1º Workshop Pronex sobre Framewoks

Pontifícia Universidade Católica do Rio de Janeiro
Departamento de Informática
Rua Marquês de São Vicente, 225, 22453-900
Rio de Janeiro, Brasil

PUC-Rio.Inf.MCC 51/98 Dezembro, 1998

ABSTRACT

During the last years, Framewoks have been an important research issue in the Software Engineering area. This technology has been increasingly adopted within different research and development domains. At the PUC-Rio Computer Science Department, several research topics within specific domains have been working in order to use the framework technology as a means of increasing quality, reusability, organization and flexibility in both domain specific modeling and software construction. The Mobile Project (Pronex) aims to approximate these different efforts stimulating cooperation among those groups. For that sake, during the year of 1998, a series of seminars were held providing a forum for discussion of the work in progress. In order to finish the year's activities, the *1st Pronex Frameworks Workshop* was held on December 15th and 21st under the coordination of Prof. Carlos José Pereira de Lucena. This technical report compiles the submitted position papers. Further information can be accessed at http://www.les.inf.puc-rio.br/~pwpf.

Keywords: Frameworks, Design Paterns, Components.

RESUMO

Frameworks têm sido alvo de grande parte das pesquisas na área de Engenharia de Software nos últimos anos. Sua utilização tem sido adotada cada vez mais em diversos domínios de pesquisa e desenvolvimento. No Departamento de Informática da PUC-Rio, vários ramos de pesquisa voltados a temas específicos vêm desenvolvendo trabalhos visando utilizar a tecnologia de frameworks. Tais iniciativas têm, de uma forma geral, por objetivo aumentar a qualidade, reuso, organização, flexibilidade na modelagem dos conceitos dos respectivos domínios e no desenvolvimento de software. O Projeto Mobile (Pronex) visa aproximar esses esforços estimulando a cooperação entre os diversos grupos. Nesse sentido, realizou-se, ao longo do ano de 1998, uma série de reuniões com apresentações e discussões dos diversos trabalhos em andamento. Marcando o encerramento das atividades, realizou-se, nos dias 15 e 21 de dezembro de 98, o *1º Workshop Pronex sobre Frameworks*, sob a coordenação do Prof. Carlos José Pereira de Lucena. Esta monografia reune os "position papers" dos trabalhos apresentados. Outras informações envolvendo as atividades desenvolvidas no âmbito desse projeto podem ser obtidas através da página http://www.les.inf.puc-rio.br/~pwpf.

Palavras chaves: Frameworks, Design Paterns, Componentes.

Coordenador: Prof. Carlos José Pereira de Lucena

Email: lucena@inf.puc-rio.br

Editores: Sérgio Crespo
Fernanda Lima
Marcus Felipe Fontoura
Fábio Porto
Sérgio Colcher

Email: {crespo,ferlima,mafe,porto,colcher}@inf.puc-rio.br

Índice

Ehancing Framework Design and Utilization

Marcus Felipe M. C. da Fontoura, Carlos José Pereira de Lucena
{mafe,lucena}@inf.puc-rio.br

Laboratório de Engenharia de Software, Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introduction

This work proposes a new approach for framework design and utilization. We believe this approach will lead to a more explicit definition of the collaboration between the framework elements and will help the designer to better implement the framework variability through the use of patterns. Regarding framework utilization, we propose a different instantiation process, where a particular application is described by domain specific languages (DSLs) that are designed precisely for the framework domain. A major topic in our work is the semi-automatic derivation of the DSLs from the framework design structure. The use of formal methods will allow us to prove that the applications that can be derived from a framework through the "standard" instantiation process are exactly the same that can be specified as DSLs programs.

2. Problem and Motivation

There are various application areas that are not established yet and for which new ideas and models are presently under development and evaluation. These are domains for which the viability of rapidly building new applications is essential and strategic from a practical point of view. Examples of application domains that can be classified in this category include Web-based Education, electronic commerce, biology, and financial market.

An interesting strategy for developing new applications for these domains is the development of object-oriented application frameworks [11, 9]. Frameworks can reduce the costs of developing applications since it allows designers and implementers to reuse their previous experience with problem solving at the design and code levels. Prior research has shown that high levels of software reuse can be achieved through the use of object-oriented frameworks. An object-oriented framework captures the common aspects of a family of applications, which can be seen as a domain theory [18].

Although object-oriented software development has experienced the benefits of using frameworks, a thorough understanding of how to identify, design, implement, use, and change them to meet evolving requirement needs is still object of research [18, 23].

Therefore, framework development is very expensive not only because of the intrinsic difficulty related to capturing the domain theory but also because of the lack of appropriate methods and techniques to support framework specification and design.

Framework utilization is also a problem noticed in companies that use this technology to create applications [http://www.objectime.com/OOPSLA98/frameworks/]. The most common way to instantiate a framework is to inherit from some abstract classes defined in the framework hierarchy and write the code that is called by the framework itself. However, it is not always easy to identify which code is needed and where it should be written since frameworks class hierarchies can be very complex, especially for non-expert users.

Therefore, the "standard" instantiation process is not always the ideal way to describe a particular application. This happens because of several facts:

- the language used to build and use the framework is a wide spectrum language, in which it is not always easy to express the user intentions;
- the complexity of framework class hierarchies and the difficulty of finding the points where the code should be written: the framework hot-spots or flexible points [16, 19].

This work proposes a new approach for framework design and utilization that addresses these problems.

3. Background and Related Work

Techniques such as design patterns [9], meta-object protocol [13], refactoring [12], and class reorganization [5] have been proposed to support framework design. However, they do not address the problem of helping the framework designer to find the commonalties (frozen spots) and flexible points (hot-spots) of a given framework. Our approach extends the idea of design patterns essentials [15, 16] to address this problem.

Domain specific languages (DSLs) are used to formally specify software designs [10]. A DSL is a formal language that is expressive over the abstractions of an application domain. An advantage of using a DSL is that the domain expert can express domain specific concepts in a formal way directly, instead of communicating the specification informally to a software specialist who is usually less familiar with the intended application.

An approach for using DSLs to enhance software libraries utilization is described in [17, 22]. The motivation for their work is that end users often do not make effective use of libraries, because most of them lack the expertise on properly identifying and composing the routines that form their application. They argue that in domains with mature subroutine libraries, one can greatly improve the productivity and quality of software engineering by automating the effective use of those libraries. Our work is an enhancement of this approach for frameworks.

4. Approach and Uniqueness

We propose the use of formal methods, specifically the Z specification language [20, 21], to describe the semantics of the framework design structure. For this reason, we have developed an object-oriented meta-model in Z and will use this meta-model to describe all the artifacts and processes involved in our approach. There is a direct mapping between the syntax of our meta-model and the current object-oriented programming languages such as Java and C++.

Our approach to framework design is based on the idea that any design can be divided into two parts: the kernel sub-system design and the hot-spot sub-system design. The kernel sub-system design is common to all the applications that the framework may generate, and the hot-spot sub-system design describes the different characteristics of each application that can be supported by the framework. The hot-spot sub-system uses the method and the information provided by the kernel sub-system and may extend it. The hot-spot sub-system elements are the ones that vary for each application, and depend on the use of the framework. These elements must be adapted to each generated application.

Several unification rules, based on our meta-model, that show how frameworks can be derived from an initial set of applications have been defined as part of our work. These rules are used to structure and classify the framework architecture, highlighting the relationship between the kernel and hot-spot elements. Design pattern essentials [15, 16] are then applied to fulfill the flexibility requirements of each framework hot-spot.

We believe this approach will lead to a more explicit definition of the collaboration between the framework elements and will help the designer better implement the framework variability through the use of patterns.

Regarding framework utilization, we propose a different instantiation process, where a particular application is described by domain specific languages (DSLs) that are designed precisely for the framework domain. The technique basic idea is to capture domain concepts in DSLs, which will help the framework user to build code in an easier way, without worrying about implementation decisions and remaining focused on the problem domain. The specification written in each DSL is translated to the framework instantiation code through the use of transformational systems [14, 6].

A major topic in our work is the semi-automatic derivation of the DSLs from the framework design structure. The use of formal methods will allow us to prove two important theoretical results:

- there is always a mapping between frameworks and DSLs, which means that for every framework there are associated DSLs;
- the applications that can be derived from a framework through the "standard" instantiation process are exactly the same that can be specified as DSLs programs (using the associated, derived DSLs) .

5. Results and Contribution

A set of desirable properties of a solution determines whether or not the solution is satisfactory. The proposed approach to enhance object-oriented frameworks design and utilization should posses the following properties.

- General: it should be general enough to be used in any application domain. This generality will be shown through case studies in different application areas;
- Precise: it should be precisely described, in order to be understood by its users and also to allow the definition of supporting environments. This property will be addressed through the formalization of the artifacts and processes involved in the approach;

Currently comparable methods are almost nonexistent or in their infancy. Case studies in three important and innovative application domains, in which we expect to contribute with the design and development of real frameworks, is also part of our work.

A number of papers [1, 2, 3, 4, 7, 8] report our current results. The consolidation of these research results will be present in my upcoming Ph.D. dissertation (July 99).

References

1. P. Alencar, D. Cowan, T. Nelson, M. F. Fontoura, and C. J. Lucena, "Viewpoints and Frameworks in Component-Based Design", in *Object-Oriented Application Frameworks*, M. Fayad, D. Schmidt (editors), John-Wiley, 1998.
2. P. Alencar, D. Cowan, S. Crespo, M. F. Fontoura, and C. J. Lucena, "Using Viewpoints to Derive a Conceptual Model for Web-Based Education Environments", MCC17/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998 (also submitted to Journal of Systems and Software)*.
3. C. Braga, M. F. Fontoura, E. H. Haeusler, and C. J. Lucena, "Formalizing OO Frameworks and Framework Instantiation", First Brazilian Workshop on Formal Methods, 1998.
4. C. Braga, M. F. Fontoura, C. J. Lucena, and L. M. Moura, "Using Domain Specific Languages to Instantiate OO Frameworks", MCC28/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998 (also submitted to IEE Proceedings – Software Engineering)*.
5. E. Casais, "An incremental class reorganization approach", ECOOP'92, *volume 615 of Lecture Notes in Computer Science*, 114-132, 1992.
6. J. Cordy and I. Carmichael, "The TXL Programming Language Syntax and Informal Semantics", Technical Report, Queen's University at Kinkston, Canada, 1993.
7. M. F. Fontoura, L. M. Moura, S. Crespo, and C. J. Lucena, "ALADIN: An Architecture for Learningware Applications Design and Instantiation", MCC34/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998 (also submitted to WWW Journal)*.
8. M. F. Fontoura, E. H. Hermann, and C. J. Lucena, "A Framework Design and Instantiation Method", MCC35/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, 1998 (also ACM TOSEM)*.
9. E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, Design Patterns, *Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
10. P. Hudak, "Building Domain-Specific Embedded Languages", *ACM Computing Surveys*, 28A(4), 1996.
11. R. Johnson, "Frameworks = (Components + Patterns)", *Communications of the ACM*, 40(10), 1997.
12. R. Johnson and W. F. Opdyke, "Refactoring and aggregation", Object Technologies for Advanced Software, First JSSST International Symposium, *volume 742 of Lecture Notes in Computer Science*, 264-278, 1993.
13. G. Kiczales, J. des Rivieres, and D. G. Bobrow, *The Art of Mataobject Protocol*, MIT Press, 1991.
14. J. C. S. P. Leite, M. Sant'anna, and F. G. Freitas, "Draco-PUC: a Technology Assembly for Domain Oriented Software Development", 3rd IEEE International Conference of Software Reuse, 1994.
15. W. Pree, *Design Patterns for Object-Oriented Software Development*, Addison-Wesley, 1995.
16. W. Pree, *Framework Patterns*, Sigs Management Briefings, 1996.
17. Pressburger, T. and M. Lowry, "Automatic Domain-Oriented Software Design using Formal Methods", Software Systems in Engineering, Energy-Sources Technology Conference and Exhibition, 33-42, 1995.
18. D. Roberts and R. Johnson, "Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks" in *Pattern Languages of Program Design 3*, Addison-Wesley, 1997.
19. H. A. Schmid, "Systematic Framework Design by Generalization", *Communications of the ACM*, 40(10), 1997
20. J. M. Spivey, *Understanding Z: A Specification Language and Formal Semantics*, Cambridge University Press, 1988.
21. J. M. Spivey, *The Z Notation: A Reference Manual*, Prentice Hall, Hemel Hempstead, 1989.
22. Stickel, M., R. Walddinger, M. Lowry, T. Pressburger, and I. Underwook, "Deductive Composition of Astronomical Software from Subroutine Libraries", 12th Conference on Automated Deduction, 1994.
23. R. Wirfs-Brock and R. Johnson, "Surveying current research in object-oriented design", *Communications of the ACM*, 33(9), 1990.

(*) These papers can be downloaded from http://www.les.inf.puc-rio.br/~mafe.

A Framework for Persistent Object Synchronization with Active Relational Databases*

Fábio A. M. Porto[1], Maurício J. V. e Silva[2] ,Sérgio R. Carvalho[3] , Rubens N. Melo[4]
{porto[1], sergio[3], rubens[4]}@inf.puc-rio.br, mjvs@domain.com.br [2]

Laboratório TecBD, Departamento de Informática – PUC-Rio
Rua Marquês de são Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introduction

The combination of object-oriented applications with relational database systems within a client-server architecture is probably one of the most common choices for enterprise systems today [Delis+98]. Unfortunately, this combination has still many issues to overcome: the impedance between the Object Oriented (OO) model and the Relational model (RM) requires different approaches to deal with structural and behavioral clashes [Keller98]. For example, pattern languages have been proposed to bridge the existing gap between the two technologies [Keller+96, Silva+97].

In [Porto+98], we discussed this issue with respect to the dynamic behavior of objects. We proposed to represent object states and transitions through database tables and triggers. However, the action part of server rules can, in many cases, change the state of stored objects. The problem of representing these changes in the correspondent client cached objects presents itself. The objective of this work, which extends the previous one, is to provide a framework for the automatic update of client object versions based on alterations performed on the database version of the object.

2. The Synchronization Problem

Consider an application that has been implemented with an OO programming language and an active relational DBMS. The application runs on a PC type desktop and the DBMS on a server machine. Clients start the application and request data from the DBMS loading the application business objects.

Once the data gets loaded into the client application, the user deals with it as an independent OO application. The relational implementation becomes entirely transparent [Keller+96].

The persistent objects become versions of the persistent relational data, cached at the client machine, or at the application layer environment. The reasons for creating such data versions are initially to offer OO semantics for the application data stored in a relational DBMS, and secondly to increase overall performance by splitting the application between various collaborating environments.

Considering updating applications, the object versions in each client and within a server (that can be distributed) can be modified by user actions in the client environment and by active behavior in the DBMS, through the stimulation of pre-defined *triggers and stored procedures* [Widom+96, Porto+98]. Thus, in the presence of either application side or database side updates, object versions cached in client environments become out of date, so-called *stale cached data* [Franklin+97].

With increasing numbers of application clients and application update rates, the lack of synchronization within object versions may turnout to be highly painful. Client data lag far behind their persistent versions. In such a scenario, a great number of transactions would need to be completely re-submitted.

3. The Object Synchronization Framework

Considering the above, update-intensive applications could benefit from a proposal that minimizes the time lag between application object updates, while providing for the resolution of the impedance mismatch between the two models. In this section we propose an architecture, based on a replication strategy, to manage the synchronization between client cached objects and RDBMS table updates.

Figure 1 presents the class diagram, showing the classes and relationships proposed in our object synchronization framework. The *Application, PersistentApplication* (PA) and *Transaction* classes are structured in ways very similar to patterns proposed in [Keller+96, Keller98, Silva+97]. The service classes *ApplicationLog, CopyManagerClient* and *CopyManagerServer* provide the functionality for implementing the synchronization we propose.

---

* An extended version of this paper was submitted to the European Conference on Object-Oriented Programming

1 This work was partially supported by CNPq and TecBD Laboratory

The auxiliary tables *PersistentTables, UserTransaction* and *ProcedureUpdate* are used as a communication broker between the server DBMS procedures and the client procedures.

The client environment is composed of the classes *Application, PersistentApplication, ApplicationLog, CopyManagerClient* (CMC) and *Transaction*. The *CopyManagerServer* (CMS) models a singleton multi-threaded server in a 3-tier environment, serving all RDBMS clients.

The *stored_procedures* and *triggers* compose the behavior encapsulated within the DBMS. They implement constraints' validation, business logic and dynamic object behavior [Porto+98]. During their execution, database data might be updated, motivating the actions to be controled within the proposed framework.

The framework frozen spots are composed by the service classes *CopyManagerClient, CopyManagerServer, ApplicationLog, Transaction and DynamicTransaction.* The Hot-spots are the domain Application classes and their correspondent Physical View that inherit from the classes *ApplicationClass* and *PersistentApplication*. In this way, the domain specific classes can be defined exclusively concerning the application domain. The persistent behavior is acquired from their superclasses.

3.1. The Architecture Behavior Scenario

The PA is a Singleton [Gamma+95] responsible for the mappings between object views and the relational tables. All communications involving object versions and the DBMS tables pass through the PA object, which acts as a broker. The type of structural mappings that may be executed by the PA are those presented in [Keller+96]. They represent the main OO structural constructs like: class, inheritance hierarchy, association etc.. It is important to note that not all object views might be translated into tables. In special, non updateable views [Silberchatz+96, Elmasri+94] like the ones containing aggregated values do not map into tables modeled over the analytical data.
An instance of the framework behaves by executing operations under the control of transactions that respect the ACID properties guaranteed by the *Transaction* object. After having been associated to a transaction identifier, the application uses it in its communication with the CMC object. All information interchange references the transaction identifier, so that in the advent of a commit, it becomes possible to identify which tables had their state changed during the transaction processing.

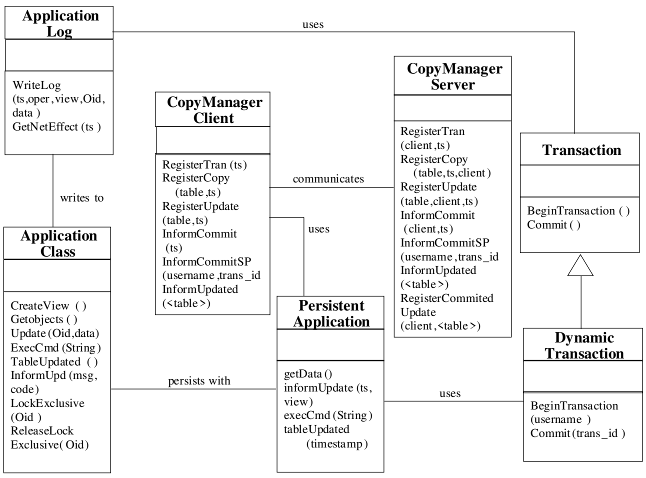


Figure 1: Object Synchronization Architecture Class Diagram

The application execution in the client side progress with almost complete independence from the DBMS side. Once the application requires an object, the PA object requests the proper tuples from the RDBMS and composes the object's view corresponding to the *Application* object for the instantiation process. The application objects inherit from their corresponding Application class the attribute *timestamp.* During the instantiation process, they receive the value of the oldest *timestamp* attribute of the corresponding tuples composing them. This attribute will be used, during the DBMS transaction commit process, to validate the consistency of the cached version versus the DBMS version.

The CMC object is informed about the objects loaded at the client, and forwards the information to the CMS object.
The user updates are executed on the *Application* objects and registered in the *ApplicationLog*, using a write-ahead policy [Gray+93]. Note that DBMS data are not locked by client updates.
The synchronization process begins by informing the correspondent PA object that the object has being changed. Considering the OO/Relational mapping, the PA object identifies the corresponding tables structurally associated with the updated object. The following step is to inform the CMC object of the tables versions updated within the current transaction.

When the user decides to commit the transaction, it invokes the corresponding *Transaction* method. The commit process begins by identifying the update operations executed during the client transaction. We obtain that by demanding the *ApplicationLog* object to provide the net effect [Widom+96] of the operations executed during the period. The operations over objects will need to be translated by the PA object into relational operation over tables. Once the set of operation is formulated, the *Transaction* objects attempts to execute them within a single DBMS transaction. During the updates of the relational information, the corresponding client objects are locked, guaranteeing a consistent synchronization of views.

The *Transaction* object waits for the DBMS return-code of the commit operation. Following a successful return, it informs the CMC object of the transaction commit. The message to the CMC dispatches the process of informing participant clients of a new version of data, stored at the database, of which they have an stale version.

The first step in the process is to inform the CMS of the committed transaction. It does that by invoking the method *InformCommit* with the parameters identifying the client and the transaction that have committed. The message prompts the CMS object for broadcasting the update message to all clients registered for the updated objects.

In this scenario, data may also be changed by DBMS server procedures. In particular, we propose that the implementation of object dynamic behavior be done through database triggers and stored procedures. These server side objects may, as part of their code, update data which have been cached into client environments. So after having terminated their execution, the result might be of changing the persistent data version of data being observed by client cached versions. In order to inform the registered clients that their version of the data is no longer valid, the server procedures have their code extended.

4. Conclusions

This work presents a proposal which attends a very common problem in the industry today, that is the development of applications using a client-server architecture where the client environment are modeled based on an OO semantics and the server side on reliable active relational database systems. We emphasize in the aspect of implementing the object dynamic behavior through database triggers and the maintenance of client cached object synchronization in the advent of server tables updates.

In this context, we present a framework composed of synchronization services providing the frozen spots and domain super-classes that provide the complementary behavior for domain specific classes.

References

[Buschmann+96] F. Buschmann, R. Meunier, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns,* John Wiley & Sons, 1996.

[Carey+91] M. Carey, M. Franklin, M. Livny, E.J. Shekita, "Data Caching tradeoffs in client-server DBMS architectures", *Proceedings of the ACM SIGMOD Conference on Management of Data,* Minneapolis, MI, May, 1991, Pages 357-366.

[Delis+98] A. Delis, N. Roussopoulus, "Techniques for Update Handling in the Enhanced Client-Server DBMS', *IEEE Transactions on Knowledge and Data Engineering,* Vol. 10, No. 3, May/June 1988.

[Duhl96] J. Duhl, "Integrating Objects with Relational Data, " In *Object-Magazine*, SIGS Publication, March 1996, pp. 89-90.

[Elmasri+94] A. Elmasri, S.Navathe, "Fundamentals of Database Systems", Addison-Wesley, second ed., 1994.

[Franklin+97] M. Franklin, M. Carey, M. Livny, "Transactional Client-Server Cache Consistency: Alternatives and Performance", *ACM Transactions on Database Systems*, vol.22, No.3, September 1997, Pages 315-363.

