An Architecture for the Evolution of Web Applications

Paulo Caroli
Computer Science Department
PUC-Rio, R. Marquês de S. Vicente, 225, Rio de Janeiro, RJ, Brazil, 22453-900
caroli@les.inf.puc-rio.br

Carlos José P. de Lucena
Computer Science Department
PUC-Rio, R. Marquês de S. Vicente, 225, Rio de Janeiro, RJ, Brazil, 22453-900
lucena@inf.puc-rio.br

Marcus Fontoura
Computer Science Department
Princeton University, 35 Olden Street, Princeton, New Jersey, USA, 08544
mfontoura@acm.org

ABSTRACT

This work presents a software architecture that is especially useful for managing the evolution of web applications. Web-based systems are a range of applications for which there are no technological standards and new concepts and tools are currently under evolution. Examples of this lack of standards include the transition from CGI scripts to Java Servlets and to Java Server Pages (JSP). Therefore, the maintenance and evolution of web applications is an important topic for software developers and the software research community. The proposed architecture combines the n-tier, broker, and blackboard architectural patterns.

1. THE ARCHITECTURE

A 3-layered architecture [1] separates the system functionality into user interface, business logic, and persistency. Brokers [4] encapsulate the interface between layers. Finally, a blackboard class repository models entities that do not naturally belong to any of the three layers. Figure 1 abstractly illustrates this hybrid architecture.

The interface layer is responsible only for validating the user interface input (e.g. verifying if all required fields have been provided). It does not perform any business operation. Technologies generally used in this layer include HTML, Java Servlets, and JSP.

The business layer models the business logic, independently from the access interfaces and persistency models. This layer focuses on the system behavior – the business rules – and not on the data being manipulated, which is modeled by blackboard classes.

The persistency layer is responsible for the physical storage of the data. It may use a wide variety of persistency models, from flat files to complex frameworks for handling heterogeneous databases [3].

The brokers that implement the interlayer communication are responsible for selecting the required information and converting it to an appropriate format. One of the main responsibilities of the interface-business broker is the conversion between strings that come from web forms into business objects that will handle the HTTP request. The business-persistency broker is responsible for manipulating persistent data to fulfill requests from business objects.

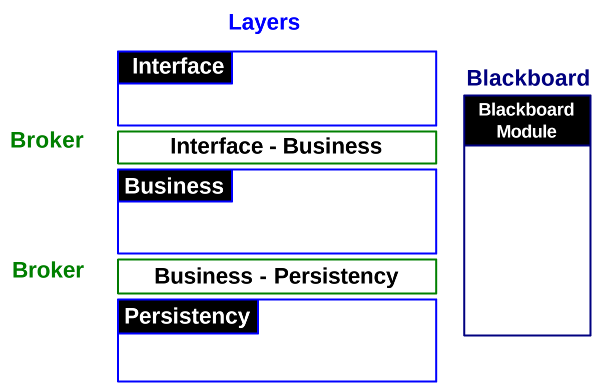

Figure 1. The architecture

Classes that do not belong to neither the three layers nor the brokers are placed in the blackboard module. These classes model real-world objects, such as employees and managers. Generally they represent persistent information and are used in the business-persistency communication protocol.

This architecture separates data from functionality: the data is provided by the blackboard classes while each of the three layers provides the appropriate functionality. Another important aspect is data conversion. In general the business layer and the blackboard will be implemented using OO technology (e.g. Java) while the interface and persistency layers may use other approaches (e.g. events and relational databases). The brokers are the entities that should make the appropriate conversions, making the interlayer communication independent of the technologies used to implement each layer.

The loose-coupled nature of this architecture makes it possible to evolve each layer independently from the others. This is especially useful for web applications, for which new techniques and tools are released constantly and no standards are defined yet. Figure 2 illustrates abstractly the evolution of an access control system [2]. It was first developed with Java Servlets as the user interface and flat files as the persistency model. A second version of the system was then developed with JSP and relational database. The architecture allowed no changes to be made to the business layer and blackboard classes, when evolving the system.

