A PROPOSAL OF SOFTWARE ARCHITECTURE FOR MULTIPLATFORM ENVIRONMENT APPLICATIONS DEVELOPMENT

A Quantitative Study

André Luiz de Oliveira
Computting Department, Federal University of São Carlos,Washington Luiz highway Km 235, São Carlos, Brazil

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

Keywords: Software architecture, multiplatform systems, patterns, metrics.

Abstract: Due to the problems caused by the increase of the complexity and dimension of the software systems, becomes necessary the adoption of patterns and principles of software to deal with those problems. For this reason, the software architecture appears as new discipline in the Software Engineering field that is already being applied thoroughly in several areas. However there is a shortage of architectures proposals addressed for the multiplatform systems development. In this work it is proposed a software architecture for development of those systems. The project of that architecture model is based in the Data Access Object, Facade and Singleton patterns. The validation process of that architecture model used the three layer software architecture model as evaluation parameter, in which it was developed a quantitative assessment of two implementations of an application, one using the three layer architecture model and other using the proposed model. This study used strong software engineering attributes, such as separation of concerns, coupling, cohesion and size like evaluation criteria. As results, it was verified that the adoption of the architecture model presented in this work provides a better separation of concerns presents in the application components in relation to implementation using the three layer architecture model.

1 INTRODUCTION

Even with the progresses of the Software Engineering, every day many developers come across with several project problems during the software development process. Those problems arise of the increase of the complexity and dimension of the software systems (Shaw & Garlan 1994). All that complexity causes situations that violate the borders of the software project, negatively affecting the cost and the quality of the developed software.

The development of multiplatform environment applications raisin by this same problem, once that the users' requirements become more and more complex (MacWilliams & Brügge 2003) and there is a constant need of applications integration through several platforms, with the purpose of making possible the global access to the information. Due this fact and to the progress of the mobile devices technology, there is one crescent demand for the software systems integration for mobile platforms, seeking to expand the borders of those systems. The development of those application types' needs of new patterns and development paradigms, seeking to assist the demand for software's more and more complex.

Currently exist few proposals of methodologies focalized in the specific characteristics for the multiplatform systems development, like (MacWilliams & Brügge 2003), that need of global access to the information. Due to shortage of software project practices turned for the development of those systems types, becomes necessary the development of new principles and patterns destined to optimize the project process of

Luiz de Oliveira A., Luis Andrade Menolli A. and Gonçalves Coelho R. (2008).
A PROPOSAL OF SOFTWARE ARCHITECTURE FOR MULTIPLATFORM ENVIRONMENT APPLICATIONS DEVELOPMENT - A Quantitative Study.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - DISI*, pages 397-404
DOI: 10.5220/0001675103970404

those systems, reducing your complexity and dimension.

Due to need of patterns and methodologies turned for the multiplatform systems development, the present work proposes a software architecture for the development of those applications. The proposed software architecture model is based on the Data Access Object, Facade and Singleton patterns. In agreement with Gamma et. al. (1995) and Sun Microsystems (2002), the use of those patterns have the purpose of to improve the general performance of the application, reducing the consumption of resources of memory and processing, avoids the coupling between the data access and presentation components, providing larger legibility and modularity in the developed code, facilitating future maintenance and extension process of the application, optimizing the reuse of components, besides of promoting the storage mechanism independence used by the application, aiding the migration process for other storage mechanism.

The software systems generally are not projected to be connected to others platforms, becoming necessary the adaptation of those systems every time that is necessary make available its services to a certain platform. That needs of adaptation harm the processes of maintenance and extension of those systems. Due to that problem, the proposed software architecture model aim on provide a simple interface of access to the functionalities of a application, seeking to supply a better support to implantation of those applications through several platforms, like the web, mobile devices and desktop stations.

In order to prove the foundations of the proposed software architecture model, it was developed a quantitative assessment between two implementations of an application, one using the three layer architecture model and other using the proposed software architecture model. That process used strong software engineering attributes, such as separation of concerns, coupling, cohesion and size as evaluation criteria. The reason of the choice of the three layer architecture model for this comparison is justified by the fact of this model to be one of the architecture models more used on the applications development currently.