[Gamma+95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Gonçalves+98] T. Gonçalves, A Rito, "Passive Replicator: A Design Pattern for Object Replication", (EuroPLoP 97), Munich, Germany, July 1997.

[Gray+93] J. Gray, A. Reuter, "Transaction Processing: Concepts and Techniques", Morgan kauffmann, 1993.

[Hanson+98] E. Hanson, I. Chen, R. Dastur, K. Engel, V. Ramaswamy, W. Tan, C. Xu, "A flexible and recoverable client/server database event notification system", *The VLDB Journal*, 7:12-24, 1998.

[Hirschfeld96] R. Hirschfeld, "Three-Tier Distribution Architecture," In *Pattern Languages of Programs (PloP),* Monticello, Illinois, 1996.

[Keating+95] G. Keating, J. Thomas, "The Winning Combination: Object/Relational Solutions," In *Object-Magazine*, SIGS Publication, September 1995, pp. 64-67.

[Keller+93] A. Keller, R. Jensen, and S. Agarwal, "Persistence Software: Bridging Object-Oriented Programming and Relational Databases," In *ACM SIGMOD*, May 1993.

[Keller+96] W. Keller, and J. Coldewey, "Relational Database Access Layer," In *Pattern Languages of Programs (PloP),* Monticello, Illinois, 1996.

[Keller98] W. Keller, "Object/Relational Access Layers – A Roadmap, Missing Links and More Patterns", Europlop98, Bad Irsee, Germany, 1998.

[Porto+98] F. A. M. Porto, M. Vianna e Silva, S. R. Carvalho, "Object Life-Cycles in Active Relational Databases". TOOLS-USA, Santa Barbara, CA, August 1998.

[Silberchatz+96] A. Silberchatz, H. Korth, S.Sudarshan, "Database System Concepts", M[c] Graw Hill, third ed., 1996.

[Silva+97] M. Vianna e Silva, S. Carvalho, J. Kapson, Patterns for Layered Object Oriented Applications, Proceedings Second European Conference on Pattern Languages of Program Design (EuroPLoP 97), Munich, Germany, July 1997, pp. 85-94.

[Widom+96] J. Widom, S. Ceri, "Active Database Systems: Triggers and Rules for Advanced Database", Addison Wesley, 1996.

Um *Framework* para Implementação na WWW de Aplicações Hipermídia modeladas com OOHDM

Andréa Miranda Pizzol
pizzol@telemidia.puc-rio.br

1. Introdução

O desenvolvimento de uma aplicação multimídia/hipermídia compreende duas etapas principais. A *primeira* consiste na especificação da aplicação através de um modelo hipermídia (por exemplo, OOHDM), enquanto a *segunda* etapa consiste na implementação, em alguma linguagem, do que foi especificado anteriormente. As linguagens de marcação[2] vêm sendo amplamente utilizadas no desenvolvimento de aplicações hipermídia. Neste trabalho será considerada a linguagem de marcação HTML (*HyperText Markup Language*), usada para criar documentos de hipertexto que sejam independentes de plataforma, isto é, que sejam portáveis de uma plataforma para outra. Esta característica a tornou amplamente utilizada como formato para hiperdocumentos disponibilizados na WWW, formato esse reconhecido pelos *browsers* existentes.
Em aplicações mais complexas, deve-se considerar as limitações das linguagens de marcação para representar as primitivas básicas de modelagem hipermídia. Neste contexto, as linguagens Java e JavaScript estão sendo introduzidas a fim de suprir essas limitações, sendo possível inserir *applets* Java e *scripts* JavaScript em páginas HTML, e também executar *servlets* em servidores *Web* habilitados para Java.
A plataforma Java tem sido utilizada para escrever a parte servidora da aplicação baseada na *web* pelas mesmas características que a popularizou para escrever aplicações clientes. Atualmente, para aplicações hipermídia mais simples, o desenvolvimento está sendo feito basicamente com a linguagem HTML, de uma maneira estática. Assim, as páginas HTML são todas construídas antecipadamente (armazenadas em disco) e carregadas pelo *browser* à medida que vão sendo requisitadas. Essa abordagem acarreta uma série de problemas, pelo fato de não ser possível representar em HTML as características do modelo de projeto elaborado durante a especificação da aplicação em OOHDM. Neste trabalho, as abordagens consideradas enfatizam a integração da linguagem Java como base das propostas de arquiteturas para implementação na *web*, que visam dar suporte às aplicações hipermídia modeladas com OOHDM. O número crescente de aplicações modeladas com essa metodologia, e disponibilizadas na WWW, motivou também a construção de um *framework* para este domínio de aplicação, com o propósito de obter um desenvolvimento mais eficiente e otimizado dessas aplicações, através da reutilização tanto de código quanto de projeto.

2. Objetivos e Contribuições

Este trabalho propõe a construção de um *framework* para implementação de aplicações hipermídia projetadas com a metodologia OOHDM e disponibilizadas no ambiente da WWW, utilizando as facilidades e o poder da linguagem orientada a objetos Java.
O trabalho tem como objetivos principais:

- especificar e implementar um *framework* de aplicação para implementação na WWW de aplicações hipermídia modeladas com OOHDM.
- implementar este *framework* como uma biblioteca de classes em Java, que representa as primitivas de modelagem básicas do OOHDM.
- apresentar abordagens de implementação em Java para o desenvolvimento de aplicações OOHDM na *web*.
- propor uma arquitetura para implementação em Java, sobre o qual será provido o *framework* de aplicação a ser especificado.

As principais contribuições deste trabalho na área de projeto hipermídia WWW são:

- estudo de abordagens para implementação na WWW de aplicações hipermídia modeladas com OOHDM, considerando a integração de Java tanto do lado cliente como do lado servidor;
- definição de uma arquitetura para implementação em Java que permita soluções independentes de *hardware* e *software* (*browser* e protocolo), através da utilização de *servlets* Java que executam no servidor WWW;
- construção de um *framework* de aplicação orientado a objetos, implementado sobre a arquitetura em Java proposta, que possibilita agilizar o processo de desenvolvimento de aplicações hipermídia modeladas com OOHDM na WWW, através do reuso tanto de código como de projeto.

[2] Por *linguagem de marcação* compreende-se um conjunto de convenções de marcação usadas em conjunto para formatar textos. Uma linguagem de marcação deve especificar que marca é permitida, que marca é requerida, como uma marca é distinguida do texto, e o que a marca significa. Exemplos de linguagens de marcação: SGML, HTML, HyTime.

3. Classes do *framework*

O domínio do *framework* de aplicação desenvolvido neste trabalho, denominado OOHDM-Java, compreende as aplicações hipermídia modeladas com o método OOHDM e disponibilizadas na WWW.
Este *framework* consiste de um conjunto de classes abstratas e concretas, que represetam os componentes do *framework*, organizadas em uma hierarquia de classes e agregações, conforme ilustrado na Figura 1. Essas classes devem ser customizadas para a construção de aplicações específicas. A estrutura básica do *framework*, ou seja, as classes que o compõe e os relacionamentos entre elas, representa as primitivas básicas de modelagem hipermídia presentes no método de projeto hipermídia OOHDM.
Na especificação do *framework* procurou-se aplicar *design patterns* de construção, tal como o *pattern Observer* [Gamma95], a fim de reusar *design* (projeto), facilitar a implementação e também a manutenção do *framework*. Na Figura 1, apresentam-se as classes (componentes) do *framework* e representam-se os meta-*patterns* utilizados na especificação de algumas relações entre os seus componentes. Maiores detalhes do *framework* e da arquitetura em Java sobre o qual foi implementado podem ser encontrados em [Pizzol98].

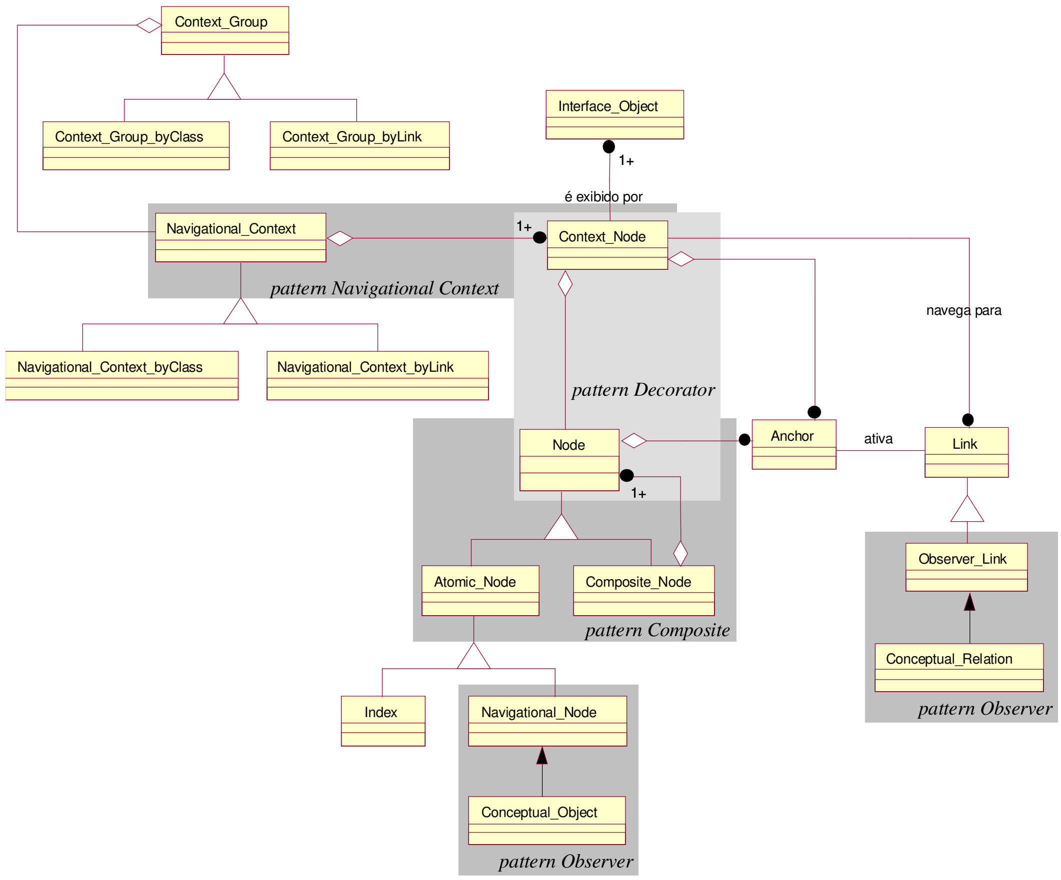


Figura 1 – Classes do *framework* OOHDM-Java

Referências

[Pizzol98] Pizzol, Andréa M., "Um Framework para Implementação na WWW de Aplicações Hipermídia modeladas com OOHDM", Dissertação de Mestrado, Departamento de Informática, Puc-Rio, Dezembro/1998.

O Paradigma de Reconhecimento/Geração de Planos

Angelo E. M. Ciarlini and Antonio L. Furtado
{angelo, furtado}@inf.puc-rio.br

Departamento de Informática
Pontifícia Universidade Católica do R.J.
22.453-900 Rio de Janeiro, Brasil

1. Introdução

Os fatos de um banco de dados podem ser convenientemente expressos sob a forma de *predicados*. O *estado* de um banco de dados é o conjunto de todas as instâncias de predicado que valem em um certo instante. Ele fornece uma descrição do micro-mundo modelado pelo banco de dados. As *ações* que ocorrem nesse micro-mundo podem ser denotadas, no nível conceitual, por operações que mapeiam estados do banco de dados. Uma operação pode ser especificada, segundo um formalismo tipo STRIPS [FN], através da declaração das suas *pré-* e *pós-condições*, que caracterizam os estados antes e depois da operação. As pré-condições estabelecem requisitos que devem valer antes da execução e as pós-condições expressam efeitos das operações. Na definição de operações, assume-se que tudo mais que não é mencionado pelas pré- e pós-condições e que valia antes da operação continuará valendo após a sua execução. Trata-se assim o problema de "Frame".

Uma seqüência de operações iterrelacionadas define o que chamamos de *enredo de narrativa*, o qual pode ser obtido a partir de consultas ao "log" de um banco de dados. Tais consultas especificariam, por exemplo, os objetos envolvidos nas operações e o período correspondente.

Ao examinarmos os enredos de narrativas verificamos que elas envolvem os *objetivos* dos diversos agentes que as executam. É, portanto, útil distingüir os *efeitos primários* de uma operação de outros, menos importantes, aparentemente inócuos.

Algumas vezes, para se atingir um objetivo, é necessária a execução de diversas operações, seja porque o objetivo é uma combinação dos efeitos primários de várias operações seja porque a execução de uma operação exige a execução de uma outra anteriormente para que suas pré-condições sejam atendidas. Esse conjunto de operações parcialmente ordenadas constitui o que denominamos *planos*. A distinção entre planos e operações desaparece se introduzimos a idéia de operações complexas, sejam elas definidas por *composição* (duas ou mais operação em conjunto – hierarquia "part-of") ou generalização (mais de uma alternativa - "hierarquia is-a").

Uma biblioteca de planos típicos pode ser construída através da estruturação de operações simples e complexas, segundo hierarquias de composição e generalização. A existência de bibliotecas desse tipo nos habilita para a utilização de mecanismos de reconhecimento de planos, que descubram o plano (e portanto os objetivos) de um certo agente comparando um conjunto de observações de seu comportamento com a biblioteca. Como parte do nosso projeto de pesquisa corrente, implementamos um protótipo que integra Reconhecimento e Geração de planos. Quando o nosso protótipo reconhece um plano, ele verifica se tal plano é aplicável. Se existem obstáculos, o plano reconhecido pode tanto ser adaptado ou pode ser gerado um novo plano para atingir os mesmos objetivos. A parte de reconhecimento de planos de nosso protótipo baseou-se no algoritmo de Kautz [Ka] e a parte de planejamento em técnicas desenvolvidas no projeto Abtweak [Wo].

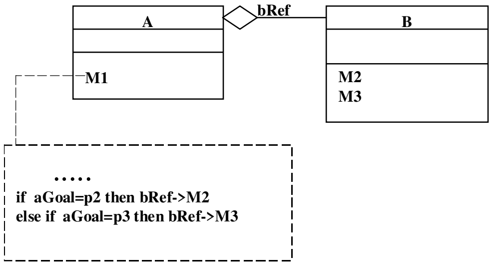
Uma das etapas importantes para a utilização do paradigma de utilização integrada de reconhecimento e geração de planos é a definição da biblioteca. Para automatizar (pelo menos parcialmente) essa etapa pode-se utilizar o próprio gerador de planos, uma vez que as operações básicas tenham sido definidas. Outra alternativa é deixar que um banco de dados seja utilizado por um certo tempo, gravando-se as operações executadas em um "log". Extraindo-se um certo número de narrativas e comparando-as, utilizando, por exemplo, conceitos como o de *generalização mais específica* [Fu], torna-se possível criar operações complexas que formarão a biblioteca.

O nosso protótipo tem se mostrado suficientemente poderoso para tratar tanto narrativas simples de bancos de dados quanto narrativas bastante intrincadas de contos de fadas. Em relação a esses últimos, cabe ressaltar a conexão de nosso trabalho com os conceitos desenvolvidos pelo pesquisador russo Vladimir Propp, segundo o qual os contos de fadas poderiam ser *estruturados* como seqüências de funções (e.g. "vilania", "combate", "retorno"), atribuídas a *esferas de ações* (personagens com diferentes papéis, tais como "herói" e "vilão"). Para os contos de fadas, ele propôs um conjunto de 31 funções divididas em 7 esferas de ação. O nosso paradigma procurou incorporar **semântica** e **pragmática** às estruturas proppianas através da definição de pré- e pós-condições e da identificação dos objetivos dos personagens.

2. Planos e Frameworks

Se o conceito de *operação* pode ser introduzido tanto no mundo dos sistemas de bancos de dados quanto no de narrativas literárias fictícias, é de se esperar que ele seja aplicável a outros contextos tais como o dos programas de computador feitos com base em "frameworks". No mundo dos programas de computador, as operações tomam a forma de *comandos* ou, em um nível mais alto, nos ambientes orientados a objetos, de *métodos.* De forma similar, as classes também podem ser associadas às citadas esferas de ação dos gêneros literários.

O primeiro passo para se ver os métodos dentro do nosso paradigma é defini-los em termos dos seus argumentos (incluindo o identificador do objeto) e de suas pré- e pós-condições, que correspondem aos requisitos de entrada e saída. Uma vez definido um conjunto de métodos básicos, uma primeira utilização do planejador poderia ser feita para gerar planos que seriam comparados e utilizados tanto na criação de "frameworks" quanto na verificação da adequação de "frameworks" já existentes. Imaginemos uma situação inicial p0 e a existência de três métodos M1 (pré-cond. p0 e pós-cond. p1), M2 (pré-cond. p1 e pós-cond. p1' $\wedge$ p2) e M3 (pré-cond. p1 e pós-cond. p1' $\wedge$ p3). Para uma situação onde o objetivo final inclua p2, o plano <M1,M2> poderia ser gerado, enquanto que em outra que inclua p3 <M1,M3> poderia ser gerado. Note-se a compatibilidade de ambos os planos com (entre outros) o "framework" da figura 1.



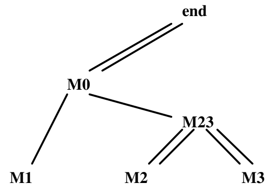


Fig.1: Uma Estrutura de "Framework" Fig.2: Bibl. de Planos Típicos (Oper. Complexas)

A criação automática de uma biblioteca de planos típicos poderia dar indicações importantes para a construção de bons "frameworks". Examinando-se o "log" de execuções de programas que resolvem um certo tipo de problema, utilizando um mesmo conjunto de métodos básicos, podemos identificar "*traces*", os quais nada mais são, nesse caso, que os nossos enredos de narrativas. A identificação de padrões de "traces" possibilita a descoberta de métodos complexos, utilizando-se composição e generalização. Enquanto que a generalização de métodos nos mostra possíveis hierarquias de classe, a composição nos possibilita identificar o encadeamento das chamadas de execução.dos métodos. A construção de uma biblioteca com base nos traces <M1,M2> e <M1,M3>, ao identificar que p1' é o efeito primário tanto de M2 quanto de M3, criaria M23 como a generalização de M2 e M3. Além disso o método complexo M0 seria criado como a composição de M1 com M23. A figura 2 mostra a biblioteca gerada. Os arcos duplos representam generalizações e os simples, composições.

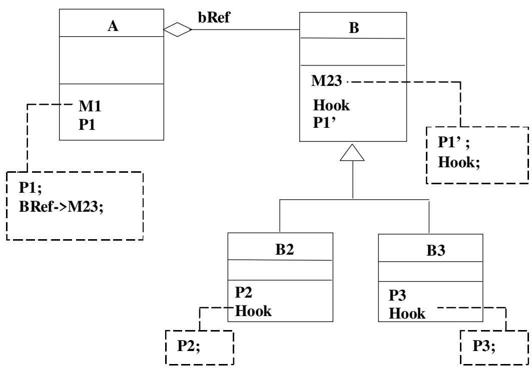


Fig. 3: Esquema de "Framework" Alternativo

Essa biblioteca sugere um esquema alternativo de "framework" (figura 3) onde a classe B é dividida em duas subclasses e M23 aparece como um método genérico ("hot spot") [Pre], que faz a parte comum do processamento nos dois casos e chama um método "hook" para atingir o efeito secundário desejado (p2 ou p3). Esse esquema faz sentido se esse objetivo secundário pode ser relacionado diretamente a características do objeto da classe B referenciado pela variável de instância bRef. Outra alternativa é criar "frameworks" separados, o que é conveniente se o efeito secundário depende do tipo de aplicação.

Em geral, o estudo proposto para o comportamento dos métodos segundo o nosso paradigma pode não ser suficiente para a escolha de uma certa alternativa de "framework", o que siginifica que outras análises complementares baseadas em "*patterns*" [Pre] e/ou "*viewpoints*" [FHL] tenham que ser feitas.

Por outro lado, o paradigma permanece aplicável depois que o projeto do "framework" está completo para o propósito de teste. Por exemplo, assumindo-se que um dos "frameworks" mostrados anteriormente foi adotado, ao identificar a intenção do usuário de definir a execução de M1 como parte da solução de um certo problema, o protótipo pode detectar a intenção de executar o plano M0, associado ao "framework". Nesse caso, o protótipo estará apto a fazer o seguinte:

- indicar que não há obstáculos para a execução dos diversos passos de M0, o que daria indicações da adequação do “framework”;
- detectar que p1 já vale e, portanto, M1 não precisa ser executado, o que sugeriria um reprojeto do “framework” para possibilitar a utilização de uma versão mais curta e eficiente;
- detectar que p0 não vale. Como a intenção de executar M0 implica na intenção de se atingir um estado onde p1’ seja válido, o algoritmo tentaria gerar um novo plano sem incluir M1. A falha na geração desse plano poderia indicar a necessidade de se rever o conjunto de métodos definidos.

Como ilustrado pelos exemplos, o reconhecimento de planos – com ou sem obstáculos – indica a possibilidades de reuso de “frameworks” já existentes. A geração de planos fornece alternativas para a adaptação ou criação de “frameworks”. Estudos mais aprofundados sobre a aplicação do paradigma proposto no mundo dos programas de computação, especialmente no de “frameworks” orientados a objetos, podem levar a métodos úteis que ajudem no projeto e manutenção de produtos de “software” altamente adaptativos.

Agradecimento: Somos gratos a Fábio A. M. Porto pelos comentários e sugestões que nos auxiliaram muito na preparação desse trabalho.

Referências

[Aa] Antti Aarne. *The Types of the Folktale: A Classification and Bibliography*. Translated and enlarged by Stith Thompson, FF Communications, 184. Helsinki: Suomalainen Tiedeakatemia, 1964.

[Af] A. Afanas'ev. *Russian Fairy Tales*. N. Guterman (trans.). New York: Pantheon Books, 1945.

[CF] M. A. Casanova and A. L. Furtado. “An information system environment based on plan generation”. in Proc. of the *International Working Conference on Cooperative Knowledge Based Systems*. Univ. Keele, England, 1990.

[FCa] A. L. Furtado and M. A. Casanova. "Plan and schedule generation over temporal databases". in Proc. of the *9th International Conference on the Entity-Relationship Approach*, 1990.

[FCi] A.L. Furtado and A. E. M. Ciarlini. "Plots of narratives over temporal databases". in *Proc. of the 8th International Workshop on Database and Expert Systems Applications*. R. R. Wagner (ed.). IEEE Computer Society, 1997.

