IMPLEMENTING THE DATA ACCESS OBJECT PATTERN USING ASPECTJ

André Luiz de Oliveira, André Luis Andrade Menolli and Ricardo Gonçalves Coelho
Informatics Department, State University of the North of Paraná, Br 369 highway Km 54, Bandeirantes, Brazil

Keywords: Data Access Object pattern, aspect oriented implementation, metrics.

Abstract: Due to the constant need of access and information storage, there is a constant concern for implementing those functionalities in large part of the currently developed applications. The most of those applications use the Data Access Object pattern to implement those functionalities, once that this pattern makes possible the separation of the data access code of the application code. However your implementation exposes the data access object to others application objects, causing situations in that a business object access the data access object. With the objective of solving this problem, this paper proposes an aspect oriented implementation of this pattern, followed by a quantitative evaluation of both, object oriented (OO) and aspect oriented (AO), implementations of this pattern. This study used strong software engineering attributes such as, separation of interests, coupling and cohesion, as evaluation criteria.

1 INTRODUCTION

A lot of applications frequently need implementing some access mechanism and data persistence, once that the organizations often need access and storage its information in some kind of database. To access this information surely and efficiently some patterns were introduced by several authors, not only in the object-oriented software development (OOSD) but also in the aspect-oriented software development (AOSD).

In literature we can find several patterns that implement the data persistence, such as Persistence Data Collections (Massoni et. al. 2001), that provides a set of class and interfaces to separate data access code and business rules from user interface, promoting the modularity. Among others patterns with the same purpose is the Decoupling Patterns, that is a set of data access patterns proposed by Nock (2003), that describes strategies to uncouple data access components from others parts of the application, and the Data Access Object (Sun Microsystems 2002), that is one of the data access patterns more often used, due to fact of this pattern isolates the whole data access code of the application code facilitating migration processes for other storage systems.

There are several forms of implementing the data access functionalities of the application. As seen in Sun Microsystems (2002), it can be adopted a data access objects factory strategy, in situations in which the applications access more than one storage mechanism, or create a generic data access object, in order to determine a common access point to the data access functions of the application.

Both implementations previously mentioned are good solutions that isolate the data access code from others application functions, optimizing the legibility and the modularity of the developed application, facilitating future maintenance and extension processes of the application, providing the reuse of several components of the system.

It was verified in Sun Microsystems (2002), that Data Access Object pattern allows the business object to using the data source without having the knowledge of the specific details of its implementation, providing a transparent access to its functionalities, once that those details of the data source implementation are encapsulated within the data access object. That whole transparency causes the exposition of the data access object to others application objects, causing situations in which a business object access the data access object, harming the modularity, the legibility and the encapsulation of the application code, besides the need of adding an extra object layer between the data client and the data source so that its implementation benefits this pattern.

With the objective of solving this problem, this study proposes an alternative implementation of the

Data Access Object pattern using the aspect oriented programming with AspectJ language. This solution provides the encapsulation of the data access functions of the application developed, once that the business object and others application components not more need access the data access functions of the application, all the control of the calls to the data access and persistence operations is made by aspects. This way, the others application objects do not need anymore know the implementation details of the classes and methods responsible by the data access functions of the application, optimizing the application modularity and legibility.

In this study, we identified some situations where it is profitable to change the Data Access Object pattern implementation to an aspect oriented implementation of this pattern. Those situations occur when there is a constant need of isolating all the data access functions implementation of the application, in order to avoid the exposition of the data access object to others application objects. Your application provides a better separation of concerns of data access which is verified by the outcomes presented in the section 4.

The results of this study were obtained through a comparative study between object oriented (OO) and aspect oriented (AO) implementations of the Data Access Object pattern. The comparison between OO and AO implementations of traditional patterns is not so new. Some papers have already made this kind of comparison, like Hannamann and Kiczales (2002) and Garcia et. al. (2005). This paper presents an alternative implementation of the Data Access Object pattern, followed by a quantitative evaluation of both OO and AO implementations of this pattern. It is important to point out that this work does not propose an approach that substitutes the Data Access Object pattern, but certainly is an alternative approach for the implementation of this pattern, which can be applied to the applications environment previously mentioned, providing a better separation of concerns of this pattern.

