégie de test unitaires et d'intégration des classes dans les applications orientées objet. Génie logiciel, no. 18.
21. Aubry, R., C. Lamy and Y. Martinez, 1984. Maitrise de la fiabilité d'un logiciel temps réel dès la phase de conception. Approche quantitatives en génie logiciel. Sophia-Antipolice, juin 1984.
22. Badri, M., R. Aubry and Y. Martinez, 1989. Indicateur de qualité d'un logiciel et suivi de son évolution en cours de test deuxième colloque annuel du club Fiabex. Sûreté de fonctionnement et sciences des systèmes Evry, édition EC2.
23. Laprie, J.C., 1988. Surrété de fonctionnement et tolérance aux fautes: Concepts de base, Rapport LAAS n° 88.287, Toulouse.
24. Badri, M., R. Aubry and Y. Martinez, 1989. Outils d'aide à l'orientation et au suivi des test d'integration. Deuxième journées internationales. Le génie logiciel et ses applications. Toulouse, France, Edition EC2.