[FHL] M. F. M. C. Fontoura, E. H. Haeusler, and C. J. P. Lucena. *A framework design and instantiation method based on viewpoints*. Technical report MCC35/98, Dept. Informatica, PUC-Rio, 1998.

[FN] R. E. Fikes and N. J. Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving". *Artificial Intelligence* , 2(3-4), 1971.

[Fu] A.L. Furtado. "Analogy by generalization and the quest of the grail". *ACM/SIGPLAN Notices*, 27, 1, 1992.

[FuN] A. L. Furtado and E. J. Neuhold. *Fornal Techniques for Data Base Design*. Berlin: Springer-Verlag, 1986.

[GH] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading: Addison Wesley Publishing Company, 1995.

[Gr] A.J. Greimas. *Sémantique Structurale*. Paris: Librairie Larousse, 1966.

[Jol] A. Jolles. *Formes Simples.*. Paris: Éditions du Seuil, 1972.

[Ka] H. A. Kautz. “A formal theory of plan recognition and its implementation”, in *Reasoning about Plans*. J. F. Allen et al (eds.). San Mateo: Morgan Kaufmann, 1991.

[MC] C.J. Matheus, P.K. Chan. and G. Piatesky-Shapiro. "Systems for knowledge discovery in databases". *IEEE Transactions on Knowledge and Data Engineering*, 5, 6, 1993.

[Oz] G. Ozsoyoglu. and R.T. Snodgrass. "Temporal and real-time databases: a survey". *IEEE Transaction on Knowledge and Data Engineering*, 7, 4, 1995.

[Pre] W. Pree. Design Patterns for Object-Oriented Software Development. Wokingham: Addison-Wesley, 1995.

[Pro] V. Propp. *Morphology of the Folktale*. Laurence Scott (trans.). Austin: University of Texas Press, 1968.

[Ru] D.E. Rumelhart. "Notes on a schema for stories". in *Representation and Understanding - Studies in Cognitive Science*. D. G. Bobrow and A. Collins (eds.). New York: Academic Press, 1975.

[SA] R. C. Schank and R. P. Abelson. "Scripts, plans and knowledge". in Proc. of the *Fourth International Joint Conference on Artificial Intelligence*, 1975.

[Sh] A. Sheth. "From contemporary workflow process automation to adaptive and dynamic work activity coordination and collaboration". in Proc. of the *8th International Workshop on Database and Expert Systems Applications,* September, 1997.

[Wa] D. H. D. Warren. "Generating conditional plans and programs". in Proc. of the *AISB Summer Conference*, pp. 344-354, Edinburgh, 1976.

[Wi] R. Wilensky. "Points: a theory of the structure of stories in memory". in *Readings in Natural Language Processing*. B. J. Grosz, K. S. Jones and B. L. Webber (eds.), San Mateo: Morgan Kaufmann, 1986.

[Wo] S. G. Woods. *An Implementation and Evaluation of a Hierarchical Non-linear Planner*. Master's thesis. Computer Science Department of the University of Waterloo, 1991.

[WW] R. Wirfs-Brock, B. Wilkerson and L. Wiener. *Designing Object-Oriented Software*. Englewood Cliffs: Prentice-Hall, 1990.

Um Meta-Ambiente para Suporte a Construção Sistemática de Frameworks

Sergio Crespo* e Carlos José P. de Lucena
{crespo, lucena}@inf.puc-rio.br

Laboratório de Engenharia de Software, Departamento de Informática – PUC-Rio
Rua marquês de são Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introdução

Este artigo apresenta um ambiente que dá suporte a um método de *design* para software orientado a objeto baseado na WEB, que combina *frameworks* e *viewpoints*. O uso do conceito de viewpoints e das relações hot-spot e views-a permitem ao projetista uma maior facilidade na identificação dos pontos de flexibilização (*hot-spots*) e do núcleo da estrutura do *framework*.

2. Meta-ambiente

O ambiente encontra-se disponível para teste na seguinte URL: http://rush.les.inf.puc-rio.br/crespo/qualify. Pode-se utilizar os Browsers da Microsoft ou da Netscape para interação. A seguir, será apresentado o meta-ambiente onde uma arquitetura do ambiente é demonstrada, focalizando seus espaços e atores no processo de criação de frameworks. Finalizando, são identificados os passos necessários para a continuidade do ambiente e as conclusões. Informações mais detalhadas sobre este trabalho podem ser obtidas em [1][2].
O ftp dos artigo completo e demais referências pode ser obtido em http://www.les.inf.puc-rio.br/~crespo

2.1. Arquitetura do Ambiente

A arquitetura do ambiente compreende três espaços principais:
- Espaço de Viewpoints
- Espaço de Frameworks baseados em template hook e
- Espaço de Framework baseado em Pattern.

Esses espaços são utilizados por diferentes atores que podem ter acesso ao sistema. A seguir vamos descrever os espaços e seus atores.

2.1.1. espaço de viewpoints

Neste espaço são criadas todas as classes para um determinado domínio de aplicação. É utilizada uma ferramenta "Viewpoint Text Tool" que permite a criação e a reutilização de classes de maneira fácil e intuitiva, além de possibilitar a definição de relacionamentos entre elas. Neste espaço podem estar atuando vários atores sob diferentes perspectivas através da WEB, utilizando um browser como sua interface. Note-se que o trabalho destes atores não é dependente de tempo ou de qualquer sincronismo entre eles. O ambiente permite que o trabalho seja realizado de forma assíncrona. Os atores, para obterem uma aplicação executável, deverão ainda passar pelos outros dois espaços. A interação entre os atores neste espaço é bem forte, pois todas as classes criadas podem ser compartilhadas por todos. Uma vez definidas as perspectivas, os atores estarão aptos a passarem para outro espaço, o *espaço de frameworks baseados em template hook* que é gerado pelo processo de unificação de viewpoints.

2.1.2. espaço de frameworks baseados em template hook

Este espaço é resultante do processo de unificação dos viewpoints e nele podemos encontrar o kernel do framework e seus hot-spots. Pode receber dois tipos de atores: os atores que definiram os conjuntos iniciais de viewpoints e um outro grupo de atores que não participou desta fase, mas pode aproveitar todo o trabalho realizado anteriormente. A partir deste ponto, inicia-se a definição das propriedades de flexibilização, utilizando uma ferramenta chamada Hot-Spot Card Tool. O ambiente separa os contextos destes atores de forma a não permitir que atores interfiram no trabalho de outros atores. Podem ser definidos vários tipos de propriedades de relacionamentos para a mesma relação. Uma vez caracterizados todos os relacionamentos ocorre uma instanciação dos hot-spot gerando o próximo espaço chamado de *espaço de framework baseado em pattern*.

* Sérgio Crespo é Professor Assistente II do departamento de informática do Centro 6 na Universidade do Vale do Rio dos Sinos - Unisinos-RS.

2.1.3. espaço de framework baseado em Pattern

O *espaço de framework baseado em pattern* é um espaço que pode contar com três tipos de atores: atores que são provenientes do primeiro espaço, atores provenientes do segundo espaço e atores que somente atuam no terceiro espaço de forma a gerar aplicações a partir do framework existente aqui e que está composto agora, não mais por relacionamentos tipo hot-spot mas por patterns apropriados que implementam tais relacionamentos. Aqui todos possuem um framework para ser instanciado usando algum tipo de DSL (Lingugem Específica de Domínio) que facilite a implementação das partes que necessitam de customização. O ambiente realiza uma checagem entre o que foi definido na seção de projeto e no que foi implementado durante a instanciação do Framework de modo a não permitir inconsistência. Esta verificação é feita pela ferramenta "Verificador de Integridade Lógica e de Implementação".

2.2. Unificação em diferentes plataformas

Estamos analisando o suporte que o ambiente irá proporcionar para a unificação de viewpoints em diferentes plataformas. Para a unificação de plataformas diferentes estamos utilizando o relacionamento Views-a [3][4] [5] [6] [7], que nos permitirá "colar" um Design Patterns para que as duas plataformas possam ser unificadas. Após esta unificação, inicia-se o processo de geração de código propriamente dito. Neste processo será utilizada a máquina DRACO-PUC[8] para as transformações necessárias na geração final do código executável.

3. Conclusões

A utilização de viewpoints e da relação de flexibilização hot-spot nos permite uma rápida e sistemática construção de frameworks. A identificação das classes comuns aos viewpoints formam o kernel e os pontos de flexibilização tornam-se facilmente visíveis. O método proposto utiliza um ambiente baseado na WEB que facilita a cooperação entre designers na construção de frameworks pois permite que usuários localizados em locais remotos trabalharem de forma harmônica sem que isso gere impactos em suas especificações. O ambiente utiliza o conceito de meta-dados para armazenar informações sobre todas as suas classes e objetos em formato XML tornando, desta maneira, toda a sua estrutura de armazenamento inter-operável e compatível com o padrão IMS. Esta característica de armazenamento de classes e objetos permite alto grau de reutilização, não só em termos de viewpoints e suas classes, mas também de conteúdo gerado após a implementação da aplicação em alguma linguagem de programação. É possível, ainda, realizar a reutilização de viewpoints e de frameworks, pois estes possuem também uma meta-informação em formato XML.
O ambiente encontra-se em teste para as seguintes etapas:

- Gerência, onde podem ser definidos usuários, perspectivas e domínios;
- Manipulação de viewpoints, através da ferramenta "*ViewPoint-Text-Tool*"; e
- A unificação dos viewpoints.

As contribuições deste trabalho compreendem:

- A viabilização prática do método proposto [9][10][11][12] e sua extensão através da definição de meta-dados para as classes, viewpoints e frameworks desenvolvidos;
- O trabalho compartilhado na definição de frameworks que o ambiente oferece;
- O suporte à identificação do kernel e dos pontos de flexibilização do framework (tarefa nem sempre trivial, dependendo do número de classes envolvidas);
- A definição e criação de uma ferramenta que faça a verificação de design e de implementação (Verificador de Integridade Lógica e de Programação);
- A existência de um dicionário de métodos, informando para o usuário: onde o método é utilizado, que classes o utilizam, sua interface, autor, implementação;
- A unificação de design com plataformas diferentes através do uso da relação Views-a;
- A definição de meta-dados para as classes, viewpoints e frameworks desenvolvidos. Isto proporciona a reutilização em grande escala de componentes em determinados viewpoints, ou mesmo de viewpoints inteiros (tipo pacote) e, em última instância, de frameworks completos.
- A possibilidade de uma instanciação direta de um modelo baseado em Patterns sem que o usuário, neste caso, tenha participado dos processos anteriores de criação/definição.

Referências

1. S.Crespo, C. J. Lucena, A. v. Staa, "Um Meta-Ambiente para Suporte a Construção Sistemática de Frameworks " , MCC39/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, Dezembro 1998.

2. S. Crespo, C. J. Lucena e Arndt v. Staa, " FrameConstructor: uma ferramenta para Suporte a Construção Sistemática de Frameworks", MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, September 1998.

3. P.S.C. Alencar, D.D. Cowan, S. Crespo, M. F. Fontoura, and C.J.P. Lucena, “A Viewpoint-based Design Method: A Case Study in Frameworks for Web-based Education”, CS-98, Computer Science Department, University of Waterloo, 1998.

8. J. C. S. P. Leite, M. Sant’anna, and F. G. Freitas, “Draco-PUC: a Technology Assembly for Domain Oriented Software Development”; Proceedings of the 3rd IEEE International Conference of Software Reuse, 1994.

4. D. Cowan and C. J. P. Lucena, “Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse”, IEEE Transactions on Software Engineering, March 1995.

5. P. Alencar, D. Cowan, S. Crespo, M. F. Fontoura, and C. J. Lucena, "Pragmatic Issues Behind Framework Design", Workshop #12 - Pragmatic Issues in Using Frameworks Implications for Framework Design, ACM Object-oriented Programming, Systems, Languages, and Applications (OOPSLA’98), Vancouver, Canada, 1998 (Web publication).

6. S. Crespo, M. F. Fontoura,and C. J. Lucena, “A Framework Development Method Using Viewpoints and the Views-a Relationship in the OO Design", MCC13/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, April 1998.

7. P. Alencar, D. Cowan, T. Nelson, M.F. Fontoura, and C. J. Lucena, "Viewpoints and Frameworks in Component-Based Design", in book "Object-Oriented Application Frameworks", M. Fayad, D. Schmidt (editors), three volumes, John-Wiley, 1998.

9. S. Crespo, M. F. Fontoura,and C. J. Lucena, “A Framework Development Method Using Viewpoints and the Views-a Relationship in the OO Design", MCC13/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, April 1998.

10. P. Alencar, D. Cowan, T. Nelson, M.F. Fontoura, and C. J. Lucena, "Viewpoints and Frameworks in Component-Based Design", in book "Object-Oriented Application Frameworks", M. Fayad, D. Schmidt (editors), three volumes, John-Wiley, 1998.

11. M. F. Fontoura, E. H. Haeusler, and C. J. Lucena, "A Framework Design and Instantiation Method Based on Viewpoints", MCC/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, August 1998.

12. M. F. Fontoura, E. H.Haeusler, and C. J. Lucena, "The Hot-Spot Relationship in OO Framework Design", MCC33/98, Monografias em Ciência da Computação, Departamento de Informática, PUC-Rio, September 1998.

Scenario Configuration Management

Karin Breitman, Julio Cesar Sampaio do Prado Leite
{karin,julio}@inf.puc-rio.br

Departamento de Informática – PUC-Rio
Rua marquês de são Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

The process of writing system specifications generates large amounts of documentation regardless of the technique employed. The scenario-based approach is no exception. Naturally descriptive, scenarios in fact contribute to the generation of a vast documentation. Contrary to prototypes that are usually discarded, there seems to be an agreement that scenarios are persistent artefacts [Sutcliffe95, Hsia94, Leite97, Rosson95, Jacobson92, Potts94, Kyng95]. As such, scenarios are liable to undergo change during the software development process. Change comes as a result of greater understanding of the software to be built [Yeh90]. It can be an addition, subtraction or refinement to the scenario contents. In either case, we observe that new versions of the scenarios are generated. During this process a massive amount of information is dealt with, placing a demand for proper configuration management techniques. In this light, scenario traceability becomes a major issue.

From an evolutionary perspective, two issues stand out. First is determining what information contained in a scenario description is worth keeping track of. We mentioned before that scenarios naturally generate a great amount of information. Is all this information relevant from a configuration management perspective? The second issue that must be considered in order to establish a scenario configuration management strategy is what kind of extra information is needed that is not usually part of a scenario. Traditional configuration management approaches such as the one proposed by Bersoff and the Requirements baseline proposed by Leite, keep record of information such as date of change, author and reason of change (rationale) [Bersoff80, Leite95, Mikkelsen97] to help keep traceability of systems.

We tackled these issues by presenting a strategy that supports scenario configuration management. The strategy proposed is independent of the scenario notation used. We provided a framework for scenario configuration management that can be customized to fit a class of scenario approaches. By instantiating a set of operations provided by the framework, users are able to define their own versioning mechanism. Versioning is defined threefold, by terms of the relevance of a scenario component to the evolutionary process; the nature of extra information that has to be elicited in order to perform configuration management, and the definition of a set of legal operations that result in new versions. Independently of the scenario notation used, the framework allows the user to decide on a different configuration management strategy for each project.

We restricted our concerns to structured scenario notations. Scenario usage such as the ones proposed by Hsia et al [Hsia94] and the narrative style [Rosson95, Kyng95] are out of scope. As a pre-requisite for being encompassed by our proposal, there should be enough structure in the scenario representation as to allow for general abstractions. Our goal is categorize scenario components into abstract types that would serve several different notations.

The experience has contributed in deepening our understanding of configuration management and scenario evolution. We have pointed out the problems related to the control of information generated by the scenario evolution process in [Leite97]. Further work enabled us to describe scenario evolution by providing a set of inter and intra scenario operations [Breitman98]. The main contribution is the organization of scenario evolution knowledge into a framework that enables users to deal with the complexities of scenario configuration management. The framework takes in consideration the use of scenarios in different development process stages and multiple viewpoints.

The framework proposed is both flexible and extensible, allowing user to tailor the configuration strategy to his or hers special needs. There is no doubt that preparing for configuration management places a load on the users. They will have to slow down and register information that would not be recorded otherwise. Implementation of a traceability strategy has been reported to increase development time and documentation costs of projects [Ramesh95]. The question is how to balance the trade off between the extra amount of work required and possible future gains.

In the future we intend to carry on with our validation process using case studies. Our agenda is to refine the proposed framework through feedback from experiences in building and using the software infrastructure necessary to provide automated support to the framework.

References

[Bersoff80] – Bersoff, E. et al. – *Software Configuration Management* – Prentice Hall, 1980.

[Breitman98] – Breitman, K.K.; Leite, J.C.S.P. – A framework for scenario evolution – in Proceedings of the International Conference on Requirements Engineering (ICRE) – pp. 214-221- Colorado Springs, USA – 1998.

[Hsia94] – Hsia, P.et al – Formal Approach to Scenario Analysis – IEEE Software, vol. 11 no. 2 – pp.33-41 – 1994.

[Jacobson92] – Jacobson, I.; Cheristerton, M.; Jonsson, P.; Overgaard, G. – Object Oriented Software Engineering : a use case driven approach – Addison Wesley – 1992.

[Kyng95] - Kyng, M. – Creating contexts for design - in Scenario-based design: envisioning work and technology in system development – pp. 85-107 - John Wiley and Sons, New York – 1995.

[Leite95]- Leite, J.C.S.P. and Oliveira, A.P. – A Client Oriented Requirements Baseline - in the *Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE'95)* – pp.108-115 - York, March 27 to 29 – IEEE Computer Society Press, 1995.

[Leite97] – Leite, J.C.S.P. et al. – Enhancing a Requirements Baseline with Scenarios - – Requirements Engineering Journal vol(2) pp. 184-198 – Springer Verlag - December, 1998.

[Mikkelsen97] – Mikkelsen, T.; Pherigo, S. – Practical Software Configuration Management – Hewlett Packard Professional Books – Prentice Hall, 1997.

[Potts94] – Potts, C.; Takahashi, K.; Anton, A. – Inquiry based requirements analysis. IEEE Software vol. 11 no. 2 – pp.21-32 – March, 1994.

[Potts98] – Potts, C. – Tutorial Material – presented during the Conference on Industrial Requirements Engineering (CEIRE'98) – London, England – 1998.

[Ramesh95] - Ramesh, B.; Stubbs, C.; Edwards, M. - Implementing Requirements Traceability: a Case Study - Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE95) - IEEE Press, pp. 89-95 - York, England, 1995.

[Rosson95] – Rosson, M.B.; Carroll, J. – Narrowing the specification implementation gap in Scenario-based design: envisioning work and technology in system development, John Wiley and Sons, New York – 1995

[Sutcliffe95] - Sutcliffe, A. - Requirements Rationales: Integrating Approaches to requirement analysis - Proceedings of the Symposium on Designing interactive systems: processes, practices, methods and techniques - ACM Press - pp.33 - 42 - Ann Arbor, USA, 1995.

[Yeh90] - Yeh, R.; Ng. P. - Software Requirements - a management perspective - System and Software Requirements Engineering - Dorfman, M. and Thayer, R. eds. - pp. 450-461 - Institute of Electrical and Electronics Engineers, Inc., 1990.

Padrões de Comunicação

Carlos Laufer, Marcelo Blois, Ricardo Choren

Laboratório de Engenharia de Software, Departamento de Informática – PUC-Rio
Rua marquês de são Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introdução

Estão surgindo atualmente na Internet diversos ambientes que têm como um de seus objetivos permitir que grupos de pessoas possam cooperar utilizando diferentes formas de interação de uma maneira integrada. Nesses ambientes existe uma carga de requisições assíncronas muito grande, bem diferente dos ambientes presenciais onde existe uma carga de requisições síncronas predominantes.

A capacidade crescente de interação proporcionada a cada pessoa demanda o desenvolvimento de mecanismos que possam auxiliar na manipulação da enorme quantidade de informações gerada a partir dessas interações.

Dentro de um ambiente para cooperação existe uma variedade de procedimentos que necessitam de uma comunicação para seu estabelecimento. Essa comunicação em muitos casos apresenta um padrão de repetição. Para cada procedimento administrativo, por exemplo, poderia ser definido um conjunto de padrões de comunicação a ser configurado. Uma instância de um ambiente para cooperação poderia ser configurada de acordo com determinados padrões de comunicação adequados a uma determinada instituição.

O objetivo do trabalho é a observação e o estudo dos processos de comunicação decorrentes das interações entre atores de ambientes de cooperação mediados por computador visando a definição de padrões de comunicação.

2. Processo de comunicação na educação à distância

O processo de comunicação de interesse nesse trabalho é o que ocorre em um ambiente de educação à distância. O conceito de educação à distância não é recente e está associado a educação de pessoas onde existe uma distância que dificulta a interação entre aqueles que seriam os consumidores do conhecimento e aqueles que seriam os transmissores do conhecimento. Em geral essa forma de educação à distância oferece uma possibilidade de interação entre os professores e os estudantes menor do que as possibilidades oferecidas pelos métodos presenciais de ensino.

Uma das diferenças entre os ambientes de ensino na Internet e as formas de ensino presencial tradicional e também os cursos anteriormente ministrados à distância é que esses ambientes na Internet oferecem novas possibilidades de interação entre os diversos participantes de um grupo. Esses novos ambientes possibilitam uma carga de interação muito mais intensa entre os diversos participantes desses ambientes. Os professores e os administradores não estão preparados para esse aumento de carga interativa.

Além disso a facilidade de disponibilização de materiais através da WWW faz com que todo um conjunto de informações, que era geralmente transmitido em forma de *broadcast* do professor para os alunos, possa ser consumido pelos estudantes independente de tempo e local e de acordo com as suas necessidades. Assim como os professores não estão preparados para o aumento de carga interativa, os estudantes não estão preparados para o aumento de suas responsabilidades na condução de seus estudos.

Outros tipos de ambientes cooperativos mediados por computador que utilizam a Internet e a WWW também introduzem essa possibilidade de interação de forma intensiva. Um computador conectado a Internet não é a mesma máquina que esse mesmo computador desconectado. O computador se aproxima cada vez mais de um dispositivo de comunicação. Claro que continua sendo uma máquina com diversas outras capacidades, mas começa a ser percebido e utilizado de forma crescente como um dispositivo que permite várias formas de comunicação.