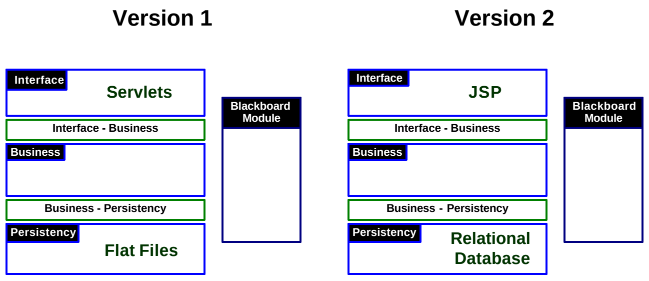

Figure 2. System evolution example

2. OBSERVED ADVANTAGES

We have successfully applied this architecture to several large web-based systems [2]. These experiments have shown us some advantages of this approach, which include:

- Database changes: within the proposed architecture, changes on the data model require modifications only on the persistence layer, the business-persistency broker, and on the blackboard data classes. In the case that business logic and interface are mixed with the persistency code, any changes in the data model may require changes of several parts of the system. In the case of a drastic change (e.g. changing from relational to OO database systems) the complete redesign of the system may be required;
- Interface changes: changes are confined to the interface layer and to the interface-business broker (e.g. changing from a Java Servlet to a JSP solution should not result in a general reengineering of the system - the business layer, persistency layer, business-persistency broker and blackboard should remain the same);
- Legacy system integration: legacy systems can be treated as data repositories, and their integration with the rest of the system can be made through specific business-persistency brokers that act as system wrappers. Figure 3 illustrates this situation;
- Maintainability and evolution: addition (or elimination) of features and changes of implementation technology are facilitated through the use of the architecture, since changes are encapsulated in layers and the system functionality is well distributed in proper modules;
- Concurrency: the architecture allows better concurrency control, due to the low granularity of data and functionality (e.g. access policies for handling data concurrent requests may be encapsulated on the blackboard module);
- Performance issues: since the system is well organized, performance bottlenecks can be more easily identified. Moreover, optimizations are confined to specific layers;
- Management issues: aspects such as team division, code ownership, cost control, and progress measurement, are also better implemented when a well-defined architecture is used.

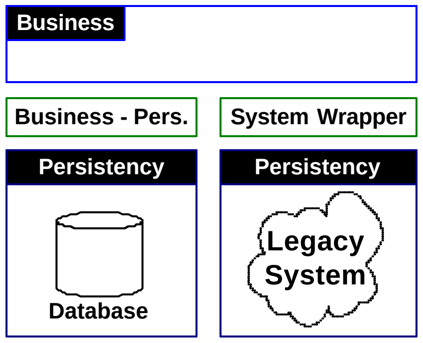

Figure 3. Legacy system integration

3. CONCLUSIONS AND FUTURE WORK

Several case studies have shown that the proposed architecture is effective for the development of web-based applications [2]. It accommodates drastic changes to the data and user interface models smoothly. It also allows for better development practices.

We plan to extend UML case tools to support the proposed architecture directly. One approach to do that is to mark architectural information in UML class diagrams, and use this information to generate code with respect to the architecture. We have already started this work using Rational Rose (http://www.rational.com/products/rose) as the UML case tool.

4. ACKNOWLEDGMENTS

We would like to thank Prof. Sergio Carvalho (in memorian) for his immense contribution to this work.

5. REFERENCES

[1] F. Buschman, R. Meunier, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture, A system of Patterns*, John Wiley & Sons, 1996

[2] P. Caroli, *An Object-Oriented Methodology for Software Projects*, M.Sc. Dissertation, Computer Science Department, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), 1999 (in Portuguese).

[3] E. Uchôa and R. Melo, *HEROS: A Framework for Heterogeneous Database Systems Integration*, in Proceedings of the Tenth International Conference on Database and Expert Systems Applications (DEXA'99), LNCS 1677, 656-667, Springer-Verlag, Florence, Italy, 1999.

[4] M. Shaw and D. Garlan, *Software Architecture - Perspectives on an Emerging Discipline*, Prentice Hall, 1996.