With the objective of explicit the proposed software architecture model, this study is structured in seven sections. The section 2 presents the definition of the proposed software architecture model. The section 3 approaches a case study in which it was applied the proposed software architecture model. In the sections 4 and 5 are presented the metrics and the assessment procedures used in the validation of the proposed model. The section 6 exhibits the outcomes obtained by the use of this architecture. And finally, the section 7 exposes the conclusions and the future works proposes.

2 THE PROPOSED SOFTWARE ARCHITECTURE MODEL

The proposed architecture model proposed in this work is based on the Data Access Object, Facade and Singleton patterns and on the N-tier architecture model (Malveau & Mowbray 2004), due to fact of this model to be the ideal for the complex software systems development. The objective of the proposed architecture ponders on getting better the general performance of the application, through the adoption of mechanisms that reduce the consumption of memory and processing resources, and flexibility the processes of maintenance and extension of the system, providing larger modularity and legibility in the development code, facilitating the reuse of several systems components.

The proposed software architecture model provides a simple interface of access to the functionalities of an application, facilitating the implantation process of that application through several platforms, once that this interface encapsulates all the services supplied by the application, what facilitates the distribution of those services for web interfaces, mobile devices, desktop stations, among others. This architecture model is constituted by the Gui, Facade, Beans, Daos and DBConnection layers, as illustrates the Figure 1.

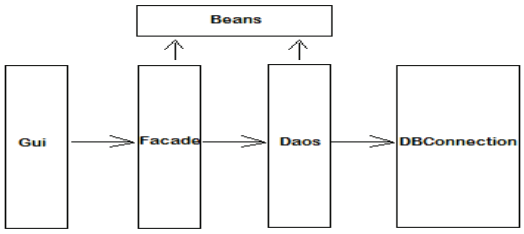

Figure 1: The proposed software architecture model.

The Gui layer is responsible by all process of interaction with the user. The components belonging to this layer can be Java Swing objects, servlets, JSP, PHP, ASP pages or a J2ME application. That layer is a data client, or be she just contains mechanisms of data presentation for the final user.

The Facade layer encapsulates all the business logic of the application, supplying one access

interface to the complex data access layer for the Gui layer. Its purpose is impede that components belonging to the Gui layer, directly access the data access mechanisms, once this cause a high coupling between Gui and Daos layer, harming the reuse of several components. The objective of this layer basically consists on making available a simple interface of access to the complex functions of the subsystems, or be, the data access functions of the application, to the presentation components. On the proposed architecture model, the Facade pattern (Gamma et. al, 1995) is applied in this layer, in virtue of this pattern to structure the system in subsystems, reducing its complexity, providing a simple interface of access to its functionalities, making them more reusable and maintained, reducing the coupling between the clients classes and the subsystems, promoting a larger independence and portability of those subsystems.

The Beans layer is responsible for encapsulating every the business model of the application. The business model corresponds to the information model that an application will manipulate. The reason of the existence of that layer is based in isolate the business vocabulary of the application of the others layers, facilitating future processes of extension of the system. Those components only possess attributes and get and set methods declarations to manipulate your information.

The Daos layer is responsible for all the data access functions of the application. This layer is the place where is the whole insert, update, delete and query code, executed by the Database Management System (DBMS). The existence of this layer is justify by the reason of that the data access mechanisms vary conform the chosen storage system, affecting futures maintenance and extension processes of the system, once that without this layer, a change in the storage system carts alterations in the data access functions of the application, that meet dispersed by several modules of the application. The addition of this layer provides a better separation of concerns, isolating the data access functions of the others functions of the application. The Data Access Object pattern was used to implement the belonging components to this layer, in virtue of this pattern to isolate all the data access functions of the others functions of the application and of providing a independence of selected storage mechanism, avoiding the coupling and facilitating future maintained and migration processes for others storage systems. This layer is constituted through classes of data access that possess a group of operations that make the persistence and the recuperation of information in the storage mechanism of the application.

And finally, the DBConnection layer is responsible by the connectivity functions with DBMS (Database Management System), or other storage mechanism. This layer possesses mechanisms that control the amount of instances of access components to the database resources, impeding the existence of redundant components to access a same resource, avoiding the waste of memory and processing resources, improving the general performance of the application. The reason of the existence of that layer is based in isolate the functions that interact with the storage mechanism of the others systems components. This layer use the Singleton pattern for control the instantiation of those components, once that this pattern (Gamma et. al. 1995) assure that a class has only a unique instance during the system execution, and guarantee a global point of access to them. In situations of concurrent access can be adopt the strategy of creation of a pool of components that interact with the storage mechanism using the Singleton pattern.