O telefone é basicamente visto como um dispositivo de comunicação, sendo transparente para as pessoas a tecnologia e as capacidades de processamento embutidas no dispositivo que vem se tornando cada vez mais sofisticado. Na verdade o telefone permite uma interação entre dois objetos localizados em suas pontas que inicialmente eram duas pessoas. A introdução do fax permitiu a interação entre duas máquinas localizadas nas pontas da linha telefônica para a transmissão de dados.

Um computador conectado a Internet, seja através de um modem utilizando uma linha telefônica ou conectado a uma LAN, permite um conjunto muito grande de interações entre uma pessoa e diversos tipos de objetos espalhados ao redor do mundo.

3. AulaNet

AulaNet é um ambiente para administração, criação, manutenção e assistência de cursos através da Internet. Três atores estão atualmente envolvidos no ambiente AulaNet: o administrador, o professor e o aluno. Existem diversos procedimentos administrativos dentro do ambiente AulaNet como por exemplo: cadastramento de participante no ambiente AulaNet, cadastramento de docente, criação de curso, matrícula em curso, etc… Vários desses procedimentos requerem uma comunicação entre os atores do ambiente.

Como exemplo de procedimentos administrativos que poderiam ser modelados de acordo com padrões de comunicação são apresentados a seguir o cadastramento de participante no ambiente AulaNet e o cadastramento de docente.

Atualmente no AulaNet o cadastramento de participante no ambiente é realizado de forma automática e não necessita de nenhuma autorização por parte do administrador. Uma vez feito o pedido de cadastramento no ambiente o participante está automaticamente autorizado a pedir matrícula em um dos cursos oferecidos. A figura 1 ilustra um diagrama de estados representando o pedido de cadastramento no ambiente AulaNet.

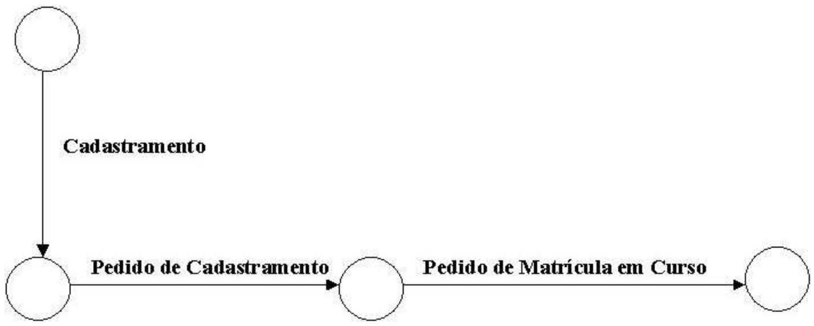


Figura 1 – Diagrama de Pedido de Cadastramento

Por outro lado o cadastramento de docente requer uma comunicação entre o administrador do ambiente e o participante que requisita a docência. Dependendo das informações trocadas nessa comunicação o administrador pode aceitar ou recusar o participante como docente, o que significa que esse participante poderá ou não criar um curso no ambiente AulaNet. A figura 2 ilustra um diagrama de estados representando o pedido de docência no ambiente AulaNet.

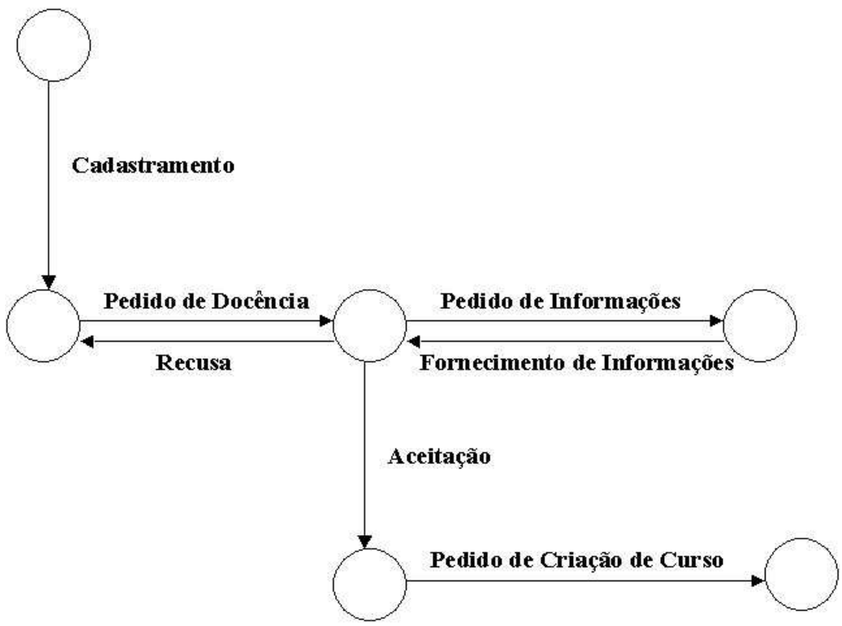


Figura 2 – Diagrama de Pedido de Docência

No exemplo apresentado podemos observar dois padrões de comunicação que poderiam ser configurados para uma instância de ambiente. Por exemplo, uma instância do ambiente AulaNet de uma determinada instituição de ensino poderia ser configurada com um procedimento para cadastramento no ambiente que necessitasse de uma negociação semelhante ao do pedido de docência.

A partir do estudo de padrões de comunicação busca-se também a definição de formas de registro das informações geradas durante a negociação eventualmente efetuada durante uma comunicação, que em diversos casos define compromissos entre os participantes dessa comunicação. Dessa forma poderíamos recuperar o histórico dos processos associados a um determinado padrão de comunicação e as opções e justificativas para a tomada de decisões dentro de um ambiente para cooperação mediado por computador.

Um Framework para Análise de Programas

Leonardo Mendonça de Moura e Carlos José Pereira de Lucena
{moura,lucena}@les.inf.puc-rio.br

Laboratório de Engenharia de Software, Departamento de Informática – PUC-Rio
Rua marquês de são Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introdução

Análise de programas e especificações é um ramo importante da engenharia de software, pois este está envolvido na otimização de código gerado por compiladores, na verificação de programas e especificações, e na transformação de software em processos de engenharia e reengenharia.
Em geral *Program analysis* [Cousot92] é definida como a geração de informação sobre o comportamento dinâmico de programas através de análise estática, isto é, o programa não é executado com todas as possíveis entradas para obter o resultado da análise. Nós acreditamos que tal definição se encaixa bem nos analisadores automáticos existentes (ex.: *Sign Analysis* e *Interval Analysis*). Entretanto, em nosso projeto nós consideramos analisadores automáticos e interativos.
Atualmente a análise de propriedades de programas é geralmente realizada por ferramentas específicas, que são desenvolvidas para cada linguagem de programação e propriedade. Já que a construção manual destas ferramentas é dispendiosa, estas são difíceis de se obter e freqüentemente não são corretas (não concordam com a semântica da linguagem).
Engenheiros de software se interessam em propriedades que nem sempre são suportadas por um conjunto limitado de analisadores específicos pré desenvolvidos. Desta forma, é desejável que o engenheiro possa desenvolver novos analisadores de forma modular, reutilizando (partes de) analisadores preexistentes.
Nós temos uma forte crença que a construção de analisadores de programas devem ser baseados na semântica formal da linguagem, e o formalismo usado na descrição semântica deve fornecer ferramental suficiente (matemático ou não) para justificar a construção. Outro requisito, é termos descrições semânticas modulares, de forma que seja possível reutilizar descrições na especificação de novas linguagens. No caso desta tese, o formalismo escolhido é uma variação de semântica ações [Mosses92] baseada em *monads* e *modular structured operacional semantics.* Em nossa abordagem, a *action notation* funciona como uma linguagem intermediária, que possui uma interpretação denotacional e operacional. Este fato é importante já que alguns analisadores são mais facilmente descritos usando uma visão operacional, enquanto outros com uma visão denotacional.
Independentemente da abordagem escolhida, operacional ou denotacional, é importante termos descrições modulares. Através de descrições modulares nós podemos reutilizar descrições semânticas e descrições de analisadores. *Monads* e *Monads Transformers* [Moggi89] são utilizados para obter o grau de modularidade desejado na visão denotacional da *action notation*.

2. Framework

O nosso framework para construção de *program analyzers* é baseado nas teorias de *abstract interpretations* [Cousot77, Cousot96, Nie96, Cousot97] e *constraint solving* [Hei92, AW93, Hei94]. Estas teorias provêem um método para construir descrições semânticas aproximadas de programas. Uma descrição semântica aproximada pode ser utilizada para coletar informações sobre programas com o intuito de prover respostas **consistentes** (*safe*) para perguntas sobre o comportamento deste em tempo de execução. Estas aproximações podem ser utilizadas no projeto de métodos de prova manuais ou para especificar analisadores de programa automáticos. No caso de analisadores automáticos, a reposta pode ser apenas parcial ou aproximada já que questões sobre terminação são indecidíveis.
Ao contrário de muitos autores, nós não vemos *abstract interpretations* e *constraint based program analysis* como métodos "rivais" para a análise de programas, mas sim como métodos complementares, já que dependendo da análise desejada um método pode gerar resultados melhores que o outro.
Na teoria de *Abstract Interpretations* a semântica de uma linguagem é vista como a semântica concreta (ou real). Nós podemos ver a definição de um *abstract interpreter* como um processo de transformação, onde os domínios semânticos concretos são transformados (substituídos) em domínios abstratos, que nada mais são do que uma versão do domínio concreto "configurado" para um determinado tipo de análise. O objetivo de uma interpretação abstrata é verificar se uma determinada propriedade abstrata pertence a semântica abstrata de um programa, que nada mais é que uma **correta** aproximação da semântica concreta.
Em geral *Program Analyzers* são desenvolvidos especificamente para uma determinada linguagem de programação e a consistência do mesmo é demonstrada[3] em relação a uma descrição semântica específica. O analisador mapeia

[3] Em geral a demonstração de consistência é feita utilizando-se argumentos informais.

programas P de uma dada linguagem de programação em sua semântica abstrata (ou aproximação) **AbsSem [[P]]**, especificando, desta forma, as propriedades descobertas durante a análise.

$$P \xrightarrow{\text{Interpretação Abstrata}} \text{AbsSem[[P]]}$$

Entretanto, este tipo de abordagem não é interessante para os nossos propósitos, pois envolve a construção de um mesmo analisador para diferentes linguagens. Como nosso objetivo principal é reutilizar ao máximo blocos (pedaços) de descrição semântica e analisadores, nós decidimos utilizar uma abordagem similar a descrita [Cousot97]. Nossa abordagem parametriza a descrição de um interpretador abstrato em função da semântica concreta da linguagem de programação, permitindo portanto que um analisador seja utilizado com diferentes linguagens de programação. Essa solução não possui nada de misterioso, nós simplesmente estamos definindo um analisador em função da meta linguagem utilizada na descrição semântica. Se a nossa meta linguagem for vista como a linguagem intermediária utilizada no *back-end* de compiladores [Muchnick97], nossa abordagem é exatamente igual a utilizada em compiladores comerciais (reais), que mapeiam uma linguagem de programação em uma representação intermediária que é submetida a análise e otimizações. A grande diferença é que enquanto as representações intermediárias utilizadas em compiladores são em geral adhoc e se assemelham com um tipo *assembly*, a nossa representação intermediária possui uma forte base teórica e diversas propriedades que justificam a construção dos analisadores. Outra justificativa para a nossa abordagem, é que da mesma forma que estamos propondo um tipo de representação intermediária, diversos grupos estão propondo representações intermediárias que possam ser utilizadas na construção de diferentes compiladores [JNO97].

$$P \xrightarrow{\text{Tradução}} \text{Meta Ling.} \xrightarrow{\text{Interpretação Abstrata}} \text{AbsSem[[P]]}$$

Constraint based program analysis utiliza a teoria de *constraint solving* [Pou95] para realizar análise de programas. Esta abordagem é interessante, pois define um determinado tipo de análise como um conjunto de *constraints*. Separando, desta forma, a especificação do analisador (extração dos *constraints*) da implementação (*constraint resolution*). A aproximação utilizada nesta abordagem é bem simples, basicamente a dependência entre variáveis do programa é ignorada. Permitindo dessa forma que variáveis do programa sejam associadas a conjuntos de valores durante a análise.

$$P \xrightarrow{\text{Tradução}} \text{Meta Ling.} \xrightarrow{\text{Extração dos } \textit{Constraints}} \textit{Constraints} \xrightarrow{\textit{Constraint Solving}} \textit{Properties}$$

Cada analisador de programas pode ser visto como um sistema lógico (teoria de *general logics*), e a interconexão de diversos analisadores de programas[4] entre si ou com ferramentas externas tais como provadores de teorema nada mais é do que *logic morphisms*. Para justificar estas conexões nós utilizaremos a teoria de *instituições* [GB84] e *general logics* [CM93]. Nós também podemos ver os analisadores como agentes (componentes ou objetos) que comunicam-se entre si para obter a resposta de uma consulta requisitada por um engenheiro, ou por um otimizador de software, ou por um sistema transformacional.

Nós acreditamos que uma abordagem minimalista é o caminho a ser seguido no desenvolvimento de software. Nesta abordagem, um desenvolvedor deve implementar um programa usando a linguagem mais abstrata e simples que permite a solução do problema. Através dessa abordagem propriedades de programas podem ser verificadas de uma forma mais simples, otimizações específicas de domínio podem ser aplicadas automaticamente, e as intenções do desenvolvedor se mantém claras, não ficando ofuscadas dentro de uma linguagem de baixo nível. Note que este princípio é difícil de ser formalizado, pois envolve noções subjetivas como "mais abstrato" e "mais simples". E o termo "mais simples" pode gerar confusões, pois alguém pode vir a questionar se lambda calculus é uma linguagem simples. Logo, o termo "mais simples" está relacionado ao poder computacional e a complexidade da semântica de uma linguagem e não a simplicidade de sua sintaxe, pois apesar da sintaxe de lambda calculus ser bastante simples, a sua semântica é complexa.

Dentro dessa abordagem minimalista, nós acreditamos no uso de *Domain Specific Languages* (DSLs) [Bell94, Hudak96]. DSL é uma linguagem formal que é expressiva em relação as abstrações de um determinado domínio de aplicação. Uma DSL é utilizada freqüentemente para especificar formalmente designs de software. Como exemplos, nós podemos citar: YACC, SQL e TEX. A principal vantagem em utilizar DSLs é que o especialista de um domínio pode expressar conceitos específicos de domínio diretamente, sem a necessidade de codificá-los em uma linguagem de baixo nível. Logo além dessa vantagem, DSLs em geral não são *Turing* completas, facilitando portanto em muito a análise de especificações escritas na mesma. Os exemplos de DSLs apresentados podem não ser muito convincentes,

[4] Construídos usando *abstract interpretation* ou *constraint solving*

mas eles foram escolhidos por serem de conhecimento geral, nós acreditamos que diversas DSLs serão *process oriented.*

Referências

[AW93] A. Aiken, E. Wimmers, “Type Inclusion Constraints and Type Inference”, In *Proceedings of the 1993 Conference on Functional Programming Languages and Computer Architecture*, 1993.

[Bell94] J. Bell, “Software design for reliability and reuse: A proof-of-concept demonstration”, In *TRI-Ada’94 Proceedings,* ACM, November 1994, pp. 396-404.

[CM93] M. Cerioli, J. Meseguer, “May I Borrow Your Logic?”, *technical report*, SRI International, Computer Science Laboratory, 1993.

[Cousot77] P. Cousot, R. Cousot, “Abstract Interpretation: a unified lattice model for static analysis of programs by construction of approximative fixpoints”, in *Proc. of the 4th ACM Symp. on POPL*, 238-252, 1997.

[Cousot92] P. Cousot, R. Cousot, “Abstract Interpretation Frameworks”, *Journal of Logic and Computation*, 2(4):511-547, August 1992.

[Cousot96] P. Cousot, “Abstract Interpretation”, In *ACM Computing Surveys*, Vol. 28, No.2, pages 324-328, June 1996.

[Cousot97] P. Cousot, “Abstract Interpretation Based Static Analysis Parameterized by Semantics”, *Proceefings of the 4th International Symposium on Static Analysis, SAS’97*, Paris, France 8-10 September 1997, P. van Hentenryck (Ed.), Lecture Notes in Computer Science 1302, Springer-Verlag, pp. 388-394

[GB84] J. A. Goguen, R. M. Burstall, “Introducing Institutions”, In E. Clarke and D. Kozen, editors, *Logics of Programming Workshop*, number 164 in LNCS, Springer Verlag, Berlin, 1984.

[Hei92] N. Heintze, “Set Based Program Analysis”, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, 1992.

[Hei94] N. Heintze, “Set Based Analysis of ML programs”, In *Proceedings of the 1994 ACM Conference on Lisp and Functional Programming,* 1994.

[Hudak96] P. Hudak, “Building Domain-Specific Embedded Languages”, *Computing Surveys*, 28A(4), ACM, 1996.

[JNO97] S. P. Jones, T. Nordin, D. Oliva, “C--: a portable assembly language”, *To appear in the proceedings of IFL’97*.

[Moggi89] E. Moggi, “Computanional lambda-calculus and monads”, *Proc. 4th Annual Symposium on Logic in Computer Science*, Asilomar CA, IEEE Computer Society Press, Washiton, 1989.

[Mosses92] P. D. Mosses, *Action Semantics*, Cambridge Tracts in Theoretical Computer Science 26, 1992.

[Muchnick97] S. S. Muchnick, *Advanced Compiler Design and Implementation*, Morgan Kaufman Publishers, 1997.
[Pou95] D. Pountain, “Constraint Logic Programming”, *Byte Magazine*, February 1995.

[Nie96] H. R. Nielson, F. Nielson, *Semantics with Applications: Model-Based Program Analysis,* Course Notes, 1996.

2BuyNet:
Um Framework para suporte ao Desenvolvimento e Manutenção de Web Sites comerciais

Luidi Xavier Fortunato

luidi@les.inf.puc-rio.br

Laboratório de Engenharia de Software, Departamento de Informática – PUC-Rio
Rua marquês de são Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introdução

A motivação para o desenvolvimento do 2BuyNet surgiu por dois motivos principais. Em primeiro lugar como estudo de casos para a metodologia de desenvolvimento de frameworks não convencionais [2]. E para a criação de um serviço de geração automática de lojas, que naturalmente surgiu com a implementação do framework. Nesse contexto, o 2BuyNet se propoem a ser uma solução para o desenvolvimente de lojas para comércio eletrôncio[4,5].

A estratégia de desenvolvimento da ferramente segue as instruções previstas na metodologia mencionada acima. As etapas de desenvolvimento começam por um estudo de casos de todas as aplicações do domínio estudado (nesse caso, comércio eletrônico). Esse estudo nasce de uma análise comparativa entre todos os produtos concorrentes, para o levantamento de todas as funcionalidades.

Como resultado desse estudo de casos e para servir como estratégia de implementação, foi criada uma lingugem de domínio específico (DSL) [3] abrangendo o domínio de comércio eletrônico com foco para criação de lojas virtuais. Essa DSL tenta abranger o máximo de funcionalidades possíveis, no domínio comércio eletrônico. Suas funcionalidades incluem comandos para manipular departamentos, produtos, gerência de loja e todas outras funcionalidades necessarias para criar e operacionalizar uma loja.

2. Estudo de Casos

Um bom estudo de casos é fator decisivo para criação de uma eficiente DSL, a metodologia não especifica quantos exemplos devem ser analisados durante o estudo de casos. Impiricamente, para o desenvolvimento da DSL do 2BuyNet, chegou-se ao número de 6 casos.

Análises fora desses 6 exemplos, recaiam em subcasos dos já analisados. Consequentemente, não agregavam nenhum valor a DSL, produto final dessa análise. Através de um processo chamado de Unificação, fez-se não só o levantamento das funcionalidades dos seis exemplos, mas também dos pontos em comum entre eles. Dessa forma, as primeiras funcionalidades do framework começam a ser delineadas.

O critério de escolha desses 6 exemplos, foi fundamentalmente baseado na potencialidade do software e da empresa que o representava. Três grupos, contendo dois exemplos em cada foram criados. O primeiro grupo representava as grandes empresas com seus produtos, Microsoft com o Site Server 2.0 e a Compaq com o Inex. O segundo grupo, duas pequenas empresas, com soluções próprias, AbleCommerce e Ez2Shop. E o último grupo com duas empresas que lançaram não produtos, mas serviços (no mesmo modelo do 2BuyNet), Icat e a Viaweb.

3. 2BuyNet

O 2BuyNet surgiu como uma interface amigável para a interação com o framework de lojas virtuais e consequentemente como um instanciador de lojas on-line.

Apesar de poder ser apresentado como um software tradicional, o 2BuyNet foi pensado e produzido para ser um serviço para a Web.

O web site que representa o 2BuyNet apresenta uma interface restritiva em relação a máquina do framework que roda por trás. Isso se deve ao fato do serviço estar voltado para um público final sem conhecimento em informática, mas com fluência no seu domínio (comércio e lojas). E também, para que o próprio usuário não se sinta perdido durante o processo de criação de uma loja, diante de inúmeras opções.

Atualmente, o serviço é dividido em três etapas principais, a criação da conta, a administração e a loja, propriamente dita. As duas primeiras, representam a visão do administrador da loja com relação ao seu negócio. A última (loja) representa a visão do cliente do sistema, e é tratada pelo sistema como a parte dinâmica do instanciador.

Para cada loja cadastrada, é gerada sua versão web automaticamente em função dos parâmetros entrados pelo administrador. Todas as metáforas normalmente utilizadas no mercado, como carrinho de compras, cadastro de clientes, carteira digital, entre outras, são implementadas nas lojas automaticamente.

Pela parte administrativa é possível abrir, configurar e gerar uma loja para a Internet contendo todas as funcionalidades de uma loja tradicional (desenvolvida a partir da especificação informal de um cliente e implementada posteriormente).

4. Conclusão

O 2BuyNet valida uma metodologia para o desenvolvimento de frameworks para aplicações em domínios ainda considerados novos. Dessa forma, sua contribuição está nas interações e mudanças que refletiram no amadurecimento da metodologia. Da mesma forma como que Aulanet iniciou o processo, o 2BuyNet vem consolidar a metodologia.
Como uma implementação, o 2BuyNet apresenta uma máquina geradora e administradora de web sites comerciais, que reduz o custo de desenvolvimento e padroniza as funcionalidades nessa área.

Referências

1. "AulaNet - An Object-Oriented Environment for Web-based Education", S. Crespo, M. F. Fontoura, and C. J. Lucena, International Conference of the Learning Sciences (ICLS'98), Atlanta, USA, 1998.

2. "Using Viewpoints, Frameworks, and Domain-Specific Languages to Enhance Software Reuse", S. Crespo, M. F. Fontoura, and C. J. Lucena, European Reuse Workshop (ERW'98), Madrid, Spain, 1998.