In order to explicit the foundation of the proposed solution, the present work is organized in eight sections. The section 2 presents the organization of the developed study that includes the illustration of the OO and AO implementations of the Data Access Object pattern, the metrics used in the validation of this study and the evaluation procedures adopted. The section 3 exposes the outcomes obtained in the comparison of the OO and AO implementations of the Data Access Object pattern. The section 4 illustrates the constraints of this study. The section 5 approaches some related works to this study. And finally, the section 6 exposes the conclusions and the future research proposes.

2 ORGANIZATION OF THE STUDY

This section describes the configuration of that empirical study. This study approaches an alternative implementation of the Data Access Object pattern, using the aspect oriented programming, with the AspectJ language. In order to explicit the foundations of proposed solution, the section 2.1 presents the object oriented implementation of the Data Access Object pattern and later the aspect oriented implementation of this pattern. Already the section 2.2 introduces the metrics used in the evaluation process of the proposed solution. And finally, the section 2.3 describes the evaluation procedures applied in this study.

2.1 Data Access Object Pattern Implementation

As seen previously, the Data Access Object aims at avoiding direct dependence situations between the application code and the data access code (Sun Microsystems 2002). When analyzing this pattern, it was verified that it defines two roles for its participant classes that corresponds to the Business Logic and the Data Access. The Business Logic role is responsible by the whole control of the business logic of the application, that means, it treats the input data before of those data have been passed into the call of one data access function, besides manipulating the obtained data of a data access function, to summarize that role is a data client (Sun Microsystems 2002). The Data Access role represents the object that abstracted the underlying data access implementation (Sun Microsystems 2002) to the Business Logic role, enabling transparent access to the data source, which means, it provides all the access and storage data functionalities to the Business Logic role. The Figure 1 illustrates the oriented object implementation of this pattern applied in the present work. When analyzing this figure it is assumed that the Data Access role implementation is dispersed by all classes of the pattern. Since the Business Logic role is implemented by the BusinessObject and TransferObject classes.

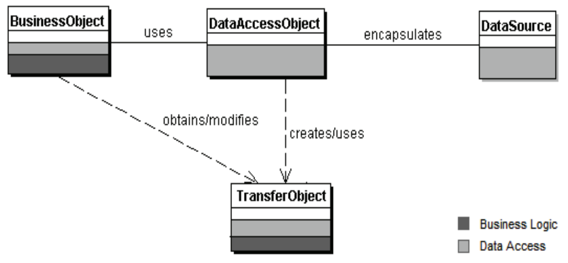
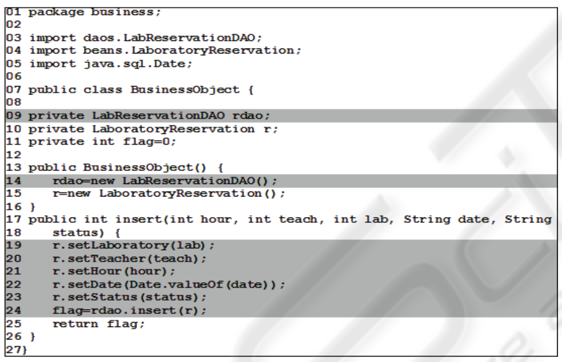


Figure 1: Object oriented model of the Data Access Object pattern.