3 CASE STUDY

With the finality of demonstrating the foundations of the proposed software architecture model, this section presents the three layer implementation and the implementation that use the proposed software architecture model of a real application that makes the control of the laboratories reservation process of the Computer Science Department of the State University of the North of the Paraná, campus Bandeirantes.

The applications developed using the three layer architecture model are implemented commonly through of the Model-View-Controller (MVC) pattern (Sun Microsystems 2000). This pattern is used to develop applications that need to support multiple clients, or be, applications that need make available the access to its services through of several interfaces (HTML, Swing, WML for mobile devices).

The application of the MVC pattern provides a better separation of the business model of the presentation functionalities and of the control logic of the application (Sun Microsystems 2000). Such separation allows that multiples views share the same information model, what makes possible the support to multiples clients easy to implement, maintain and to test. This pattern divides the application implementation in three layers: model, view and control. The model represents the data and

the business rules that govern the access and the updating of those data. The view renders the content of the model, access the data and specifies the way like those data are presented. This layer is responsible by maintain the consistence in the presentation when occur changes in the model components. The control layer is entrusted of convert the interactions of the view layer in actions to be executed by the model. Those operations correspond to the activations of business process or in the state change of the model, as illustrates the Figure 2.

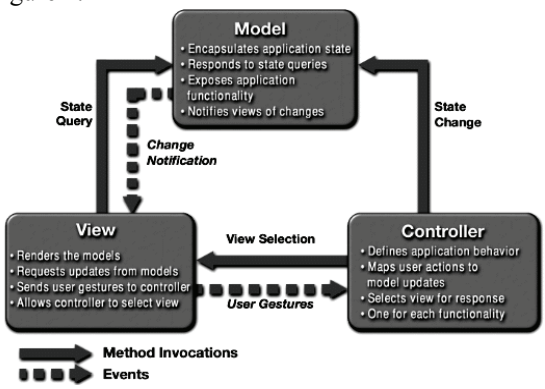

Figure 2: The Model-View-Controller design pattern.

The three layers application implementation presented in this study is constituted by the Beans, that represent the model, Gui, that represent the view and Daos layers, that represent the control, as illustrate the Figure 3. The Gui layer possesses a group of components responsible by the interaction process with the user. Already the Beans layer contains a group of components that implement the business rules of the application. And finally, the Dao layer possesses the components responsible by the business and data access logic of the application.

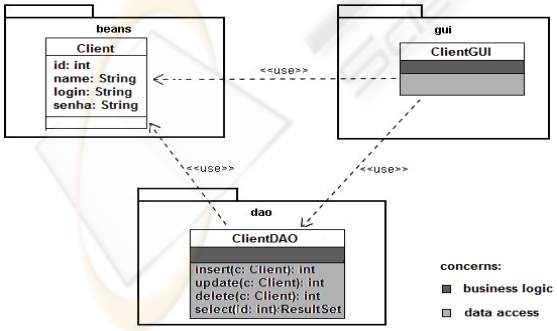

Figure 3: Class diagram of the three layer implementation of the application.

The three layers implementation of the reservation system presented in the Figure 3 provides a good separation of the functions of the application, isolating the business rules implementation, the control logic and presentation logic in modules, facilitating future maintenance and extension processes of the application. However in spite of the good modularization that this implementation seemingly provides, she not possesses the capacity of completely isolate the concerns of the application, once that inside of the ClientGUI component implementation are presents the concerns that implement the business logic and data access logic instead of only containing the presentation and user interaction functions, as illustrates the code space shaded of the Figure 4. That whole dependence harms the distribution of the application for several platforms, in virtue of the need to redraft code that implements those concerns every time that is added a new interface of access to the services of that application.

```
01 import beans.Client;
02 import daos.ClientDao;
03
04 public class ClientGui extends javax.swing.Dialog {
05
06    public ClientDao cdao;
07    public Client cli;
08
09   public ClientGui(){
10     cdao=new ClientDao();
11     cli=new Client();
11   }
12   private void btnExecuteActionPerformed(java.awt.event.ActionEvent
13     evt) {
14     int value=0;
15     cli.setName(txname.getText());
16     cli.setAge(Integer.parseInt(txage.getText()));
17     cli.setDate(Date.valueOf(cbdate.getSelectItem().
18     toString()));
19     cli.setAddress(txaddress.getText());
20     value=cdao.insert(cli);
21
22     if(value==1){
23       JOptionPane.showMessageDialog(null, "sucess register
24       ","Alert message",
25       JOptionPane.INFORMATION_MESSAGE);
26       this.dispose();
27     }
28     else{
29       JOptionPane.showMessageDialog(null, "error in the database",
30       "Alert message",
31       JOptionPane.INFORMATION_MESSAGE);
32     }
33   }
34 }
```

Figure 4: ClientGui class implementation in the three layer model.