3. P. Hudak, "Building Domain-Specific Embedded Languages", Computing Surveys, 28A(4), ACM, 1996.

4. The White House, "A Framework for Global Electronic Commerce", July 1, 1997, http://www.ecommerce.gov/framewrk.htm

5. William J. Clinton, "Presidential Directive on Electronic Commerce", July 1, 1997, http://www.ecommerce.gov/presiden.htm

Uso de “Frameworks” para o Projeto de Aplicações Hipermídia

Luiselena Luna
Luiselena@inf.puc-rio.br

Departamento de Informática – PUC-Rio
Rua Marquês de são Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introdução

Com a consagração da Internet e dos browsers para Web, as aplicações que usam modelos hipertextuais ganharam difusão mundial, tornando-se uma das principais formas de navegação para sistemas de informação. Contudo, desenvolver uma aplicação multimídia/hipermídia complexa tem uma grande dificuldade pois são muitos os fatores envolvidos e que precisam ser controlados. A mudança de modelo necessária para a utilização de acessos hipertextuais às informações faz com que os projetistas de sistemas de informação, acostumados a fazer projetos para ambientes convencionais, enfrentem várias dificuldades ao projetar sistemas de informação para a Web [Rossi96, Pontes97]. A maior destas dificuldades está na falta de uma metodologia adequada de projeto de sistemas que contemple a idéia de hipertexto, especificando e documentando o projeto de maneira clara e concisa.

Várias metodologias de projeto têm sido desenvolvidas com o objetivo de auxiliar na autoria de aplicações hipermídia. Estas metodologias descrevem os passos necessários para o desenvolvimento de uma aplicação hipermídia, sendo utilizadas como forma de comunicação entre projetistas, implementadores e usuários. O processo de desenvolvimento pode ser mapeado em quatro etapas: Modelagem Conceitual do domínio do problema, Projeto de Navegação, Projeto de Interface e Implementação.

A metodologia OOHDM (Object Oriented Hypermidia Method) [Rossi96] é, entre todas as metodologias, aquela que tem o maior grau de desenvolvimento, particularmente no projeto de navegação. Também é cada vez maior o número de aplicações modeladas através desta metodologia, disponibilizadas na WWW. Por estes motivos e por ser possível estudar aplicações já implementadas na WWW, modelando-as com o OOHDM, é que esta metodologia será a base do enfoque do presente trabalho. Mesmo utilizando um modelo de projeto para desenvolver aplicativos hipermídia, a complexidade do projeto é elevada. Esta complexidade é a grande motivação para o desenvolvimento deste trabalho, que propõe a construção de frameworks para o Projeto de Navegação de Aplicações Hipermídia., visando a reutilização de designs para domínios específicos.

2. Abordagem

Aplicações hipermídia são geralmente construídas a partir do zero, isto porque o mercado hipermídia ainda é muito jovem e principalmente porque os modelos de projeto existentes não são eficientes no que diz respeito a reutilização de componentes de estruturas hipermídias existentes. Entretanto, projetistas experientes reusam soluções que eles já usaram previamente. A idéia aqui é a de reutilização de projeto, ou seja, a reutilização de abstrações de mais alto nível que o código.

A abordagem utilizada neste trabalho para prover a reutilização de projetos de aplicações hipermídia modelados com OOHDM é a construção de frameworks OOHDM para domínios específicos.

3. Objetivos

Com a criação de frameworks para o Projeto de Navegação usando o OOHDM, procura-se definir padrões de modelagem para domínios determinados e atingir, através de seu uso, uma construção mais rápida e otimizada de aplicações multimídia/hipermídia. A opção por frameworks se deve ao fato que frameworks definem um projeto genérico e o seu uso implica em reuso de projeto, diminuindo tempo e esforço para a construção de uma aplicação.

Como resultado deste trabalho teremos a definição de notações para expressar padrões de modelagem, a definição de pontos de flexibilizações, a definição de regras de consistência para se fazer a instanciação e a construção de alguns modelos de frameworks para domínios determinados.

O processo realizado para a construção dos frameworks aqui apresentados compreende o estudo de várias aplicações de um mesmo domínio, desenvolvendo a modelagem OOHDM para cada caso. A partir das diversas soluções propostas, foi projetado o framework daquele domínio. Cada framework representado utiliza-se de recursos de design patterns já conhecidos e amplamente utilizados. Unindo a metodologia OOHDM e o conhecimento de design patterns teremos os frameworks OOHDM para domínios específicos.

A modelagem OOHDM provê primitivas de projeto de alto nível e mecanismos de abstração baseados no paradigma de orientação a objetos. O processo OOHDM compreende as fases de Modelagem Conceitual, Projeto de Navegação, Projeto de Interface Abstrata e Implementação. A proposta para a construção de frameworks abrange os modelos Conceitual e Navegacional do método OOHDM, tendo como resultado um Modelo Genérico Conceitual e um Modelo Genérico Navegacional.

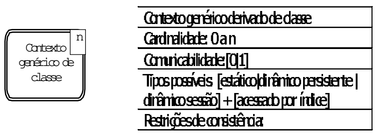
4. Pontos de Flexibilização do Framework

Para estudar os domínios e definir os frameworks foram consideradas alguns pontos de flexibilização que irão permitir a instanciação do modelo navegacional OOHDM para cada domínio de aplicação.

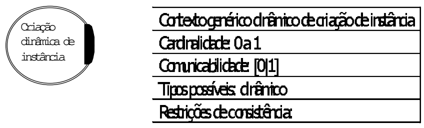
Esq.Conceitual	**Esq.Classes de Navegação**	**Esq. Classes em Contextos**	**Esq. de Contextos**	**Cartões**
1. modelo conceitual proposto para cada domínio de aplicação é abrangente o suficiente dentro dos domínios estudados. Esta abrangência fez com que para o modelo conceitual dos frameworks construídos não existam pontos de flexibilização.	2. Não mapeamento de uma classe conceitual 3. Mapeamento Fixo 4. Mapeamento Circunscrito 5. Não representação de Elos 6. Não Mapeamento de Métodos	7. Não Mapeamento de uma classe em contexto 8. Mapeamento Fixo 9. Mapeamento Circunscrito	10. Comunicabilidade 11. Cardinalidade 12. Tipo (estático, dinâmico, acessado por índice..) 13. Subdiagramas	14. Campos dos cartões podem ser preenchidos pelo projetista na hora da instanciação 15. Grupos de usuários podem ser instanciados ou não

5. Exemplo de Algumas Notações Utilizadas no Framework

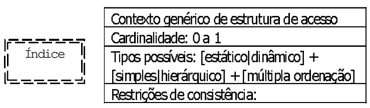
Contexto genérico para os contextos derivados de classe



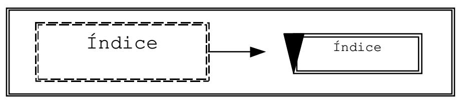
Contexto genérico de criação de instância dinamicamente



Estrutura de acesso genérica

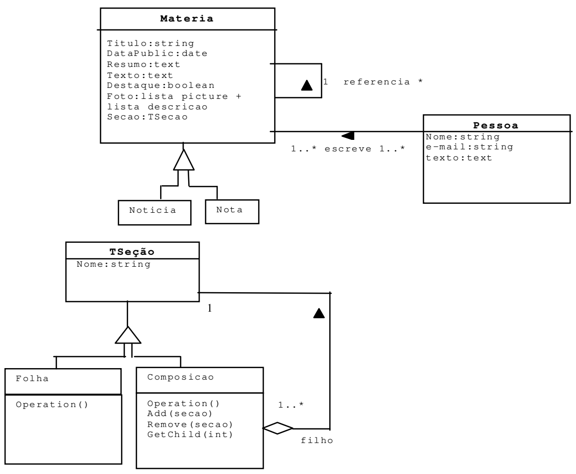


Subdiagrama de índice

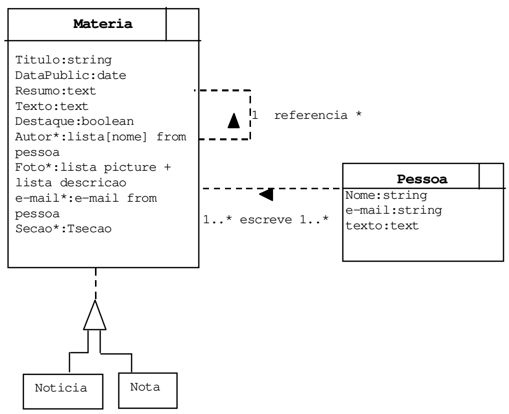


6. Exemplo de um Framework para o domínio de Publicações On-Line

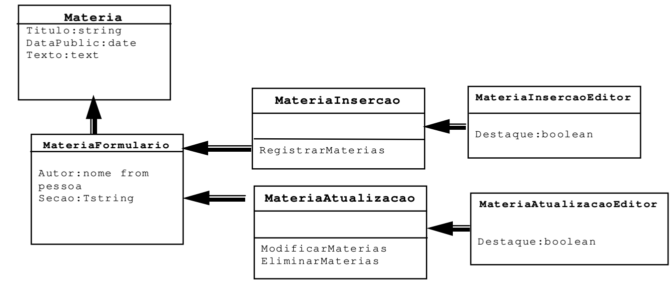
Modelagem Conceitual



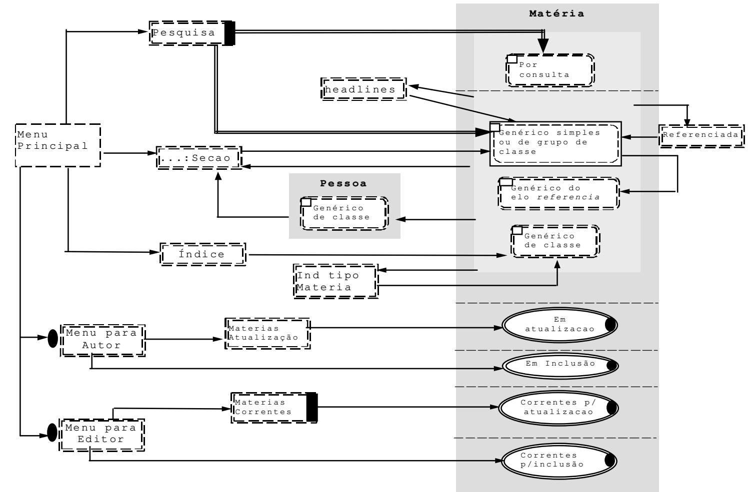
Modelagem de Classes de Navegação



Classes em Contexto



Esquema de Contextos de Navegação



7. Contribuições Esperadas

Com este trabalho espera-se: Catalogar frameworks que apresentem a modelagem conceitual e de navegação para vários domínios de aplicações hipermídia, usando a metodologia OOHDM, contribuindo dessa forma, para a minimização da complexidade de modelar aplicações hipermídia. Mesmo que um domínio não tenha sido estudado nesta dissertação, o projetista de uma aplicação hipermídia para o domínio não catalogado pode se beneficiar dos frameworks que serão construídos, pois neles existirão problemas comuns a qualquer domínio e a solução destes problemas comuns será dada nos frameworks e em seus estudos de casos. Esta é mais uma contribuição que será o uso de Design Patterns para solucionar problemas comuns a qualquer domínio.

Referências

GAMMA — Gamma, R.; Helm, R.; Johnson and J. Vlissides "Design Patterns: Elements of reusable object-oriented software", Adidson Wesley, 1995.

GARRIDO 96 — Garrido, A.; Rossi, G. "A Framework for Extending Object-Oriented Applicatios with Hypermidia Functionality", New Review of Hypermidia and Multimidia, Volume 2, Taylor Graham Publishing, 1996.

JOHNSON 97 — Johnson, R. "Components, Frameworks, Patterns", 1997

PIZZOL98 — Pizzol, A. "Um Framework para Implementação de Aplicações Hipermídia modeladas com OOHDM na WWW", PUC-RIO, Dissertação de mestrado – 1998

VILAIN 98 — Vilain, P. "Notação da Metodologia OOHDM utilizando UML", PUC-RJ, 1998.

SCHWABE — Schwabe, D.; Rossi, G.; Garrido, A. "Design Reuse in Hypermidia Applications Development"

LUNA 98 — Luna, L. "Uso de "Frameworks" para o Projeto de Aplicações Hipermídia", PUC-RIO Proposta de Dissertaçào de Mestrado - 1998

ECOHOOD: Construção de SGBDs Configurados utilizando Frameworks

Rubens Melo, Fábio Porto, Fernanda Lima,Álvaro Barbosa
{rubens, porto, ferlima, alvaro} @ inf.puc-rio.br

Laboratório TecBD, Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro – PUC/RJ
Rua Marquês de São Vicente, 225 - Rio de Janeiro - RJ
22453-900 – Brasil

1. Introdução

O uso de Sistemas Gerenciadores de Banco de Dados (SGBDs) para armazenamento de dados em aplicações tradicionais é uma realidade na indústria de software, reforçada com o surgimento de bancos de dados relacionais. Hoje, com o aparecimento de novos tipos de aplicações, como Multimídia, Workflow, Datawarehousing, CAD/CAM, CASE, GIS e aplicações baseadas na Web, novos desafios surgiram para a pesquisa em Bancos de Dados. Bancos de Dados Relacionais foram inicialmente concebidos para fornecer suporte a aplicações com estruturas de dados e transações simples. As novas aplicações mencionadas possuem novos requisitos, exemplificadas por: transações longas, processamento de consultas complexo, armazenamento de objetos grandes como vídeo e audio.

Os desafios apresentados foram abordados em projetos classificados por Cattell [Catt94] como SGBDs Extensiveis e Geradores de DBMS. A proposta de extensibilidade, em Postgres [SR86] e Starbust [S+86] permite acrescentar novos tipos de dados fornecendo estrutura e comportamento necessários aos requisitos dos novos tipos de aplicação. SGBDs Extensiveis, em geral, possuem arquitetura estática. Sua extensibilidade provem do suporte aos novos tipos de dados através de novos métodos de acesso, bem como linguagens de consulta estendidas capazes de suportar, por exemplo, objetos complexos e funções definidas pelo usuário. No contexto de Geradores de SGBDs, EXODUS [C+90] é um gerador de código que adapta funcionalidade de SGBD específica, enquanto GENESIS [B+90] é um construtor de SGBD baseado em composição. Ambos propoem uma abordagem mais genérica para a construção de SGBDs.

De acordo com Silberschatz e Zdonik [SZ97], extensibilidade e componentização são importantes tópicos de pesquisa. Os autores afirmam que é necessário buscar formas de “abrir” a arquitetura de SGBDs de modo que novos serviços possam ser incorporados e funcionalidades do banco de dados possam ser configuradas de maneira flexível, de acordo com as necessidades da aplicação.

Nosso projeto ECOHOOD (Environment of Configurable Heterogeneous Object-Oriented Databases) propoe uma abordagem nova para construir SGBDs baseados em "Frameworks". Nosso objetivo é configurar SGBDs específicos para diferentes domínios de aplicação. Nesse cenário, um SGBD configurado é um conjunto de serviços interrelacionados requeridos para a gerência de dados num domínio específico. Uma primeira abordagem para atender a este objetivo esta sendo investigada utilizando-se os conceitos de "Frameworks" da area de pesquisa de Engenharia de Software.

ECOHOOD é um projeto de pesquisa em andamento no Departamento de Informatica da Universidade PUC-Rio e utiliza a experiência adquirida em dois de nossos projetos anteriores: HEROS (Heterogeneous Object System) [USM94] e EITIS (Environment of Integrated Tools for Interactive Systems) [Melo87].

Em [M+98], apresentamos uma contribuição ao projeto ECOHOOD, através de uma proposta genérica para a construção de SGBDs configurados. Nosso objetivo é prover uma arquitetura onde um Implementador de Banco de Dados possa adaptar qualquer parte de um SGBD que tenha sido previamente identificada como "flexível”. A adaptação será possível para todos os servicos básicos de SGBDs como: processador de consultas, gerente de transação, gerente de regras, gerente de armazenamento, e gerente de buffer, também sendo permitida a inclusão de novos serviços.

Conforme mencionado, nossa proposta se baseia no conceito de Frameworks. Frameworks orientados a objetos (OO) definem uma arquitetura OO com classes abstratas pré-definidas a serem especializadas e componentes a serem “plugados” na arquitetura. Esta tecnologia provê a flexibilidade exigida pelos novos tipos de aplicações através da configuração de SGBDs com comportamento adequado.

Acreditamos que, ao invés de sobrecarregar um SGBD com vários serviços não aplicáveis, podemos configurar um SGBD composto somente de módulos de software que implementam os serviços necessários ao domínio da aplicação onde sera utilizado.

As principais contribuições de nosso trabalho são: a arquitetura proposta, que isola as dificuldades de desenvolver um software complexo como um DBMS; o uso de esquemas em um ambiente de framework e a modelagem dos serviços de banco de dados com a identificação de Hot-Spots como pontos de flexibilidade. Apresentamos, tambem, dois exemplos de Frameworks de serviços de banco de dados: um Framework de Gerente de Regras e um Framework de Catálogo, com seus respectivos Hot-Spots.

O trabalho será inicialmente testado através da construção de framerworks para *middlewares* do tipo SGBD para acesso a fontes de dados heterogêneos.

Referências

[BP98] Blaha, M., and Premerlani, W. *Object-Oriented Modeling and Design for Database Applications,* Prentice-Hall, 1998.

[BT95] Batory, D.S., and Thomas, J. "P2: A Lightweight DBMS Generator", Department of Computer Sciences, University of Texas at Austin, *Technical Report TR-95-26*, http://www.cs.utexas.edu, August 1995.

[B+90] Batory, D.S., Barnett, J.R., Garza, J.F., Smith, K.P., Tsukuda, K., Twichell, B.C., and Wise, T.E. "GENESIS: An Extensible Database Management System", in Maier, D., and Zdonik, S. (editors), *Readings on Object-Oriented Database Systems*, Morgan-Kaufmann, 1990.

[Catt94] Cattell, R.G.G. *Object Data Management, Object-Oriented and Extended Relational Database Systems*, Addison-Wesley, 1994.

[C+90] Carey, M.J., DeWitt, D.J., Graefe, G., Haight, D.M., Richardson, J.E., Schuh, D.T., Shekita, E.J., and Vandenberg, S.L. "The EXODUS Extensible DBMS Project: an Overview", in Maier, D., and Zdonik, S. (editors), *Readings on Object-Oriented Database Systems*, Morgan-Kaufmann, 1990.

[Elma92] Elmagarmid, A.K., editor. *Database Transaction Models For Advanced Applications,* Morgan-Kaufmann, 1992.

[EN94] Elmasri, R., and Navathe, S.B. *Fundamentals of Database Systems,* 2nd Edition, Addison-Wesley, 1994.

[FS97a] Fayad, M., and Schmidt, D. "Object-Oriented Application Frameworks", *Communications of the ACM*, vol. 40, n. 10, October 1997, pp.32-38.

[FS97b] Fauler, M., and Scott, K. "UML Distilled – Applying the Standard Object Modeling Language", Addison-Wesley, 1997.

[GR93] Gray, J., and Reuter, A. *Transaction Processing: Concepts and Techniques,* Morgan-Kaufmann, 1993.

[Melo87] Melo, R.N. "EITIS: Environment of Integrated Tools for Interactive Systems", *Anais do 1º Simpósio Brasileiro de Engenharia de Software*, Petrópolis, Rio de Janeiro, Brasil, 1987.

[M+98] Melo, R.N., Porto, F.A.M., Lima, F., Barbosa, A. "ECOHOOD: Constructing Configured DBMSs based on Frameworks*", Anais do 13º Simpósio Brasileiro de Banco de Dados*, Maringá, Paraná, Brasil, 1998.

[Pree94] Pree, W. *Design Patterns for Object-Oriented Software Development*, Addison-Wesley/ACM Press, 1994.

[Pree96] Pree, W. *Framework Patterns*, SIGS Books & Multimedia, 1996.

[RJ97] Roberts, D. and Johnson, R. *Evolving Frameworks: A Pattern Language for Developing Frameworks*, Addison-Wesley Reading, Massachusetts, 1997.

[SKS97] Silberschatz, A., Korth, H.F., and Sudarshan, S. *Database System Concepts,* 3rd Edition, McGraw-Hill, 1997.

[SR86] Stonebraker, M., and Rowe, L. "The Design of POSTGRES*", Proceedings of the SIGMOD Conference*, May 1986.

[SZ97] Silberschatz, A., and Zdonik, S. "Database Systems – Breaking Out of the Box", *SIGMOD Record*, vol. 26, n. 3, setembro 1997, pp. 36-50.

[Shor94] The Shore Project Group, "An Overview of Shore", Computer Sciences Department, *http://www.cs.winsc.edu/shore/doc/overview*, University of Wisconsin, 1994-97.

[S+86] Schwarz, P. et al. "Extensibility in the Starbust Database Systems", *Proceedings of the International Workshop on Object-Oriented Database Systems*, setembro 1986.

[USM94] Uchoa, E.M., Silva, S.D., and Melo, R.N. "HEROS Data Model – Object Oriented Heterogeneous Database System*", Anais do 9º Simposio Brasileiro de Banco de Dados*, São Carlos, São Paulo, Brazil, 1994.

[WBT92] Wells, D.L., Blakeley, A., and Thompson, C.W. "Architecture of an Open Object-Oriented Database Management System", *IEEE Computer*, outubro 1992.

[WC96] Widom, J., and Ceri, S., editors, *Active Database Systems - Triggers and Rules For Advanced Database Processing,* Morgan-Kaufmann, 1996.

[YM98] Yu, C.T., and Meng, W. *Principles of Database Query Processing for Advanced Applications*, Morgan-Kaufmann, 1998.

Utilizando Requisitos Não Funcionais para Análise de Modelos Orientados a Dados

Luiz Marcio Cysneiros *, Julio Cesar Sampaio do Prado Leite*
{cysneiro, julio}@inf.puc-rio.br

Departamento de Informática PUC- Rio
Rua Marquês de São Vincente 225
22453-900 - Rio de Janeiro, Brasil

1. Introdução

Recentes pesquisas apontam para a necessidade de modelos conceituais serem capazes de lidar com metas, de forma a poderem retratar situações complexas que existem no mundo real. Este trabalho aborda o uso de requisitos não funcionais (RNF) na análise de modelos orientados a dados. Esta análise é baseada no uso do Léxico Ampliado da Linguagem (LAL) como âncora para a construção de um grafo de RNFs e para a construção do modelo de dados e de heurísticas para a validação do modelo de dados. Nós adotamos o largamente utilizado modelo entidade relacionamento como o modelo de representação de dados. Apesar de apresentarmos apenas o modelo ER, acreditamos que o método apresentado neste artigo possa ser estendido para outros modelos de dados.