The Figure 2 presents the source code of the BusinessObject class. The code spaces that do not implement the Business Logic role are shaded**.** This class possesses a reference to DataAccessObject and TransferObject classes (line 09-10), that are instantiated in the constructor method of the class (line 14- 15). This class contains the insert method (line 17-26), that modifies the attribute values of the LaboratoryReservation object (line 19-23), and passes this object as argument to the insert method, which is present in the LabReservationDAO (line 24) class, to register a new reservation.



```
01 package business;
02
03 import daos.LabReservationDAO;
04 import beans.LaboratoryReservation;
05 import java.sql.Date;
06
07 public class BusinessObject {
08
09 private LabReservationDAO rdao;
10 private LaboratoryReservation r;
11 private int flag=0;
12
13 public BusinessObject() {
14    rdao=new LabReservationDAO();
15    r=new LaboratoryReservation();
16 }
17 public int insert(int hour, int teach, int lab, String date, String
18    status) {
19    r.setLaboratory(lab);
20    r.setTeacher(teach);
21    r.setHour(hour);
22    r.setDate(Date.valueOf(date));
23    r.setStatus(status);
24    flag=rdao.insert(r);
25    return flag;
26 }
27}
```

Figure 2: Business Object class in the OO implementation.

As it is observed in Figure 2, the shadowed code implements the Data Access role within the Business Logic role implementation. Those shadowed points are places where there is a concern switch in the pattern implementation, once that the data access concern is interlaced with the business logic manipulation concern.

In order to solve this problem, the aspect oriented implementation of this pattern avoids that the business components, in this case the BusinessObject class, have the knowledge of the data access objects of the application, totally isolating those objects of the application code, providing a better separation of the Business Logic and Data Access roles, as illustrates the Figure 3.

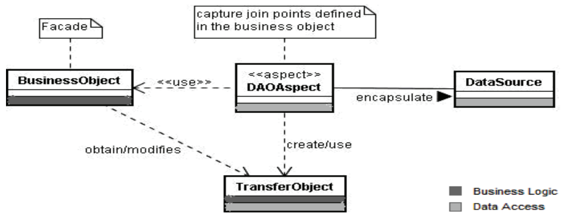


Figure 3: Aspect oriented model of the Data Access Object pattern.

In the diagram previously shown, the business object acts as a Facade (Gamma et. al. 1995) object that works as an access interface to functionalities present in the subsystem, in your conventional implementation. But in the AspectJ implementation, this object acts as one interface that contains only the method body (assign, parameters list and the return value) needed for the execution of the data access functionalities of the application.

The DAOAspect object, showed in the Figure 3, is the key point of the AO implementation of the Data Access Object pattern proposed in this work, due to fact this object makes the whole control of the data access logic of the application, through the pointcuts definition, that contains the join points that will be captured her for the execution of the data access functions of the application, implemented in the advices, as displayed in full detail the class diagram of Figure 4. This diagram is based on an aspect oriented modelling notation proposed by Jacobson and Ng (2003).

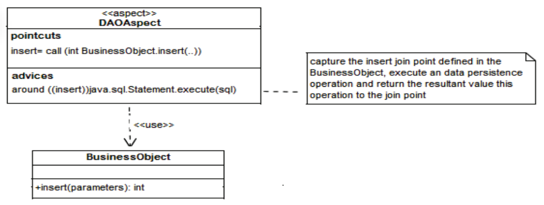

<<aspect>>
DAOAspect
pointcuts
insert= call (int BusinessObject.insert(..))
advices
around ((insert))java.sql.Statement.execute(sql)
capture the insert join point defined in the BusinessObject, execute an data persistence operation and return the resultant value this operation to the join point
<<use>>
BusinessObject
+insert(parameters): int

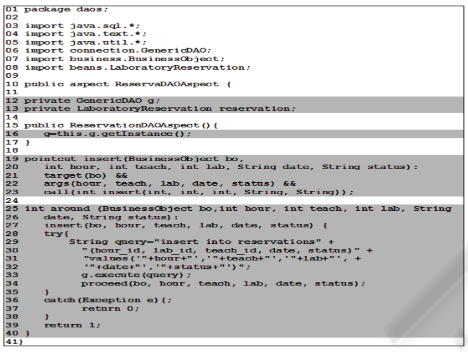
Figure 4: Class diagram of the aspect oriented implementation of the Data Access Object pattern.

Due to that solution, it can be observed that the aspect makes the whole control of the data access logic of the application, capturing a join point, executing a data access operation and returning the operation value to this join point.