Observing the ClientGui component implementation in the Figure 4, can be ended that the use of the three layer architecture model don't get entirely modularize the present concerns in the application, taking to the development of the components of weak cohesion and highly coupled, in that the ClientGUI component implements as the data access concerns as the business logic, besides of the concern for which that component was really projected, that corresponds to the interaction and data presentation functions to the user. The implementation of that component is strongly coupled to the business and data access components, to implement its functionalities, harming futures processes of maintenance and extension of the application, reducing the potential of reusability of the components and hindering the application distribution process to others platforms.

The project of the software architecture model proposed in this work arose during the development of a laboratories reservation system, destined to attend the needs of the Computer Science Department of the State University of the North of the Paraná, campus Bandeirantes.

During the modeling project of that system happened some problem situations, where it was verified that the use of several instances of the same object to access database resources, harms all the performance of the application, once those instances occupy larger memory space and demand more processing resources. It was verified also that the data access mechanisms can vary conform the chosen storage system. This can affect futures maintenance processes of the system, besides of impeding the reuse of several components. Another important point verified during the modeling process was the possibility of the presentation components (system interfaces) access directly the complex data access layer, causing a high coupling between those components.

In order to solve the found difficulties in the project of the application, it was used one combination among the Data Access Object, Facade and Singleton patterns, as illustrates the Figure 5. A complete description of each one of those patterns can be found in Sun Microsystems (2002) and Gamma et. al. (1995).

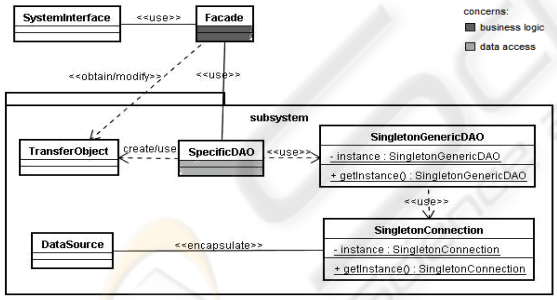

Figure 5: Class diagram of the solution.

In the proposed solution, the Singleton pattern is used for guarantee the existence of one only instance of the SingletonConnection and SingletonGenericDAO classes, getting better application performance. The SingletonConnection encapsulates the DataSource and the SingletonGenericDAO supply an access interface to the SingletonConnection. To separate the application code of the data access code, it was applied the Data Access Object pattern (DAO), where was adopted the creation of one generic DAO strategy, that works as a Singleton access interface. Each SpecificDAO accesses the present functionalities in the generic DAO and use a TransferObject to make the data transport. And finally, to avoid that the presentation components access directly the data access layer, the use of the Facade pattern was stipulated, once that it encapsulates the whole complex interface of the subsystem and supply one simplified interface to access its functionalities.

The application developed aims to automate and activate the laboratories reservation process made by teachers and supply larger access mobility to the users of the reservations service through several interfaces, where it was developed a J2SE application, acting as Bluetooth server, a J2ME application, acting as Bluetooth client, a Web page, using the J2EE technology and a J2ME application of Web access.

4 METRICS

This study selected a group of metrics of separation concerns, coupling, cohesion and size (Chidamber & Kemerer 1994) to evaluate the foundations of the software architecture model proposed in this work. These metrics have already been used in four different studies (Garcia et. al. 2005; Garcia 2004; Garcia et. al. 2004; Soares 2004).

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 to this study, and associates them with the attributes measured by each one. More detailed information about these metrics can be found in Chidamber and Kemerer (1994).