2. Requisitos Não Funcionais

Recentes trabalhos na engenharia de requisitos [Dardenne 95] [Mylopoulos 92] [Mylopoulos 95] apontam para a crescente necessidade de modelos conceituais serem capazes de lidar com metas, de forma a retratar com mais precisão necessidades do mundo real. Uma das vantagens de termos modelos conceituais que lidam com metas é a capacidade de se representar aspectos não funcionais como, confidencialidade, performance, qualidade e precisão.
Acreditamos, assim como outros autores [Mylopoulos 92] [Loucopoulos 95] [Kirner 96] [Chung 95] [Kotonya 95], que esses aspectos não funcionais devam ser tratados como requisitos não funcionais. Requisitos não funcionais, ao contrário dos funcionais, não expressam nenhuma função (transformação) a ser implementada em um sistema de informações; eles expressam condições de comportamento e restrições que devem prevalecer.
A identificação de impactos, causados por restrições operacionais ao software ou por específicos comportamentos, mostrou-se como um grande desafio a ser contornado para que tenhamos softwares que, satisfazendo seus requisitos não funcionais, sejam desenvolvidos em tempo hábil.
A literatura tem registrado vários casos onde fica evidente que a falta de um tratamento adequado dos requisitos não funcionais pode levar a resultados desastrosos. Um desses é o caso do serviço de ambulâncias da cidade de Londres [Finkelstein 96] que foi alvo de uma rigorosa auditoria. Essa auditoria é, certamente, uma das peças mais significativas sobre os custos da não utilização dos preceitos de engenharia de software na construção de sistemas complexos.
Acreditamos, assim como no projeto EAGLE [EAGLE 95], que requisitos não funcionais estão normalmente ligados a requisitos funcionais.
Este trabalho, em sua versão origina estendida, é um aprofundamento da proposta apresentada por Cysneiros [Cysneiros 97], onde os requisitos não funcionais são apresentados e representados inicialmente em um grafo de RNFs [Mylopoulos 92], para em seguida serem representados no modelo entidade relacionamento de forma a possibilitar uma melhor identificação dos impactos que a inclusão de requisitos não funcionais podem trazer ao modelo de dados.

Referências

[Chung 95] Chung, L., Nixon, B.A. "Dealing with Non-Functional Requirements: Three Experimental Studies of a Process-Oriented Approach" *Porch. 17th Int. Con. on Software Eng*. Seatle, Washington, April 24-28, 1995.

[Cysneiros 97] Cysneiros, L.M. and Leite, J.C.S.P. "Definindo Requisitos Não Funcionais" *Anais do XI SBES,* Out 1997, pp:49-54.

[Dardenne 95] Dardenne, A.. van Lamsweerde, Fickas, S.. "Goal Directed Requirements Acquisition". *Science of Computer Programming, Vol. 20* Apr. 1993, pp. 3-50.

[Finkelstein 96] Finkelstein, A. and Dowell J. "A comedy of Errors: The London Ambulance Service Case Study" *Proceedings of the Eighth International Workshop on Software Specification and Design*, IEEE Computer Society Press 1996, pp. 2-5.

[EAGLE 95] Evaluation of Natural Language Processing Systems" http://www.issco.unige.ch/ewg95, 1995.

[Kirner 96] Kirner T.G. , Davis A .M. , "Nonfunctional Requirements of Real-Time Systems", *Advances in Computers, Vol 42* pp 1-37.

* This work was suported by CNPq
* This work is supported in part by CNPq grant n. 510845/93-2

[Kotonya 95] Kotonya G. , Sommerville I. , "Requirements Engineering With Viewpoints", *Cooperative Systems Engineering Group Technical Report* CSEG/10/1995.

[Lamsweerde 91] A. Lamsweerde, A. Dardenne and F. Dubisy, KAOS Knowledge Representations as Initial Support for Formal Specification Processes, *Universite Catholique de Louvain, Unite d'Informatique, Research Report* RR-91-8, 1991.]

[Leite 93] Leite J.C.S.P. and Franco, A.P.M. "Languages" *in Proceedings of the First IEEE International Symposium on Requirements Engineering*, SanDiego, Ca, IEEE Computer Society Press, 1993 pp. 243-246.

[Leite 95] Leite J.C.S.P., Oliveira, A.P.A., "A Client Oriented Requirements Baseline", *in Proceedings of the Second IEEE International Symposium on Requirements Engineering*, York, UK, IEEE Computer Society Press, 1995 pp. 108-115.

[Loucopoulos 95] Loucopoulos, Pericles and Karakostas, *"Vassilios System Requirements Engineering" ,* McGraw Hill International Series in Software Engineering, 1995.

[Mylopoulos 92] Mylopoulos,J. Chung, L., Yu, E. and Nixon, B., "Representing and Using Non-functional Requirements: A Process-Oriented Approach." *IEEE Trans. on Software Eng, 18(6),* June 1992, pp.483-497.

[Mylopoulos 95] Mylopoulos, J. "Capturing Intentions in Requirements engineering: A modeling Perspective" *Invited Talk at the IFIP WG 2.9 meeting*, Bramshill, UK, March 1995.

[Navathe 92] Navathe, S.B., "Evolution of Data Modeling for Databases*" Communications of the ACM, Vol 35* No 9 1992 pp 112-123.

Designing Configurable Distributed Multimedia Services*♣

Sérgio Colcher, Luiz Fernando G. Soares
{colcher,lfgs}@inf.puc-rio.br

Laboratório Telemídia, Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introduction

Multimedia services must be designed to attend the quality constraints required by any combination of media types and applications. The increase in the diversity of such media types and applications has led to the definition of a multitude of different multimedia services with different quality of service (QoS) requirements.

Most distributed multimedia systems do not provide QoS at all phases of media treatment, and when provided, QoS is usually not integrated throughout these phases. The orchestration of the different phases of media treatment is fully delegated to the architecture or application developer. Moreover, the ongoing increase in multimedia codification, services and applications continuosly impose new characteristics not previously considered, which cannot be handled if services are not designed to be dynamically changed.

We propose the construction of a flexible environment, where QoS constraints can be dynamically configured by applications. The targets of the configuration process might be the architecture itself, which includes network protocols and execution environments, as well as applications or end-user services. We focus on an object model that can be used as the basis of new OO programming languages or as extensions to the existing ones. The model can also be introduced as a platform specific feature or service within some distributed object computing (DOC) environment like CORBA [8], ODP [7] or TINA DPE [1].

2. Virtual Processing Environment

We define a *virtual processing environment* as being composed of an *object engine* and *several components*. An object engine provides a complete execution environment for each component, acting as the controller of the virtual processing environment and holding the control of a set of virtual resources, including a *virtual memory space* and one or more *virtual processors*. A virtual memory space is used for the storage of components and engine, and virtual processors are allocated to components in order to provide them with processing capabilities.

Components are subdivisions of the virtual memory space structured to be filled with implementations. An implementation comprises code and data and is structured through an object. Objects are defined as usual: entities that possess internal state and are instances of classes. A class is an implementation of a particular type.

Components can interact, requesting services to other components or to themselves, assuming the role of clients and servers during their lifecycles. An interaction corresponds to a request for the execution of an operation. Interactions are intermediated by the object engine.

The generic form of describing the cooperation among components of an object-oriented (OO) application consists of considering clients and servers that interact through operation requests. An *interaction* corresponds to a client request for the execution of an operation and a possible reply from the server. Interactions among components in a virtual processing environment involve, in an implicit way, the generation of messages exchanged through the object engine, corresponding to requests and replies. Participants of an interaction are not aware of the delivery of implicit messages.

However, in certain applications, some more specific patterns could be taken into account and object engines should support these patterns in order to ease some of the programmer tasks. Design patterns [5,9,2]} have been used as a form of increasing software quality through the reuse of well designed and documented structures.

Some patterns place purely architectural constraints, as for example the *pattern of pipes and filters*. In pipes and filters based applications, a system is modeled as: (i) a set of components (filters) that process data and (ii) a set of connections (pipes) among these components, that allow the explicit exchange of the manipulated information. Each filter is responsible for a specific sub-activity (for example, a compiler parser or a protocol layer entity) of the entire application.

The kind of constraint imposed by pipes and filters can be considered as a cooperation pattern among components. In opposition to the free and direct interaction (through operation requests) between components, in the pipes and filters

* An extended version of this paper was submitted to the *International Conference on Multimedia Computing and Systems (ICMCS'99).*

♣ This research was supported in part by the Brazilian Telecommunications Enterprise (Embratel). The opinions and conclusions contained in this document are those of the authors and should not be interpreted as representing official views or policies.

pattern, the cooperation among components that process activities (filters) are always intermediated by pipes, which work as elements of explicit communication.

Explicit communication services, also called messaging services, can be directly incorporated into object engines in order to facilitate the development of applications that follow the structure described above. Particularly, in the construction of certain functions of communication protocols, it might be convenient to use this kind of service. In order to provide a messaging service, object engines define an internal component called pipe, with which components may interact to request the transfer of messages. In that way, when using a messaging service, there are no more direct interactions among components but only between components and engine.

3. Object Engines

Object engines offer a set of basic services that include: component creation and destruction, generation and maintenance of unique identifiers for each component, creation and destruction of virtual processors, interaction among components, pipe creation and destruction, and messaging service through pipes. We pay special attention on two of these services that, besides their importance, are strongly related: the service of interaction between components and the messaging service through pipes.

Pipes are unidirectional and their interface is the result of a combination of the characteristics of a shared memory with the ones of an event handler. A pipe offers read and write operations that work similarly to a queue contained in a virtual storage area shared among components. Optionally, pipes can inform components of the presence of a message, similar to event handlers.

Pipes can be associated to QoS parameters, as will be presented in Section 4, when they are called *MediaPipes*. Some aspects of the use of pipes and MediaPipes can be found in [4,3].

The concept of an object engine as the central element of a virtual processing environment allows programmers to abstract from the dispersion of components. The same model can be used in different situations as, for example, the local communication among components representing protocol entities of adjacent layers, or else the horizontal communication among peer protocol entities of the same layer, or even the interaction among application components regardless of their distribution.

The components dispersion refers to a locality relationship, that is, the distribution of components within the virtual memory space of the environment. In a first dispersion level, components can be distributed into *zones*. Each virtual processor can access a single zone, although several virtual processors can access the same zone. In a second level, zones are distributed into *sites*. Components are disjoint areas of zones that, in turn, are disjoint areas within sites. Sites are distributed into *domains*. Sites can, however, belong to more than one domain. Domains can be recursively contained in other domains. A domain that only contains sites (i.e., that does not contain other domains) is called a terminal domain.

Based on those dispersion levels, we define different object engines and compositions. A *cluster* (virtual processing environment) is defined by a composition (called cluster component) whose subcomponents are contained in a single zone and for which there is an object engine (called cluster engine) that is unable to create new virtual processors and of making asynchronous calls.

A virtual processing environment is called a *capsule* when it is defined within a composition (called capsule component) whose subcomponents are contained in a single zone and its object engine is capable to create new virtual processors and of making either synchronous or asynchronous calls. The subcomponents of a capsule component can be cluster components or simple components.

All components of capsules defined within the same site are also components of a single virtual processing environment called *host*. As capsules, hosts are capable of creating virtual processors and making either synchronous or asynchronous calls.

A virtual processing environment whose components are spread over sites of a single domain is called a *layer*. Each layer engine must provide a messaging service through pipes. A layer engine may contain other layer environments. The enclosing layer engine and its external components are called the *higher layer*. The components of the enclosed environment are internal components of the higher layer. Enclosed layer environments are called *subnets*.

Pipe construction within layer environments will be accomplished through a recursive algorithm that results in structures like the one shown in Figure 1.

4. QoS Specification

In order to achieve flexibility, QoS parameters are defined as object classes whose content is not known or manipulated directly by the object engines. The QoS negotiation, provision, management and assurance, the organization of these mechanisms, as well as the cooperation among them are part of the definition of the Framework for QoS Provision in

Generic Processing and Communication Environments [6]. Thus, for each type of a defined QoS parameter, the mechanisms and involved strategies should also be defined according to this framework.

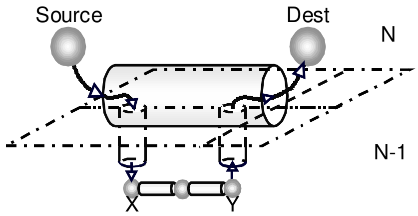


Figure 1: MediaPipes Within Layers

An object engine can be configured to work with specific QoS descriptions, with its own parameters and specialized mechanisms. New QoS parameter types can be dynamically defined by application programmers, provided that related mechanisms are also defined following the rules dictated by the framework and adequately configured into the engine by applications.

QoS parameters can be associated to pipes to provide control over the cooperation and exchange of messages among producers and consumers. A series of scheduling and internal communication requirements can be controlled through the creation of MediaPipes.

Given the recursive nature of the algorithm for the creation of pipes, QoS parameters will also be recursively distributed through the pipe segments that compose the higher pipes (see Figure 1). When MediaPipes are defined within clusters and capsules, as the vertical ones in Figure 1, virtual processors scheduling mechanisms are responsible for QoS assurance. For MediaPipes defined within layer environments, part of this responsability is delegated to the vertical MediaPipes and part is delegated to MediaPipes within the subnets (which are themselves layers). We suppose that there is an adequate mechanism that permits the mapping from the higher Layer QoS parameters to the lower ones used by vertical and subnet MediaPipes. This mapping mechanism is part of the QoS parameters framework presented in [6]. We also assume that there is a primitive subnet whose internal structure is unknown, but which is capable of assuring some QoS level. Operation definitions can also be the targets of QoS specifications. Operation signatures may be extended to attach QoS parameters to each operation parameter, return value and request message. Depending on operation signatures, one or more MediaPipes have to be established in order to provide the desired QoS.

5. Conclusions

We have presented a conceptual object-oriented model for the programming of multimedia services that can be dynamically adapted to specific QoS requirements of applications. We have defined a virtual processing environment concept that generalizes and abstracts specific object-oriented run-time environments. We have shown how these environments can be constructed and how QoS parameters can be introduced into them. A key characteristic of the model is its capacity to support the programming of actual end-to-end QoS guarantees. This characteristic is achieved as a direct result of QoS specifications being made on cooperation patterns. Since cooperation is present in the application level as well as in the architecture of processing environments, QoS can be assured along the whole path between any two components. This is a point of potential failure in some QoS architectures, notably the ones that define a specific layer for handling end-to-end QoS guarantees. In these architectures, it can be difficult to assure that QoS will be preserved for users of those layers, which might be even located on a different process.

References

[1] H. Berndt, P. Graubmann, and M. Wakano. Service specification concepts in TINA-C. In *International Conference on Intelligence in Broadband Services and Networks*, 1994.

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *A System of Patterns*. Wiley, 1995.

[3] S. Colcher and L. F. G. Soares. Modelo de Referência Unificado para Arquitetura de Protocolos e Programação de Aplicações Multimídia. In J. Leite and E. S. Silva, editors, *XVI Simpósio Brasileiro de Redes de Computadores (SBRC'98)*, pages 631–650, Rio de Janeiro, RJ, May 1998.

[4] S. Colcher, L. F. G. Soares, M. A. Casanova, and G. L. Souza. Modelo de objetos baseado no CORBA para sistemas hipermídia abertos com garantias de sincronização. In M. Oliveira, editor, *XIV Simpósio Brasileiro de Redes de Computadores (SBRC)*, pages 18–37, Fortaleza, CE, May 1996.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[6] A.T.A.Gomes and L. F. G. Soares. Framework para provisão de QoS em ambientes genéricos de processamento e comunicação. Technical report, Telemídia Lab, Catholic University of Rio de Janeiro – Brazil, 1998.

[7] ISO/IEC JTC1/SC21/WG7. ISO/IEC DIS 10746-1/ITU-T X.901 — Reference Model of Open Distributed Processing, Part 1: Overview, May 1995. output from the editing meeting in Helsink (Finland).

[8] Object Management Group (OMG). The Common Object Request Broker: Architecture and Specification, Feb. 1998. Revision 2.2.

[9] W. Pree. *Design Patterns for Object-Oriented Software Development*. Addison-Wesley/ACM Press, 1995.

Antônio Tadeu A. Gomes, Luiz Fernando G. Soares
{atagomes,lfgs}@inf.puc-rio.br

Laboratório Telemídia, Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introdução

Nos últimos anos, tem-se observado uma crescente demanda por *sistemas multimídia*, graças à utilização cada vez maior de *aplicações multimídia*, como videoconferência, correio multimídia, etc. A diversidade de requisitos impostos pelas diferentes mídias é um problema em voga na área de sistemas distribuídos. Por exemplo, enquanto mídias contínuas (áudio e vídeo) impõem requisitos relativos ao tempo de processamento e comunicação, mídias discretas (imagem estática e texto) impõe outros relativos à tolerância a perdas [1].

É trivial para um sistema oferecer a *qualidade* de processamento e comunicação requerida por uma mídia específica se recursos dedicados são usados, como acontece nos sistemas telefônicos atuais. Porém, devido a restrições gerenciais e de custos, há uma necessidade cada vez maior dos sistemas terem recursos eficientemente *compartilhados* por diferentes mídias, através do oferecimento de diferentes *serviços* às aplicações [2]. É importante notar que os requisitos impostos pelas mídias a um sistema são *fim-a-fim*, ou seja, todos os seus *subsistemas* devem ter ser seus recursos *orquestrados* de modo que o serviço como um todo possa ser corretamente fornecido (vide Figura 2). Tanto a orquestração quanto o compartilhamento eficiente de recursos só são viáveis se as aplicações indicarem a qualidade de serviço (*QoS*) desejada ao sistema, para que este possa provê-la corretamente.

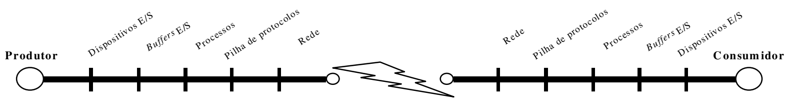


Figura 2: cenário de um sistema multimídia.

1.1. Soluções Presentes

Até recentemente, boa parte do desenvolvimento relacionado ao suporte à provisão de QoS ocorreu no contexto de subsistemas isolados, gerando sistemas incompletos, pouco integrados e muito específicos [5]. Por isso, vários grupos de trabalho têm proposto novas abordagens de arquitetura de sistema como solução para o problema da provisão de QoS fim-a-fim. *Arquiteturas de QoS* [3] definem conjuntos de interfaces que formalizam o conceito de QoS nos diferentes subsistemas. A maior limitação dessa abordagem é a sua granulosidade de aplicação, pois a provisão da QoS é uniformizada somente na fronteira entre os subsistemas. As arquiteturas internas dos subsistemas ficam totalmente a cargo dos projetistas de sistema, o que é indesejável, visto que: (i) pode induzir o desenvolvimento de mecanismos de provisão de QoS que atendam a todos os possíveis tipos de requisitos em cada subsistema, tornando-os complexos e pouco flexíveis; (ii) dificulta a reutilização de mecanismos de provisão de QoS potencialmente presentes em subsistemas distintos; e (iii) não modela a orquestração de recursos entre subsistemas.

Outra abordagem considerada em recentes trabalhos é a do oferecimento de um *único* serviço configurável de acordo com a necessidade das aplicações [6,7]. Tais trabalhos caracterizam-se pela definição de *ambientes de processamento e comunicação* compostos por pequenos componentes OO reutilizáveis que, conectados entre si, implementam desde subsistemas específicos até sistemas completos. É com base nessa abordagem que o *Framework Para Provisão de QoS*[5] é proposto. A granulosidade do *framework*, mais fina do que a das arquiteturas de QoS, possibilita: (i) a sua aplicação independentemente das diversas escalas de distribuição e níveis de abstração dos componentes de um subsistema[6]; (ii) a definição de sistemas altamente flexíveis no que concerne à provisão de QoS, através da configuração de alguns poucos

[5] Este framework engaja-se no projeto RAVel (convênio de pesquisa entre a PUC-Rio e a EMBRATEL), cujo objetivo principal é dar suporte à construção de arquiteturas de protocolos e aplicações multimídia, e que inclui também um *Framework* Para Provisão de *Multicast* [10] e um Modelo de Referência Unificado Para Arquiteturas de Protocolos e Programação de Aplicações Multimídia[7].

[6] Enquanto *distribuição* refere-se à localização relativa de componentes (em um mesmo processo, em processos distintos de uma mesma máquina, em máquinas distintas de uma mesma rede), *abstração* refere-se à visão que se tem desses componentes (seres humanos, aplicações multimídia e dispositivos de E/S quaisquer são exemplos de produtores/consumidores de dados em diferentes níveis de abstração).

mecanismos adaptáveis e reutilizáveis; e (iii) a modelagem da orquestração de recursos entre subsistemas, graças à sua estruturação recursiva em diferentes escalas de distribuição.

1.2. Descrição do Framework

Um modelo genérico de operação de um sistema que aceita requisições de usuários (produtores/consumidores de dados) e provê mecanismos de provisão de QoS pode ser dividido nas seguintes fases:

1. *Iniciação*: fase de definição da infra-estrutura que dará suporte aos serviços oferecidos, incluindo-se aí inventários de recursos disponíveis e configuração dos mecanismos de provisão de QoS.
2. *Requisição*: fase na qual os usuários caracterizam a carga[7] de processamento e comunicação a ser produzida ou consumida por eles, e especificam a QoS desejada.
3. *Estabelecimento*: fase em que o sistema primeiramente verifica se há recursos livres suficientes para manter a QoS especificada pelos usuários, e posteriormente aloca os recursos necessários às suas aplicações, ou negocia com os mesmos uma outra QoS possível de ser atendida.
4. *Manutenção*: fase na qual o sistema escalona seus recursos, objetivando otimizar a sua utilização, e tenta manter a QoS negociada com os usuários, policiando seus fluxos de dados e monitorando a QoS oferecida periodicamente.

A idéia de se definir o *framework* proposto nasceu da percepção de que, em todas as fases de provisão de QoS acima apresentadas, há padrões de estrutura e de comportamento que independem da escala de distribuição e do nível de abstração do sistema. Nesse sentido, a abordagem de *design patterns* [8,9] permite a descrição de *frameworks* de forma mais abstrata do que o código correspondente que os implementaria. O *framework* proposto utiliza-se dessa abordagem, ocultando do projetista detalhes de implementação e distribuição de componentes, ajudando-o a adaptar o *framework* mais facilmente às suas necessidades. Alguns dos *patterns* relativos à provisão de QoS definidos no *framework* podem ser vistos na Figura 3, sendo sucintamente apresentados adiante.