The AO implementation of the Data Access Object pattern approached in this work avoids that the business object instantiating a data access object, in order to have access to its functionalities, once that the aspect captures the join points present in the business object, executes the required operation and returns the resultant value of its execution to this join point.

The presented solution provides the encapsulation of all data access functionality of the application, contributing to the modularity and legibility of the development application, once that in this solution the business object does not need to interact with the data access object that is implemented as one aspect that controls all the data access functions defined in the business object. The optimization of those software engineering attributes was verified through the application of a measurement process of this solution which is described in the next sections of this work.

In the AO solution of the Data Access Object pattern, the code that implement the Data Access role is textually localized in the aspect DAOAspect, that encapsulates all the data access functions of the application, isolating I from the remaining the application code, as illustrates Figure 5.

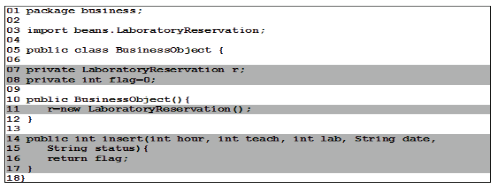


```
01 package daos;
02
03 import java.sql.*;
04 import java.text.*;
05 import java.util.*;
06 import connection.GenericDAO;
07 import business.BusinessObject;
08 import beans.LaboratoryReservation;
09
10 public aspect ReservaDAOAspect {
11
12 private GenericDAO g;
13 private LaboratoryReservation reservation;
14
15 public ReservationDAOAspect(){
16    g=this.g.getInstance();
17 }
18
19 pointcut insert(BusinessObject bo,
20    int hour, int teach, int lab, String date, String status):
21    target(bo) &&
22    args(hour, teach, lab, date, status) &&
23    call(int insert(int, int, int, String, String));
24
25 int around (BusinessObject bo,int hour, int teach, int lab, String
26    date, String status):
27    insert(bo, hour, teach, lab, date, status) {
28    try{
29        String query="insert into reservations" +
30          "(hour_id, lab_id, teach_id, date, status)" +
31          "values('"+hour+"','"+teach+"','"+lab+"', +
32          '"+date+"','"+status+"')";
33          g.execute(query);
34          proceed(bo, hour, teach, lab, date, status);
35    }
36    catch(Exception e){;
37          return 0;
38    }
39    return 1;
40 }
41}
```

Figure 5: DaoAspect implementation.

The related code to the Data Access role implementation is shaded and contains a reference to the GenericDAO object (lines 12 and 16), that corresponds to the data source implementation and a reference to the LaboratoryReservation object (line 13), which is the object that implements part of the business rules of the application.

The whole implementation process of the data access concern ponders in the insert pointcut (line 19-23), that captures the insert join point, defined in the BusinessObject class, and in the around advice (line 25-40), that executes a database manipulation operation around the call of a join point, returning the resultant value of that operation to the same joint point. That implementation removes the whole data access concern from the class that implements the business rules of the application due to the fact those classes do not depend on the data access classes of the application anymore. This independence can be better visualized in Figure 6 that illustrates the BusinessObject class implementation after the insertion of the data access aspect DAOAspect.



```
package business;

import beans.LaboratoryReservation;

public class BusinessObject {

private LaboratoryReservation r;
private int flag=0;

public BusinessObject(){
   r=new LaboratoryReservation();
}

public int insert(int hour, int teach, int lab, String date,
   String status){
   return flag;
}
}
```

Figure 6: BusinessObject class after of the DaoAspect implementation.

As displays Figure 6, in the AO implementation of the Data Access Object pattern, the whole data access concern code was removed from the classes that carry out the Business Logic role, making them more modulated and maintained, once that in those classes it is pondered only the Business Logic concern, that corresponds to shaded code of the Figure 6 and there is not dependence on calls in relation to the data access functions of the application, as illustrated in the conventional object oriented implementation in the Figure 2. This is possible due to the whole call control and execution of the data access functions of the application is pondered in a data access aspect.