5 ASSESSMENT PROCEDURES

With the purpose of explicitly the foundations of the proposed software architecture model, it was elaborated a quantitative assessment of the proposed architecture model and of the three layer architecture model. In this evaluation, both the application versions, presented in the case study, implement the same functionalities, with the same codification style. A few modifications happened on the implementation that uses the proposed software architecture model in relation to the implementation that use the three layer architecture model, in that

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.

some attributes and methods were removed of some classes and others classes were added, seeking to provide larger modularity, legibility and reuse of the application components.

In that measurement process, the data were gathered with base on the code analysis, using the Eclipse 3.3 tool. The measures of separation of concerns (CDC, CDO and CDLOC) were preceded by the shading of all classes in both application implementations. That shading it was accomplished with the roles found in the implementation of the classes of the application. The Figures 3 and 5 exemplify the shading of some classes in both applications implementations, considering the Business Logic and Data Access roles. The Business Logic role is responsible by all control of the business logic of the application, that means, it treats the input data before those data be passed into the call of a data access function, besides manipulating the obtained data of a data access function, to summarize that role is a data client (Sun Microsystems 2002). Already 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 that means, it provides all functionalities of access and data storage to the Business Logic role.

Likewise to the Hannemann & Kiczales (2002) study, in this study each role found in the application was treated as a concern, once that roles are primary sources of crosscutting structures (Garcia et. al. 2005). After the shading, the separation of concerns data metrics was manually collected.

6 RESULTS

This section presents the measurement process results. The data have been collected with base on the metrics defined in section 4. The objective of that measurement process is describe the results of the metrics application in the three layer implementation and in the implementation that uses the proposed software architecture model, of a real application, with the finality of compare and prove the foundations of the proposed software architecture model. This analysis is divided in two parts. The section 6.1 focus on the analysis of what extent in that both solutions provide support to the separation of the application-related concerns. The section 6.2 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 of the measurement process. The results of the graphs present the data obtained from both applications implementations. 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 both applications implementations, is attached to a percentage value, which represents the difference between the results of the implementations that use the three layer model and the proposed architecture model. A positive percentage means that the implementation of the proposed architecture model was superior, while the negative percentage means that the implementation of the proposed architecture model was inferior. Those graphics support the analysis toward how the introduction of new classes affects both solutions in relation to the selected metrics.

In order to obtain the separation concerns metric values of both the application implementations, firstly it was verified the presence of two roles in the application implementation, the Business Logic role, that consists basically in the business rules of the application, and the Data Access role, that corresponds to the data access functionalities of the application.

6.1 Separation of Concerns

The Figure 6 presents the separation concerns metrics results for both application implementations. As it is observed in the Figure 6, the most of the measurements significantly the implementation that uses the proposed software architecture model in this work. This solution reduced the coupling of the classes that play the Business Logic role, and consequently the number of operations for implementing that role, besides demanding few concerns switches between the components that play the Business Logic and Data Access roles.

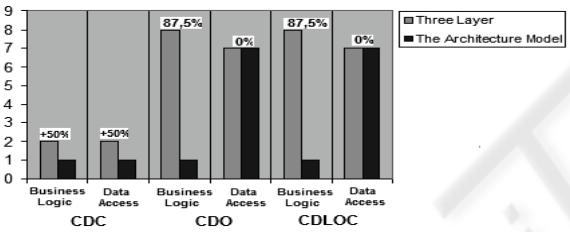

Figure 6: Results of separation of concerns metrics.

An analysis of the Figure 6 shows that those improvements was reached through of the separation of concerns related to the application in a additional layer of Facade classes, that possess the finality of provide a simplified interface of access to the services of the application. The definition of the Data access and Business Logic roles demanded 2 classes in the three layer implementation, while that the use of the proposed software architecture model reduced this number for 1, providing a better separation of those roles in relation to the three layer solution. This improvement is equivalent to the 50% of superiority of the proposed architecture model with relation to the three layer implementation. The results were even better for the concern diffusion over operations (CDO) and concern diffusion over lines of code (CDLOC) metrics on the implementation of the Business Logic role, which reached optimizations of 87,5% in relation to the three layer implementation.

In addition, it can be observed that good results were reached in the modularization of the Business Logic role. After that analysis, is ended that the implementation that uses the proposed software architecture model optimized about 74,6% the isolation of the implemented concerns by the Business Logic role in comparison with the use of the three layer architecture model.

One of the reasons for the superiority of the proposed architecture model over the three layer architecture model is that in the three layer solution there are several operations implementations mixed with the specific code of the role.