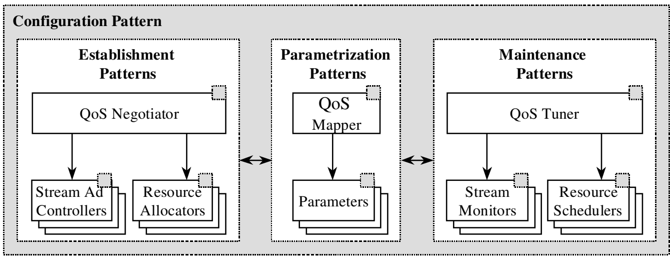


Figura 3: *patterns* definidos no *framework*.

1.3. Patterns de Parametrização de Serviços

Mecanismos de provisão de QoS são essencialmente modelados segundo: (i) a descrição da infra-estrutura existente, isto é, do *estado interno atual* do sistema; (ii) a caracterização da carga; e (iii) a especificação da QoS. Todas essas informações podem ser estruturadas através de *parâmetros*. O propósito dos *patterns* de parametrização de serviços é definir a estruturação desses parâmetros. Tais *patterns* incluem a definição de dependências entre parâmetros, que são úteis, por exemplo, no mapeamento da QoS especificada pelo usuário em parâmetros entendidos por mecanismos presentes nas diferentes escalas de distribuição de um sistema multimídia distribuído.

1.4. Patterns de Estabelecimento de Contratos

O propósito desses *patterns* é definir mecanismos que criem condições para que *contratos de serviço* possam ser negociados entre o usuário, o sistema, e possivelmente outros usuários (no caso de um serviço de comunicação). Além de (re)negociarem contratos entre usuários e o sistema, os *mecanismos de (re)negociação da QoS*[8] desempenham um papel especial, por serem eles os responsáveis por mediar a orquestração de recursos durante a fase de estabelecimento. Os *mecanismos de controle de admissão* são responsáveis por testar se o sistema tem recursos suficientes para manter a QoS desejada pelo usuário, e se a admissão do fluxo do mesmo trará algum impacto na QoS de fluxos previamente admitidos. Os *mecanismos de alocação de recursos* cedem aos fluxos parcelas de utilização de recursos (*recursos virtuais*) que possibilitam a satisfação da QoS especificada.

[7] *Carga* refere-se à dinâmica do *fluxo de dados* gerado entre os usuários e o sistema, e que está associado ao fornecimento do serviço.

[8] Durante a prestação do serviço, mudanças na QoS podem ser necessárias. Em um exemplo típico, um usuário inicialmente requer um serviço de vídeo sob demanda de alta definição. Como ele é cobrado com base na QoS, ele pode posteriormente decidir reduzir a qualidade do vídeo. A renegociação da QoS pode ser provocada também pelo fornecedor, como por exemplo na ocorrência de erros que afetem a QoS.

1.5. Patterns de Manutenção de Contrato

O propósito desses *patterns* é definir mecanismos que façam com que tanto usuários quanto o sistema respeitem o contrato estabelecido, e que iniciem ações apropriadas caso isso não ocorra. Os *mecanismos de escalonamento* controlam o compartilhamento de recursos entre os diversos recursos virtuais criados durante a alocação de recursos, mantendo a QoS negociada para cada fluxo. Os *mecanismos de monitorização de fluxos* registram violações no contrato, podendo enviar esses registros aos usuários de interesse sob a forma de *alertas* [4], ou então acionar os mecanismos de renegociação ou sintonização da QoS, dependendo da *política de provisão de QoS*[9] adotada. Os *mecanismos de sintonização* são responsáveis por manter a orquestração de recursos definida durante a negociação da QoS, sem que haja a necessidade de interrupção (renegociação) do serviço em caso de violação do contrato.

1.6. Pattern de Configuração

Esse *pattern* visa organizar a configuração conjunta dos mecanismos de provisão de QoS, de acordo com os parâmetros escolhidos durante a definição dos serviços a serem oferecidos e da política a ser adotada.

2. Cenário de Utilização do Framework

A Figura 4 ilustra um caso de uso do *framework* no projeto de um sistema de transmissão de mídias contínuas (em desenvolvimento) entre estações rodando o sistema operacional SOLARIS 2.5, conectadas por uma rede ATM.

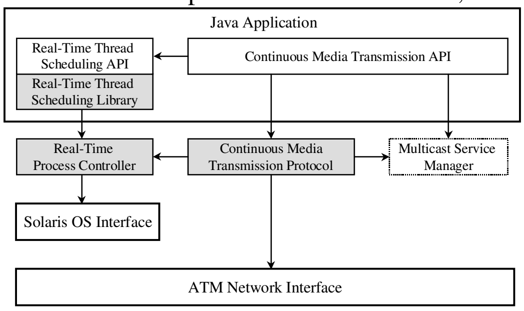


Figura 4: caso de uso do framework

Os seguintes elementos do sistema (hachurados na figura) correspondem a utilizações distintas do *framework*:
1. A biblioteca de escalonamento de *threads* de tempo real, sendo implementada em Java, responsável pela provisão de QoS a nível de *threads*, através de um controle de escalonamento das mesmas.
2. O controlador de processos de tempo real, sendo implementado em C++, responsável pela provisão de QoS a nível de processos, através de um controle de prioridade dos mesmos.
3. O protocolo de transmissão de mídias contínuas, sendo implementado em C++, responsável pela provisão de QoS a nível de rede, através do uso de uma interface de acesso à rede ATM nas estações.

Referências

[1] SOARES, L.F.G., LEMOS, G., COLCHER, S. *Redes de Computadores: das LANs, MANs e WANs às Redes ATM*. Campus, 1995.

[2] LU, G. *Communication and Computing for Distributed Multimedia Systems*. Artech House, 1996.

[3] CAMPBELL, A., et al. *A Review of QoS Architectures*. In: ACM Multimedia Systems Journal, Nov de 1995.

[4] International Standardization Organization. *ISO/IEC JTC1/SC 21 N9309 – QoS: Basic Framework*. 1995.

[5] HUTCHISON, D., et al. *Quality of Service Management in Distributed Systems*. 1994.

[6] LAZAR, A. A., LIM, K. S., MARCONCINI F. *Binding Model: Motivation and Description*. 1995.

[7] COLCHER, S., SOARES, L. F. G. *Modelo de Referência Unificado para Arquitetura de Protocolos e Programação de Aplicações Multimídia*. In: Anais/16º Simpósio Brasileiro de Redes de Computadores, 1998.

[8] GAMMA, E., et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[9] PREE, W. *Design Patterns for Object-Oriented Software Development*. Addison Wesley, 1995.

[10] RODRIGUES, M. A. A., SOARES, L. F. G. *Framework Para Provisão de Serviço de Multicast em Ambientes Genéricos de Comunicação*. Relatório Técnico. Laboratório Telemídia. 1998.

[9] Uma política de provisão de QoS é um conjunto de leis, regras e práticas que regulam a maneira como um fornecedor controla e gerencia a QoS por ele provida. A política de provisão de QoS usada pelo fornecedor está intimamente relacionada à sua implementação real.

Frameworks na Medição de Software

Arndt von Staa, Luiz Paulo Alves Franca
{arndt, franca}@inf.puc-rio.br

Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introdução

Na área de Medição de Software, o conceito de *Framework* vem sendo utilizado no contexo de metodologia (/*guideline*), mas recentemente passou a ser utilizado como sendo um artefato de design/implementação de sistemas Orientados a Objetos. Este artigo apresenta uma visão geral da aplicação do conceito de *Framework* dentro da área de Medição de Software. Dentre as aplicações destacaremos a utilização de *Framework*s para Programas de Medição, que é o tipo de aplicação que estamos mais envolvidos no trabalho de doutorado.

2. Framework para Validação da Medição

Devido a inexistência de um modelo de validação amplamente aceito, Kitchenham et al. [KLF95] sugerem uma proposta de *Framework* para Validação da Medição. O *Framework* proposto é baseado na identificação dos elementos de medição e de suas propriedades, e na definição de métodos teóricos e empíricos apropriados para validação. O *Framework* está dividido em 6 modelos: Estrutural da Medição, Relacionamento de Atributos, Unidade, Instrumentação, Protocolo, e Populacional. O principal modelo é o Modelo Estrutural da Medição que representa os componentes da medição e do processo de medição. Os demais modelos são utilizados para definir os elementos do modelo estrutural quando uma medição é realizada. Em relação ao Modelo Estrutural são apresentados versões paras as medições realizadas de forma direta e para as métricas calculadas indiretamente.

Mas recentemente, Zuse em [Zus98] aprofunda o tratamento formal para o *Framework* para Validação de Métricas. Buscando padronizar a validação de métricas para sistemas Orientados a Objetos, Briand et al. [BDW96, BDW97] propuseram *Framework*s para as métricas de Coesão e de Acomplamento dentro do contexto da Orientação a Objetos.

3. Framework para Avaliação de Métodos/Ferramentas

No sentido, de evitar problemas no processo de avaliação, Kitchenham [Kit96a, Kit96b, Kit96c, Kit96d] propôs uma *Framework* para Avaliação de Método/Ferramenta. Este Framewok procura definir critérios de seleção para as diferentes formas de avaliação, levando em consideração restrições e os impactos do tipo sócio-culturais. A tabela 1 apresenta uma descrição dos elementos do *Framework* para Avaliação de Métodos/Ferramentas.

TIPOS DE AVALIAÇÃO	**Quantitativo: Experimento Estudo de Caso Survey Benchmarking** **Análise de Propriedades: Experimento Estudo de Caso Survey** **Análise Qualitativa de Efeitos**
CRITÉRIOS DE SELEÇÃO	**Contexto da Avaliação Natureza do Impacto** **Natureza do Objeto Avaliado Escopo do Impacto** **Maturidade do Objeto Tempo de Aprendizado** **Maturidade de Avaliação da Organização**
RESTRIÇÕES	**Duração Custo Confiança**
EFEITOS SOCIOLÓGICOS	**Curva de Aprendizado Observação Expectativa**

Tabela 1. Elementos do *Framework* de Avaliação de Métodos/Ferramentas

4. Framework para Condução de Experimentos Formais

Com o objetivo de garantir que os resultados de um experimento não sofram interferências que invalidem o próprio experimento, Fenton [FPG94] propôs um *Framework* para Condução de Experimentos Formais (Tabela 2).

CONCEPÇÃO	**Definição clara e precisa do objetivo do estudo**
DESIGN	**Hipóteses Sujeitos Objetos de Controle** **Variáveis Corridas ('trials")**
PREPARAÇAO	**Infra-estrtutura Treinamento**
EXECUÇÃO	**Execução do experimento Monitoramento**
ANÁLISE	**Validação das medições Análise estatítica**

Tabela 2. Visão Esquemática do *Framework* de Condução de Experimento

Kitchenham et al. em [KPP95] sugerem uma proposta para a condução de avaliações através de Estudos de Caso. Destacamos o trabalho desenvolvido pelo ISERN (International Software Engineering Research Network), que é um grupo de pesquisa que procura viabilizar a replicação de experimentos [BDMRW96], a partir de sua normatização.

5. Frameworks para Programas de Medição de Software

No sentido de suplantar estas dificuldades, alguns sistemas de apoio ao Programa de Medição estão sendo desenvolvidos disponibilizando as seguintes facilidades:

- Escopo de medição flexível: As interfaces do sistema variam com o perfil de medição definido pela empresa;
- Integração com o ambiente de desenvolvimento: Procura-se a adotar a aplicação da coleta de dados não intrusiva. Em geral os sistemas apresentam um conjunto de drivers para importação/exportação de dados.
- Medição distribuída: O sistema de medição deve acompanhar as atividades de desenvolvimento, independente do local de sua realização (Ex.: site da empresa ou site do cliente).
- Incorporação da experiência de utilização: A adoção de um novo programa de medição pode ser inicializado a partir das configurações de programas de medições anteriores.

Além destas facilidades, este tipo de sistema deve ser capaz de permitir a introdução de novas métricas e de novas correlações, a base de métricas e correlações originais. Esta necessidade deve-se ao fato de ainda não existir padrões de medições amplamente aceitos, e devido a criação de métricas para aplicação em novas técnicas/métodos (ex: métricas para sistemas OO, métricas para sistemas baseados na Web).

Para a produção deste tipo de sistemas, algumas propostas estão sendo desenvolvidas através da utilização de *Frameworks*, dentre elas destacamos as ferramentas: "Squatter toolset" [OX96, OJ97], COMPASSO [FSL98].

6. Medição de Software para Frameworks

Na proposta de desenvolvimento de sistemas através de uma abordagem Orientada a Objetos, o conceito de *Framework* vem sendo empregado como sendo um artefato que representa um "design" que pode ser reutilizável [GHJV95]. Através da customização de suas classes abstratas e concretas, o *Framework* é preparado para resolver um problema especifico do domínio da aplicação do *Framework*. Espera-se com esta abordagem a produção de sistemas de forma mais rápida e que possam ser evoluídos mais facilmente. Na época da divulgação das propostas de desenvolvimento Orientado a Objetos, a área de Medição de Software procurou contribuir para solidificação desta abordagem, através da proposição de métricas que avaliassem a qualidade de sistemas Orientados a Objetos. De forma similar, a atenção dispensada ao desenvolvimento Orientado a Objetos, começam a surgir alguns trabalhos [SLB98, BCKW98, BMB98] sobre avaliação da utilização de *Frameworks* e dos sistemas desenvolvidos através desta nova técnica.

Referências

[BCKW98] Briand, L., Carriere, J., Kazman, R., Wuest, J., "COMPARE: A Comprehensive Framework for Architecture Evaluation", *ISERN technical report*, 98-28, 1998. http://www.iese.fhg.de/ISERN/pub/isern.biblio.html

[BDMRW96] Brooks, A., Daly, J., Miller, J., Roper, M., Wood, M., "Replication of experimental results in software engineering", *ISERN technical report*, 96-10, 1996. http://www.iese.fhg.de/ISERN/pub/isern.biblio.html

[BDPW98] Briand, L., Daly, J., Porter, V., Wuest, J., "A Comprehensive Empirical Validation of Product Measures for Object-Oriented Systems", *ISERN technical report*, 98-07, 1998. http://www.iese.fhg.de/ISERN/pub/isern.biblio.html

[BDW96] Briand, L., Daly, J., Wuest, J., "A Unified Framework for Coupling Measurement in Object-Oriented Systems", *ISERN technical report*, 96-14, 1996. http://www.iese.fhg.de/ISERN/pub/isern.biblio.html

[BDW97] Briand, L., Daly, J., Wuest, J., "A Unified Framework for Coupling Measurement in Object-Oriented Systems", *ISERN technical report*, 97-05, 1997. http://www.iese.fhg.de/ISERN/pub/isern.biblio.html

[BMB98] Briand, L., Morasca, S., Basili, V., "Defining and Validating Measures for Object-Based High-Level Design", *ISERN technical report*, 98-04, 1998. http://www.iese.fhg.de/ISERN/pub/isern.biblio.html

[FPG94] Fenton, N., Pfleeger, S.L. and Glass, R., "Science and Substance: A Challenge to Software Engineers", *IEEE Software*, 11(4), 1994, pp. 86-95.

[FP96] Fenton, N. and Pfleeger, S.L. , *Software Metrics: A Rigorous & Practical Approach*, ITP, Cambridge, 2° edition, 1996.

[FSL98] Franca, L.P.A., Staa, A., Lucena, C.J.P., "Medição de Software para Pequenas Empresas: Uma Solução Baseada na Web", Anais do XII Simpósio Brasileiro de Engenharia de Software, Ed. SBC, 1998, pp. 71-85.

[GHJV95] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Kit96a] Kitchenham, B., "Evaluating Software Engineering Methods and Tool", Software Engineering Notes vol 21 , no. 1, 1996. pp. 11-15.

[Kit96b] Kitchenham, B., "Evaluating Software Engineering Methods and Tool", Software Engineering Notes vol 21 , no. 2, 1996, pp. 11-15.

[Kit96c] Kitchenham, B., "Evaluating Software Engineering Methods and Tool", Software Engineering Notes vol 21 , no. 4, 1996, pp. 9-12.

[Kit96d] Kitchenham, B., "Evaluating Software Engineering Methods and Tool", Software Engineering Notes vol 21 , no. 5, 1996, pp. 11-13.

[KPF95] Kitchenham, B., Pfleeger, S.L., and Fenton, N., "Towards a Framework for Software Measurement Validation", IEEE Transactions on Software Engineering, 21(12),1995, pp. 929-944.

[KPP95] Kitchenham, B., Pickard, L., Pfleeger, S.L., "Case studies for method and tool evaluation", IEEE Software, 12(4),1995, pp. 55-62.

[OJ97] Offen, R.J. and Jeffery, R. "Establishing Software Measurement Programs", IEEE Software, 14(2), 1997, pp. 45-53.

[OX96] Offen, R.J. and Xia, G., "SQUATter-M Conceptual Requirements Document",JRCASE Research Report, No. 96/13.

[SLB98] Shull, F., Lanubile, F., Basili, V., "Investigating Reading Techniques for Framework Learning", *ISERN technical report*, 98-16, 1998. http://www.iese.fhg.de/ISERN/pub/isern.biblio.html

[Zus98] Zuse,H., *A Framework of Software Measurement*, Walter de Gruyer, 1998.

Arquitetura Orientada a Objeto para Integração de Facilidades de Persistência, Distribuição, Concorrência e Transação

Maria Alice Silveira de Brito, Sergio E. R. Carvalho, Rubens N. Mello

Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introdução

Nossa motivação neste trabalho é a de conseguir integrar as facilidades de persistência, distribuição, concorrência, e transação em um sistema de programação orientada a objeto, explorando os aspectos de tipos abstratos de dados e de herança.
Surgiram diferentes iniciativas para a integração dessas facilidades em linguagens de programação orientada a objeto, enquadrando-se em um dos três principais modelos de arquitetura, a seguir: 1) *linguagens de programação orientada a objeto com facilidades de bancos de dados*, como, por exemplo, O2 [LRV88], GemStone [MS97], ITASCA (ORION) [GK88] e ObjectStore, entre outros; 2) *linguagens de programação orientada a objeto concorrente*. Como, por exemplo, POOL-T [Ame87], Hibrid [Nie92], DRAGOON [Atk91], Trellis/Owl [MK87] e ABCL/1 [YBS86], SINA [AWB+94] entre outras; e 3) *ambiente de execução transacional*, como, por exemplo, Argus [WL85], Avalon/C++ (sobre Camelot) [DHW88], Arjuna [Gra90], Amadeus/ RelaX [TCM94], entre outros.
Embora as facilidades que procuramos se encontrem, nessas três arquiteturas, ou dispersas, ou ainda com potencial de deslocamento das gerências centralizadas, consideramos suas contribuições valiosas para a arquitetura que pretendemos.
Com base nessa avaliação, o nosso modelo de integração reúne as quatro facilidades: persistência, distribuição, concorrência, e transação, aproveitando as contribuições dos modelos acima, e, ao mesmo tempo, traz para os objetos da aplicação as tarefas localizadas nos gerentes centralizados, que tenham possibilidade de deslocamento.

2. Nossas Contribuições

Durante o processo de aproveitamento das contribuições existentes para o nosso modelo, conseguimos reorganizar as facilidades e seus aspectos, em relação aos conceitos envolvidos, aos seus paradigmas, aos seus papéis principais e cooperativos. Essa reorganização nos trouxe as contribuições enumeradas, a seguir:

1) **Nosso modelo de arquitetura**, resumidamente, envolve três componentes principais: a classe gerente de objetos; a classe transação; e a classe de objeto de aplicação. Distribuímos os aspectos das facilidades que estamos integrando por essas três classes, sendo que o peso maior coube aos objetos da aplicação, concentrado em suas tarefas de tratamento de mensagens, no monitor. Também reduzimos significativamente as tarefas do gerente de objetos e deslocamos praticamente toda a gerência de uma transação para a classe transação.

2) **Transações como objetos programáveis em classes,** essa facilidade foi inspirada no *deslocamento do controle de concorrência* das linguagens de programação orientada a objeto concorrente e dos ambientes de execução transacionais. Percebemos a chance de deslocar também a gerência de uma transação para o espaço da programação, localizada em uma classe transação. Um objeto dessa classe reuniria a mesma habilidade de *administração de todos os recursos envolvidos na transação*, encontrada nos gerentes de transações centralizados dos ambientes de execução transacional[10], reforçando o encapsulamento ao deslocar para um objeto criado pela aplicação mais este controle dos recursos envolvidos em uma transação, antes localizado nos gerentes centralizados. Ao mesmo tempo, ela oferece opções ao programador de desenvolver outras classes de transação, tornando a gerência de transação mais flexível, com capacidade de absorção de certas diferenças que possam surgir nas aplicações, sendo um dos hotspots encontrados em nosso modelo.

3) **Mensagem isolada com garantia das propriedades ACID.** Introduzimos uma abordagem diferente em relação aos limites de uma transação em programação orientada a objeto. Podemos notar que uma simples mensagem localizada no objeto cliente *top-level* da aplicação e desvinculada de transação, ao ser enviada a um objeto, pode propagar-se em muitas outras mensagens a outros objetos, devido aos relacionamentos de composição. Percebemos que essa mensagem isolada poderia assumir proporções que exigissem uma garantia de atomicidade, quer dizer, ou todas as operações envolvidas deveriam ser completamente efetuadas, ou, em caso de conflito ou falha, completamente desfeitas. Para as mensagens localizadas no objeto cliente top-level, que estejam desvinculadas de transações explícitas, criamos uma transação implícita.

---
[10] Esta habilidade também se encontra no conjunto de serviços de transação proposto no padrão OMG.

4) **Verificação de comportamento de uma mensagem assíncrona.** Esta facilidade é resolvida de uma forma mais flexível do que as outras soluções existentes[11], preservando os modos (assíncronos e síncronos) de qualquer mensagem enviada.

5) **Extensão de facilidades de depuração e recuperação, favorecida pelas transações programáveis e pela combinação do retorno/envio**. Essa facilidade é alcançada pelo registro das mensagens nas transações. Essas informações podem ser úteis em depuração simples, ou em depuração com objetivos de recuperação, principalmente em sistemas distribuídos, quando algum nó falhar.