2.2 Metrics Used

This study selected a group of metrics of separation concerns, coupling, cohesion and size (Sant'Anna et. al. 2003) to evaluate the foundations of the aspect oriented implementation of the Data Access Object pattern approached in this work. Those metrics have already been used in four different studies (Garcia 2004; Garcia et al. 2005; Garcia et. al. 2004; Soares 2004). Some of them have been automated in the context of a query based tool for the measurement and analysis of aspect oriented programs (Alencar et. al. 2004). Those metrics were defined based on the reuse and refinement of some classical and object oriented metrics (Chidamber and Kemerer 1994). The original definitions of object oriented metrics (Chidamber and Kemerer 1994) were extended to be applied in a paradigm independent way, supporting the generation of comparable results.

The separation of concern metrics measures the degree to which a single concern in the system is mapped for the design components (classes and aspects), operations (methods and advices), and lines of code (Sant'Anna et. al. 2003). The Table 1 presents a brief definition of each metric applied in this study, and associates them with the attributes measured by each one. More detailed information about those metrics can be found in Garcia (2004) and Sant'Anna et. (2003).

Table 1: Metrics.

Attributes	Metrics	Definitions
Separation of Concerns	Concern Diffusion over Components (CDC)	Counts the number of classes and aspects whose main purpose is to contribute to the implementation of a concern and the number of other classes and aspects that access them.
	Concern Diffusion over Operations (CDO)	Counts the number of methods and advices whose main purpose is to contribute to the implementation of a concern and the number of other methods and advices that access them.
	Concern Diffusions over LOC (CDLOC)	Counts the number of transition points for each concern through the lines of code. Transition points are points in the code where there is a "concern switch".
Coupling	Coupling Between Components (CBC)	Counts the number of other classes and aspects to which a class or an aspect is coupled.
	Depth Inheritance Tree (DIT)	Counts how far down in the inheritance hierarchy a class or aspect is declared.
Cohesion	Lack of Cohesion in Operations (LCOO)	Measures the lack of cohesion of a class or an aspect in terms of the amount of method and advice pairs that do not access the same instance variable.
Size	Lines of Code (LOC)	Counts the lines of code.
	Number of Attributes(NOA)	Counts the number of attributes of each class or aspect.
	Weighted Operations per Component (WOC)	Counts the number of methods and advices of each class or aspect and the number of its parameters.

2.3 Assessment Procedures

With the purpose of comparing the OO and AO implementations of the Data Access Object pattern, both versions of the pattern implement the same functionalities, with the same style of codification. This pattern has several implementation forms, in which this study chose the one that seemed to be the most suitable. A few modifications happened in the aspect oriented solution in relation to the object oriented solution, in which some methods were modified or excluded, some attributes and a class were excluded and an aspect was added. Likewise to the HK study, this work treats each role as a crosscutting concern, because the roles are primary sources of crosscutting structures (Garcia et. al. 2005).

In this measurement process, the data were gathered with base on the code analysis, using the Eclipse 3.3 tool (Eclipse Project). The measures of separation of concerns (CDC, CDO and CDLOC) were preceded by the shade of all classes and aspects in both pattern implementations. This shade was accomplished with the roles found in the Data Access Object pattern implementation. Likewise to the Hannemann and Kiczales (2002) study, in this study each pattern role was treated as a concern, once roles are primary source of crosscutting structures [8]. The Figures 2, 5 and 6 exemplify the shade of some classes and aspects in both the Java and AspectJ implementations of this pattern, considering the Business Logic and Data Access roles. After the shade, the separation of concerns data metrics was manually collected.

3 RESULTS

This section presents the measurement process results. The data have been collected with base on the metrics defined in section 2.3. The objective of that measurement process is to describe the results of the metrics application in the OO and AO implementations of the Data Access Object pattern, in order to compare and prove the foundations of the aspect oriented implantation of this pattern, proposed in this work. This analysis is divided in two parts. The section 3.1 focus on the analysis of what extent aspect oriented and object oriented solutions provide support to the separation of the pattern-related concerns. The section 3.2 presents the results toward the coupling, cohesion and size metrics.