6.2 Coupling, Cohesion and Size

In this section are presented the results of the coupling, cohesion and size metrics. It was used graphs for the representation of the gathered results, which represents the metric values associates with all the classes for each application 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 implementation of the application that uses the proposed software architecture model, the improvements were reached in the CBC and NOA metrics, as illustrates the Figure 7. The use of the proposed architecture model increased in 12,52% the LOC metric value in relation to the three layer implementation. This happen due to the addition of a component that makes the manipulation of the business logic of the application in this solution.

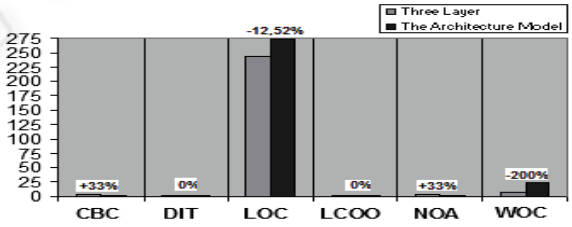

Figure 7: Coupling, cohesion and size metrics results.

In relation to the NOA metric results, the implementation using the proposed architecture model obtained 33% of superiority in relation to the three layer solution. This happen in virtue of in the three layer solution, the classes that carry out the presentation logic possess a reference to the data access object and make all the manipulation of the business logic of the application, that in the project that uses the proposed architecture model was retired. The use of this model also provides a reduction of 33% of the coupling between components (CBC) metric value in relation to the traditional solution in three layers.

With relation to the cohesion metrics, its values continued constant in both solutions, once they use the same operations and references to the attributes. The value of the WOC metric was the only in which the proposed architecture model obtained result inferior to the three layer solution. This inferiority corresponds to the 200%. The reason of this is based on the fact that the proposed architecture model adds a component, called Facade that is responsible by the manipulation of the business logic of the application. This component has declarations of methods with many parameters, increasing the value of this metric.

7 CONCLUSIONS

The present work approaches a proposal of the software architecture based in the Data Access Object, Facade and Singleton patterns, addressed to the multiplatform systems development. This architecture optimized the following aspects of the software systems: the general performance of the application, once that were eliminated several redundant instances of data access components of the application, reducing the consumption of memory and processing resources; brought larger flexibility in the maintenance and extension process of the application, reducing the complexity of the software components; and provides larger modularity and flexibility of the developed software components, favoring the reuse of several components of the application.

The positive results obtained by the application of that architecture confirmed its efficiency and effectiveness in the multiplatform environment systems development process, could be applied in others similar application projects.

As proposed of future works can be placed a study about the approaches of formal specifications of software architectures approaches, the calls ADLs (Architecture Description Language), in order of selecting one of those approaches and apply her in the specification of the software architecture proposed in this work.

Like another proposal of future work is the study of the use of the aspect oriented abstractions in the project of software architectures, due to effectiveness of the aspect oriented methodology in the modularization of crosscutting concerns that meet tangling and scattering for several modules of the application, with the purpose of providing an improvement still larger in the separation of the concerns presents in the application.

REFERENCES

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

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., 2004, 'From Objects to Agents: An Aspect-Oriented Approach', Doctoral Thesis, PUC-Rio, Rio de Janeiro, Brazil.

Garcia, A. F., Sant''anna, C., Chavez, C., Silva, V., Lucena, C. J. P. de. & Staa, A. V., 2004, 'Separation of Concerns in Multi-Agent Systems: An Empirical Study', In Software Engineering for Multi-Agent Systems II, Springer, LNCS 2940.

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

MacWilliams, A. & Brügge, B., 2003, 'Self-Extending Systems for Context-Aware Mobile Computing', International Conference on Software Engineering, Portland, Oregon, USA.

Malveau, R. & Mowbray, T. J., 2004, 'Software Architecture: Basic Training', Prentice Hall PTR, viewed 25 October 2007, http://www.phptr.com/articles/article.asp?p=169547&seqNum =12&rl=1.

Sant'Anna, C., Garcia, A., Chavez, C., Lucena, C., Staa, A. V., 2003, '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.

Shaw, M. & Garlan D., 1994, 'An Introduction to Software Architecture'. School of Computer Science Carnegie Mellon University Pittsburgh, PA.

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

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

Sun Microsystems 2000, 'Model-View-Controller', viewed 27 October 2007, http://java.sun.com/blueprints/patterns/MVC-detailed.html.