In the exhibition of the results of this study graphics are used to represent the data gathered in the measurement process. The results of the graphs present the data obtained from the OO and AO implementations of the Data Access Object pattern. The Y-axis of the graphic presents the absolute values gathered by the metrics. Each bar pair, that corresponds to the metric values gathered from the OO and AO implementations of the pattern, is attached to a percentage value, which represents the difference between the OO and AO results. A positive percentage means that the AO implementation was superior, while the negative percentage means that the AO implementation was inferior. Th0se graphics support the analysis toward how the introduction of new classes and aspects affect both solutions in relation to the selected metrics. The results presented in the graphics were

gathered based on the pattern point view that means that they represent the metric values associated with all classes and aspects for each pattern implementation.
In order to obtain the separation concerns metric values of both OO and AO implementations of the Data Access Object pattern, firstly it was verified the presence of two roles in this pattern implementation. This was made because the roles are primary sources of crosscutting structures (Garcia et. al, 2005).

3.1 Separation of Concerns

The Figure 7 presents the separation of concerns metrics results for the OO and AO implementations of the Data Access Object pattern. As it is observed in Figure 8, most of the measurements favoured significantly the AO implementation of this pattern. This solution reduced the coupling between the classes that play the Business Logic role, and consequently the number of operations for implementing this role, besides demanding few concerns switches between the components that play the Business Logic and Data Access roles.

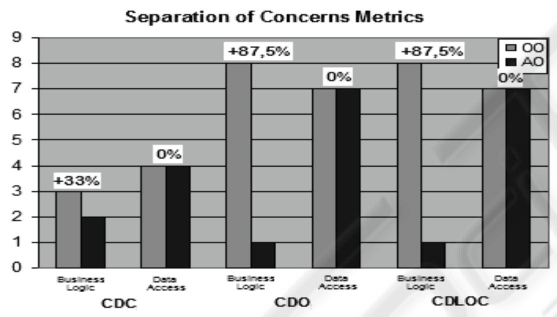


Figure 7: Results of separation of concerns metrics.

An analysis of the Figure 7 shows that those improvements were reached through separating the pattern roles in aspects. The definition of the Business Logic role demanded 3 classes in the OO solution, while the aspects application reduced that number to 2, providing a better separation of that role in relation to the OO solution. That improvement is equal to the 33% of AO solution superiority with relation to the OO implementation. The results were even better for the concern diffusion over components (CDC) and diffusion of concerns over lines of code (CDLOC) metrics in the implementation of the Business Logic role, which reached optimizations of 87,5% in relation to OO implementation.

In addition, it can be observed that good results were reached in the modularization of the defining role, which is the case of the Business Logic. After this analysis, is ended that the AO implementation of the Data Access Object pattern optimized about 69% the isolation of the implemented concerns by the Business Logic role in comparison with the OO solution.
One of the reasons for the superiority of the AO implementation over the OO implementation is that in the OO solutions there are several operations implementations mixed with the specific code of the role.

3.2 Coupling, Cohesion and Size

In this section are presented the results of coupling, cohesion and size metrics. It was used graphs for the representation of the gathered results, which represents the metric values associates with all classes and aspects for each pattern implementation, with the exception of the DIT metric. The results of DIT represent the maximum value of this metric for all the implementation.

In the OO implementation of the Data Access Object pattern the improvements were reached only in the NOA metric, as illustrates the Figure 8. The use of aspects led the reduction of only 0,5% of the LOC metric in relation to the object oriented project, therefore is assumed that this difference is insignificant. In spite of the data access code retreat of the classes that carry out the Business Logic role, the aspect that carry out the Data Access role has more code lines than the OO implementation of that role.

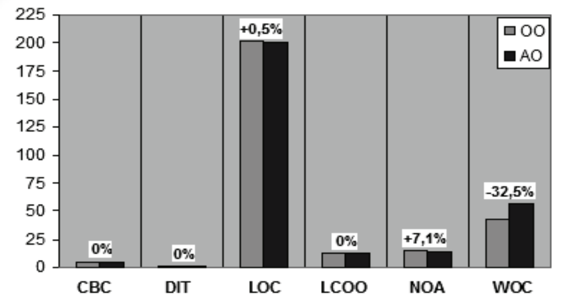


Figure 8: Coupling, cohesion and size metrics results.

In relation to the NOA metric results, the AO implementation obtained 7,1% of superiority in relation to the OO solution. This occurs in virtue of in the OO solution, the classes that carry out the Business Logic role possesses a reference to the data access object that in the AO project was retired.

In relation to the CBC and LCOO metrics, the results were same to both OO and AO implementations. This similarity happens due to the occurrence of just one a coupling inversion between the business object and data access object components in both implementations of the pattern. In the OO implementation, the business object component has a dependence relationship with the data access object. On the other hand in the AO implementation, the data access object component possesses a dependence relationship with the business object component, as shown in the Figure 3. In relation to the cohesion metrics, its values continued constant in both solutions, once that they use the same operations and references to the attributes.

The value of the WOC metric was the only in which the AO implementation obtained result inferior to the solution OO. This inferiority corresponds to 32,5%. The reason of this is based on the fact that the AO implementation has declarations of advices with many parameters, and this does not happen in the OO solution.

4 STUDY CONSTRAINTS

In relation to the metrics used in this study, it was verified in Garcia et. al. (2005) that there are some criticisms about its application. Those critics refer to the theoretical arguments related to the use of conventional size metrics that are commonly applied to the traditional software development. In spite of the simplicity of those metrics, they have been the target of several criticisms (Zuze 1995). The reason of those criticisms is the difficulty of evaluating software quality attributes, for instance, the LOC measures are difficult to interpret, since in certain situations a high value of LOC can mean a better modularization, or mean code replication by the project components.

It was verified in Garcia et. al. (2005) that due to the limitations of those metrics, they not can be analyzed in an isolated way. However they are extremely useful when analyzed together with others metrics. In addition, it was verified in Garcia et. al. (2005) that some researchers, as Henderson-Sellers (1997), have criticized the lack of a solid theoretical base and empirical validation of the cohesion metrics.

The AO implementation of the Data Access Object pattern presented in this study can restrict the extrapolation of the results obtained in this work. The evaluation process of the presented implementation is restricted to one specific instance of the Data Access Object pattern in this case the most used was selected.

5 RELATED WORKS

There are a few related works focusing on the quantitative evaluation of aspects oriented solutions to modularize crosscutting concerns of the classic design patterns. It was verified in Garcia et al (2005) that early experiments (Hannamann and Kiczales 2002) were based on qualitative measures. On the other hand the study undertaken by Garcia et al. (2005) presented a quantitative evaluation of the OO and AO implementations of the 23 design patterns of the GoF (Gamma et. al. 1995).

The section 4 exposed some criticism about the cohesion metrics. Seeking to optimize those metrics, the studies of Zao (2002) and Xu (2004) have proposed new cohesion measures that considerer the peculiarities of the AO abstractions and mechanisms. Those metrics are based on a dependence model for AO software that consists in a group of dependence graphs. In that study the authors have show that their measures have satisfied some properties good measures should have. However, as seen in Garcia et. al. (2005), those metrics have not been validated or applied to the assessment of realistic AO systems yet.

6 CONCLUSIONS

This study presents an alternative aspect oriented implementation of the Data Access Object pattern, and makes a comparative evaluation between the OO and AO of this pattern. The results of this study have been showing that the AO implementation improves the separation of concerns, reduces the amount of the lines of code (LOC) and optimizes the number of attributes (NOA) of the pattern. In relation to the coupling and cohesion metrics, there not was difference between the results in both implementations, due to the occurrence of dependence inversion between the BusinessObject and DAO components and due to the use of the same attributes and methods in both implementations. This occurs because the use of aspects increases the cohesion and reduces the coupling only in patterns that posses roles that are highly interactive. Meanwhile the WOC metric was the only metric that obtained inferior result in relation to the OO implementation. This result is a reflex of the advices declarations with many parameters in the AO solution.

With the obtained results, it can be conclude that the alternative implementation of the Data Access Object, approached in this work, provides a larger modularity and legibility in the development application code, facilitating maintenance and extension processes of the application, once that this solution provides the encapsulation of all data access functionalities of the application, due to fact that the business objects not more need to interact with the data access object, that is implemented with an aspect, which controls all the data access functions defined in the business object.

The approach presented in this work allows that all the data access layer of the application be implemented through aspects, providing a total isolation of the data access functions of the application. The positive results obtained by the application of this solution confirmed the optimization of legibility and modularity aspects in the development application, could be implemented in any application that needs access and persistence of data functionalities.

As a proposal for future researches it can be set a quantitative study about the OO and AO implementations of the Core J2EE patterns, aiming at verifying in which patterns the aspect oriented programming provides a better modularization in relation to the object oriented implementations.

Other proposal for future researches is the project of one aspect oriented software architecture, based on the discussed approach in this writing. This proposal has the purpose of reducing the increase in the complexity and dimension of the software systems projects currently caused by the increase in the complexity of the user's requirements and by the constant need of the integration of applications through several platforms, with the purpose of making possible the global access to the information.

REFERENCES

Alencar, P. et al. *A Query-Based Approach for Aspect Measurement and Analysis*. TR CS-2004-13, School of Computer Science, Univ. of Waterloo, Canada, Feb 2004.

Chidamber, S. & Kemerer, C., 1994, 'A Metrics Suite for OO Design'. IEEE Trans. on Soft. Eng*., 20-6,* 476-493.

Eclipse Project. http://www.eclipse.org.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J., 1995. *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U., 2005, 'Modularizing Design Patterns with Aspects: A Quantitative Study', International Conference on Aspect-Oriented Software Development (AOSD'05), Chicago, USA. ACM Press. Pages 3-14.

Garcia, A. et al., 2004, 'Separation of Concerns in Multi-Agent Systems: An Empirical Study. In *Software Engineering for Multi-Agent Systems', II, Springer, LNCS 2940*.

Garcia, A., Silva, V., Chavez, and C., Lucena, C., 2002, 'Engineering Multi-Agent Systems with Aspects and Patterns', *J. of the Brazilian Computer Society, 1, 8* (July), 57-72.

Hannemann, J. & Kiczales, G., 2002, 'Design Pattern Implementation in Java and AspectJ', Proc. OOPSLA'02, 161-173.

Henderson-Sellers, B., 1996. *Object-Oriented Metrics: Measures of Complexity,* Prentice Hall.

Jacabson, Ivar and Ng, Pan-Wei, 2004. Aspect Oriented Software Development with Use Cases. Addison-Wesley.

Java Reference Documentation. http://java.sun.com/refer ence/docs/index.html.

Massoni, T.; Alves, V. and Soares, S., 2001, 'PDC: Persistent Data Collections pattern', First Latin American Conference on Pattern Languages of Programming, Rio de Janeiro, Brazil.

Nock, Clifton, 2003, 'Data Access Patterns: Database Interactions in Object-Oriented Applications', Addison-Wesley.

Sant'Anna, C. et al. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. *Proc. of Brazilian Symposium on Software Engineering (SBES'03)*, Manaus, Brazil, 19-34.

Soares, S., 2004, 'An Aspect-Oriented Implementation Method', Doctoral Thesis, Federal Univ. of Pernambuco.

Soares, S. et. al., 2002, 'Implementing distribution and persistence aspects with AspectJ', ACM. In Proceedings of the OOPSLA' 2002, pp. 174-190, Seattle, USA.

Sun Microsystems 2002, 'Core J2EE Patterns: Data Access Object', viewed 01 November 2007, http://java.sun.com/blueprints/corej2eepatterns/Pattern s/DataAccessObject.html.

Zuse, H., 1995, 'History of Software Measurement. viewed 27 October 2007, http://irb.cs.tu-berlin.de/~