3. Conclusões

Conseguimos garantir o encapsulamento dos objetos da aplicação, empregando, principalmente, os conceitos de objetos ativos das linguagens de programação orientada a objeto concorrente e o de tipo atômico dos ambientes de execução transacionais. Programamos os componentes de nossa arquitetura, segundo a orientação a objeto, com o compromisso de explorar os aspectos de tipos abstratos de dados e de herança em todos os níveis, tornando-os modulares e flexíveis, ampliando o número de beneficiários de uma framework pautada nessa nova arquitetura.

Os principais beneficiários são as próprias linguagens de programação orientada a objeto seqüencial, que podem ser transformadas em linguagens de programação orientada a objeto com facilidades de persistência, distribuição, concorrência e transação com a *inclusão de uma simples chamada ao monitor* do próprio objeto, no código gerado para o envio de mensagem e para o retorno de método.

Além das linguagens seqüenciais, uma framework assim pode beneficiar os seguintes usuários:
1. os programadores de transações;
2. os programadores da aplicação, que precisam intervir no comportamento da concorrência no objeto, por exemplo, otimizando as regras de validação para a sincronização de exclusão mútua, na consistência entre as mensagens;
3. os programadores de persistência e distribuição:
a) quando necessitam substituir uma nova plataforma que realize estes serviços, bastando reprogramar a interface para a nova plataforma;
b) quando precisam programar recuperação de falhas, contando com recursos oferecidos pela classe transação, que tem meios de auxiliar o diagnóstico se sua instância possuir algum objeto no nó da rede que falhou.

Referências

[Ame87] Pierre America. POOL-T: A Parallel Object-Oriented Language. *Object-Oriented Concurrent Programming*, A Yonezawa and M. Tokoro (Ed.), MIT Press, Cambridge, Mass., 1987.

[Atk91] Colin Atkinson. *Object-Oriented Reuse, Concurrency and Distribution* – AN ADA-BASED APPROACH. Addison-Wesley Publishing Company. ACM Press. New York, New York, 1991.

[AWB+94] Mehmet Aksit, Ken Wakita, Jan Bosh, Lodewijk Bergmans and Akinori Yonezawa. Abstracting Object Interactions Using Composition-Filters. Proceedings *ECOOP'93*, LNCS 791, Springer-Verlag, 1994.

[Cat94] R.G.G. Cattel. *Object Data Management*. Addison-Wesley Publishing Company. 1994.

[CLS96] S. J. Caughey, M. C. Little, and S. K. Shrivastava. Checked Transactions in na Asynchronous Message Passing Environment. Technical Report — Department of Computing Science, University of Newcastle, Newcastle upon Tyne, NE1 7ru, UK. 1996.

[DHW88] David L. Detlefs, Maurice Herlihy, and Jeannette M. Wing. Inheritance of Synchronization and Recovery Properties in Avalon/C++. *IEEE Computer*. December 1988.

[Gra90] Grahan Parrington. Reliable Distributed programming in C++: The Arjuna Approach. In *Proceedings of USENIX/C++ Conf.*, San Francisco, April 1990.

[GK88] Jorge F. Garza, and Won Kim. Transaction Management in an Object-Oriented Database System. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*. 1988.

[LRV88] Christophe Lécluse, Philippe Richard, and Fernando Velez. O2, an Object-Oriented Data Model. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*. 1988.

[MK87] J. Elliot B. Moss and Walter H. Kohler. Concurrency Features for the Trellis/Owl Language. Proceedings *ECOOP'87*, LNCS 276, Springer-Verlag, June 1987.

[MS87] D. Maier and J. Stein. Development and Implementation of an Object-Oriented DBMS. In B. Shriver and P. Wegner, eds. *Research Directions in Object-Oriented Programming*, MIT Press. Cambridge, MA. 1987.

[Nie92] Oscar Niertrasz. A Tour of *Hybrid* – A Language for Programming with Active Objects. *Advances in Object-Oriented Software Engineering*. ed. D. Mandrioli and B. Meyer, Prentice-Hall, 1992.

[TCM94] Paul Taylor, Vinny Cahill and Michael Mock. Combining Object-Oriented Systems and Open Transaction Processing. *The Computer Journal*, 37(6), 1994.

[WL85] Willian Weihl and Barbara Liskov. Implementation of Resilient, Atomic Data Types. *ACM Transactions on Programming Languages and Systems*, 7(2):244-269, 1985.

[YBS+86] Akinori Yonezawa, Etsuya Shibayama, T.Takada and Yasuaki Honda. Modelling and Programming in an Object-Oriented Concurrent Language ABCL/1. In *Proceedings of the ACM Conference on Object-Oriented Programming, System, and Applications (OOPSLA'86)*. 1986.

[11] No modelo OMG, a verificação de comportamento é assegurada, permitindo apenas invocações síncronas ou *deferred synchronous* no interior de transações, enquanto que, em [CLS 96], os autores substituem dinamicamente todas as invocações assíncronas que ocorrem no interior de uma transação por *deferred synchronous*.

Processo de Classificação em Arquitetura de Software

Marcelo Cabral, Júlio C. Leite

Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introdução

Em meados dos anos 70 Warnier [1], Jackson [2], Yourdon [3], Constantine [3], entre outros, perceberam boas práticas comuns em programação e desenho físico de sistemas. Estas boas práticas foram percebidas de maneira tal que foram postas em padrões preestabelecidos e respectivos métodos e ferramentas foram propostos para apoiar estes padrões. Os padrões, os métodos e as ferramentas foram postos no ciclo de desenvolvimento de sistemas de tal forma que era possível comparar o sistema em desenvolvimento ao padrão proposto. Assim, era verificado o que poderia ser melhorado no produto a ser desenvolvido ao mesmo tempo que fornecia um guia inicial para a construção do sistema (ou programa) em questão.

Atualmente passamos por um processo semelhante, porém no que vem sendo chamado de *Arquitetura de Software*. Em um nível mais alto de abstração que o proposto nos anos 70, os estudiosos deste assunto vêm se preocupando em identificar estilos de soluções e suas características comuns. Estando estes estilos bem identificados e suas características bem definidas, passa-se a poder compará-las com sistemas em desenvolvimento (ou já existentes). Com base na taxonomia preexistente (proveniente dos estilos e características), podemos sugerir melhorias no produto a ser desenvolvido (ou já existente).

Os esforços em Arquitetura de Software tem se concentrado no processo de taxonomia e definição de linguagens formais ou semi-formais para a representação das arquiteturas. O presente trabalho estará abordando apenas a parte de taxonomia, ou seja, quais são as arquiteturas existentes e quais são suas respectivas características.
Quando se fala em características aparece uma outra questão, ainda não definida, que é a granularidade do problema a ser tratado. Em Arquitetura de Software temos a unidade *componente* que pode ser em programa, um subsistema ou até mesmo um sistema completo. Porém, independente da granularidade, o que estamos buscando é a estrutura comportamental deste *componente*.

Algumas propostas tem ocorrido neste assunto, com especial destaque para os trabalhos feitos na Universidade Carnegie Mellon [4]. O objetivo deste trabalho consiste em apresentar uma taxonomia (estilos de arquiteturas e respectivas características), posteriormente submeter um produto em desenvolvimento (ou já existente) a um processo classificatório, verificar se este pertence a algum estilo definido e com base nas características do estilo, ao qual o produto pertence, verificar a falta de alguma característica no produto em avaliação.

Cabe ressaltar que as heurísticas resultantes deste trabalho provêm do estudo da bibliografia corrente sobre arquitetura ([4], [5], [7]) e da experiência prática do grupo que participou da elaboração deste trabalho.

Referências

[1] Warnier, Jean-Dominique. "Logic Construction of Programs". Leiden: Holanda, H.E. Stenfert Kroese BV, 1974

[2] Jackson Michael A "Principles of Program Desing". Londres: Academic Press, 1975

[3] Yourdon, Ed. e Constantine, L. L.: "Structure Design: Fundamentals of a Discipline of Computer Programming and System Design", 2ª edição. New York: Yourdon Press, 1978

[4] Shaw, Mary e Garlan, David : "Software Architecture – Perspectives on a Emerging Discipline", 1996

[5] Shaw, Mary e Clements, Paul : "A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems", April 1996

[6] Formalizing Style to Understand Descriptions of Software Architecture. ACM Transactions on Software Engineering and Methodology, 4(4): 319-364 October 1995

[7] Abd-Allah, Ahmed e Boehm, Barry, "Reasoning about the Composition of Heterogeneous Architectures" – USC Technical Report: USC-CSE-95-503, 1995

Uma Arquitetura Extensível e Portátil para Sistemas com Múltiplos Documentos

Antônio Scaño Scuri
scuri@tecgraf.puc-rio.br

Laboratório TecGraf, Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introdução

A partir da identificação de um problema de navegação em sistemas com múltiplos documentos, propomos uma possível solução, que utilizada em conjunto com um mecanismo de extensão baseado em bibliotecas dinâmicas carregadas em tempo de execução, nos levou a criação de uma arquitetura extensível e portátil para sistemas com múltiplos documentos, inclusive com suporte para múltiplos tipos de documentos.

2. Motivação

O desenvolvimento e a aplicação de técnicas de Processamento de Imagens Digitais requerem muitas vezes uma avaliação visual comparativa de resultados de operações sobre imagens. Em geral, o operador executa a mesma operação em uma ou mais imagens, repetidas vezes, variando alguns parâmetros que precisam ser ajustados para um resultado ótimo. Essa avaliação é comumente feita colocando-se lado a lado duas ou mais imagens. Assim, sistemas de Processamento de Imagens possuem necessariamente suporte para múltiplos documentos.

A navegação em sistemas de múltiplos documentos pode se tornar um processo penoso com a rápida proliferação de documentos abertos simultaneamente. Além disso, os sistemas atuais de interface com o usuário provem pouco suporte a uma navegação mais controlada, e em alguns casos não há suporte algum. Assim como os sistemas mais populares que possuem suporte são muito diferentes entre si.

3. MVC

Junto com a especificação de SmallTalk-80, foi proposta uma arquitetura para a gerência da interface de sistemas com documentos que possuíssem uma visualização e interação com o usuário através dessa visualização. Essa arquitetura foi chamada de MVC ou Model-View-Controller.

Várias bibliotecas voltadas para a criação de interface com o usuário se basearam nesta arquitetura para fornecerem suporte a documentos e visualizações, tais como a OWL (Object Windows Library) da Borland e a MFC (Microsoft Foundation Classes) da Microsoft.

Mas essa arquitetura deixa em aberto a questão de existirem múltiplos documentos simultaneamente a serem gerenciados e acessados pelo usuário.

4. MDI

Um dos poucos sistema de interface que trata múltiplos documentos de maneira diferenciada é o **Microsoft Windows**. Nele existe a definição de uma janela pai que contém todas as suas janelas filhas de forma que elas não ocupem uma área maior que a da janela pai. Esse mecanismo é chamado de MDI (*Multiple Document Interface*).

O primeiro grande problema de MDI é que quando uma janela filha é maximizada, ocupando toda área cliente da janela pai, não é possível saber se existem outras janelas abertas nem quais são. Uma confusão muito comum feita pelos usuário do editor de textos Microsoft Word é ignorarem o fato de que ele é uma aplicação MDI.

No sistema de interface do **Apple Macintosh** esse problema não acontece por que não existe o conceito de maximizar uma janela. Além do que, todas as aplicações são de uma certa forma MDI. Como o menu ocupa um lugar fixo na tela, cada aplicação pode ter quantas janelas filhas quiser.

Em ambos os sistemas de interface aparece um outro problema quando se está trabalhando com múltiplos documentos. Devido ao menu ficar na janela pai, usa-se o conceito de janela filha ativa para se efetuar a operação selecionada no menu. Quando se está dentro do diálogo da operação já não se sabe mais quem era a janela ativa, até por que ela deixa de ser a janela ativa para que o diálogo o seja. A não ser que o programador seja cuidadoso para todo diálogo de operação mencionar sobre quem ele irá operar (o que geralmente não acontece) o usuário pode facilmente operar sobre o documento errado.

Uma solução para isso é explicitar a lista de janelas filhas. O editor de textos **UltraEdit** faz exatamente isso. Contudo essas soluções ainda não resolvem o problema de termos mais de uma janela filha por documento.

O editor de textos FrontPage, por editar código HTML em WYSIWYG, permite três visualizações do mesmo documento, a edição em WYSIWYG, a edição em HTML e o *preview* de como ficaria no *Browser*, já que o WYSIWYG dele não é totalmente WYSIWYG. Mas são todas realizadas na mesma janela, e assim são mutuamente exclusivas, ou seja, só pode haver uma ativa de cada vez.

Nem sempre se pode restringir um sistema a visualizações mutuamente exclusivas.

5. Soluções Propostas

O que falta nos sistemas de interface com suporte para MDI é justamente a distinção entre os documentos que estão abertos e suas respectivas visualizações. A proposta deste trabalho é criar uma lista de documentos que ficaria visível na janela principal da aplicação e que conteria em um segundo nível, a lista de visualizações abertas de cada documento. Quando a janela de uma visualização é ativada, seu item na lista de documentos também seria realçada. Assim como, as janelas filhas não ficariam contidas na janela principal como no caso do MDI no Microsoft Windows.

Essa solução não só melhora o sistema de navegação para os diversos documentos abertos, como também resolve o problema de portabilidade da interface, já que pode ser implementado em sistemas nativos, seja no Macintosh, seja no Windows ou seja no X-Windows.

Neste momento observamos que essa solução pode receber qualquer tipo de documento. É de imediato que possamos complementá-la com um mecanismo de extensão. Observando que cada documento possui um ou mais modelos de dados, formas de armazenamento, formas de visualização e operações de processamento, cada uma dessas classes pode ser estendida o quanto for necessário. Se cada uma estiver associada a uma possível extensão, então temos uma arquitetura extensível e portátil para sistemas com múltiplos documentos.

6. Contribuição

Acreditamos que o ponto mais relevante desse trabalho seja a melhoria do mecanismo de navegação para sistemas com múltiplos documentos.

Devido a suas características esse mecanismo é portátil e permitiu associarmos um mecanismo de extensão criando uma nova arquitetura.

7. Trabalhos Futuros

Os possíveis mecanismos de explicitar as relações visuais entre o documento sendo processado e suas visualizações e operadores ainda estão em estudo e requerem uma validação através de um teste de usabilidade com um grupo de usuários.

Referências

[CONSTAN. 90] L. L. Constantine; *"Objects, Functions and Program Extensibility",* Computer Language 7, No.1 (January), p. 34-54; 1990.

[IERUSALIM. 96] R. Ierusalimschy, L. H. de Figueiredo and W. Celes, *"Lua - an Extensible Extension Language"*, Software: Practice & Experience 26 #6 635-652; 1996.

[BARKAKATI 91] N. Barkakati; *"Object Oriented Programming in C++"*; SAMS; 1991.

[GOLDBERG 90] A. Goldberg; *"Information Models, Views and Controllers"*; Dr. Dobb's Journal , (July) p. 54-61; 1990.

[BURBECK 92] S. Burbeck; *"Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)"*; http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

[LEVY 95] C. H. Levy, L. H. de Figueiredo, M. Gattass, C. J. P. Lucena, D. D. Cowan; *"IUP/LED: A Portable User Interface Development Tool"*; Software-Pratice and Experience; John Wiley & Sons, Ltd; 1995.

[APPLE 92] Apple Computer, Inc; *"Macintosh Human Interfaces Guidelines"*; Addison-Wesley; 1992.

[MICROSOFT 98.1] Microsoft Corp.; *"The Windows Interface Guidelines for Software Design"*; Microsoft Press; 1998.

[MICROSOFT 98.3] Microsoft Corp.; *"Microsoft Foundation Class Library"*; Microsoft Press; 1998.

[BORLAND 93] Borland International Inc. *"ObjectWindows for C++ 2.0, Programmer's Guide"*; 1993.

[BAXES 94] G. A. Baxes; *"Digital Image Processing (Principles and Applications)"*; John Wiley & Sons, Inc.; 1994.

[CASTLEMAN 95] K. R. Castleman; *"Digital Image Processing"*; Prentice-Hall, Inc.; 1995.

[GONZALEZ 92] R. C. Gonzalez, R. Woods; *"Digital Image Processing"*; Addison-Wesley; 1992.

VMARKET: A Framework Design for an Agent Mediated E-Commerce System Based on Virtual Marketplaces

Pedro S. Ripper
pedro@les.inf.puc-rio.br

Laboratório de Engenharia de Software, Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introduction

Software agent technology is still an emerging technology, and as such, agent based software design is still in its infancy. Software agents have just started to be used in the e-commerce domain, and they are already starting to create a series of new possibilities to this arena. Agents can be used to automate as well as to enhance many stages of the traditional consumer-buying behavior process. Through the minimization of transaction costs, elimination of geographic barriers and time issues, many new markets, not viable before, are now being created, traditional markets are being made more efficient, and the role of the middleman is drastically changing.

This paper proposes a software engineering approach to the design of an agent mediated e-commerce system, through the definition of a framework and its implementation. This framework will focus mainly in e-commerce applications based on virtual marketplaces. A virtual marketplace is nothing more than an internet based system, where software agents interact and negotiate, on behalf of their respective users, to buy or sell goods. In this type of system, all users are potential buyers or sellers depending on their specific interests.

The main goal of this framework is to facilitate the creation of this type of applications, as well as to make them more robust and flexible. It is expected that this approach will greatly enhance the process of experimentation and research on the new possibilities brought by software agents to the e-commerce application domain.

2. Problem Description

A software agent is neither a new concept nor a well-defined one. Although, not clearly defined, agents have been used for quite some time in many different fields of computer science, and depending on the field of study, its definition can vary. In this document, an agent is defined as a piece of software (not dependent on its implementation language) used to automate specific tasks. Also, to be considered as an agent, this piece of software needs to be proactive, to be able of personalization, and to have a certain level of autonomy. Other features such as mobility and collaboration might be desirable for some applications, but will not be considered as pre-requisites for this definition.

Given the exponential increase on information resources available in the Internet (World Wide Web), these software agents have been given a lot of attention lately. Agents have the distinguishing ability of automating repetitive and time consuming tasks, including those of searching, buying and selling products over the Internet. Most of the tasks involved in the consumer buying behavior process [1,2,3] can be automated. Stages such as needs identification, product brokering, merchant brokering and negotiation can now be assisted or automated by many different agent based systems. While merchants struggle to explore new channels to negotiate their goods, looking for opportunities to maximize their profits and at the same time to satisfy consumers, most of the electronic commerce stores available on the web today still take the form of static "catalogs" of products, where customers manually select items and purchase them online. So far, this model has fallen short on redefining the marketplace.

As mentioned before, agents are autonomous, proactive, customizable and sometimes mobile, and these capabilities have been used to overcome many barriers of the traditional commerce, such as geographic barriers, transaction costs and time, creating this way many new scenarios that were thought not possible before. It is expected that software agents will make existing markets more efficient [20], change the role of the middleman [4], and make many small niche markets viable. As a development technique, agent technology brings many new possibilities to the now reasonable well-established object-oriented technology. Being able to roam from machine to machine (mobility), take independent actions, and communicate among themselves to better accomplish their objectives, are all very powerful features. However, together with all these new capabilities comes a series of new considerations to the software development process (Agent-Oriented-Programming).

Most of the agent based e-commerce systems built so far, were developed in an ad-hoc basis with no software reuse in mind. Many of these systems share a very large set of functionality and requirements, and all these systems were built from the ground up. Each system created almost the same solutions for the same problems, reinventing them over and over again.

Although it's quite clear that there are a series of new possibilities brought by software agents to e-commerce applications, with the current generation of applications and tools, it is a very arduous task to explore and experiment these new scenarios. Despite the existence of a few agents based frameworks/toolkits that are available or still under development, such as ObjectSpace's Voyager [5], IBM's Aglets [6], General Magic's Odyssey [7] and Mitsubishi's Concordia[8], these frameworks/toolkits are only general-purpose solutions for software agent programming. They all provide some of the basic functionality for distributed agent programming, such as agent migration facilities, different message passing semantics, agents' life-span control mechanisms, and even some persistence options. However, there is still a lack of domain specific frameworks for agent based systems. Specifically at the e-commerce domain, there are a series of common design decisions and functionality shared by many systems that still have not being captured by any framework.

3. Proposed Solution

Based on a thorough study, and implementation of different agent mediated e-commerce systems, [10,12,13,14], I propose a more formal software engineering approach to the development of applications based on virtual marketplaces. Using a framework design method [15,16] and Object-oriented patterns [17], as well as the more recent Agent patterns [18], the resulting framework should facilitate the development of such applications, enabling its users to focus on the higher level aspects of agent mediation. The framework proposed is greatly inspired by Media Lab's Kasbah [14] system, and will concentrate mainly on the ability to create applications based on virtual marketplaces, where buying and selling agents interact. Its implementation will make use of lower level agent services provided by some of the agent toolkits mentioned earlier, and will build upon these tools to create a higher level design solution for this specific application domain. These applications should be easy to instantiate and customize, creating a powerful experimenting and research tool.

One of the most common problems in framework design is to find a balance between generality and ease of use. Normally, the more generic and flexible a framework is, the harder it is to use it, and most of all, the harder it is to build it. After participating on the development of some agent based e-commerce systems such as Kasbah, Tete-a-tete and the AmEC infrastructure [29], it is not hard to identify some specific points (hot spots [15]) in the design of these applications, that are critical to their flexibility. These specific "flexibility points" alone would allow the instantiation of a wide range of different applications, with flexible behavior, without requiring any fundamental change in their design. The main hot spots identified are:

- *Multiple item support, and Structured item ontology:*
  One of them main problems faced with the current implementation of Kasbah is that it is completely tied to the two types of goods that it now supports (i.e. Books and CD's). To add a new type of goods it would be necessary a major change to the system's structure. The current implementation of the buying and selling agents, the UI and the persistence scheme are extremely tied to the specific description of each item.
  The proposed framework should address these issues, allowing new types of goods to be easily added to the virtual marketplace. For this to be possible, the goods' definitions should be generic enough to support not only commodity type goods such as books and CD's, but also intangible type goods such as knowledge in a specific subject, skills, or services. Also, a standard item description/structure must be developed, so that, agents don't need to be redesigned for every new item added to the marketplace. So far, there are three main approaches being considered for structuring and describing these items: standard Java interfaces, XML [23] or an ontology language such as Ontolingua [26].
  With these simple extensions, many new virtual marketplaces, based on tangible and non-tangible type goods, could be created. An example of a marketplace based on non-tangible goods would be a Language translation market. Agents could be created to represent users selling their services as translators as well as for users looking for translation of their documents. Agents in such a system would be responsible for brokering the appropriate skills and delivering the documents to the appropriate parties as well as mediating the payment negotiation process. There is an infinity of possible scenarios such as: smart job boards where companies and prospective employees would create their respective agents, car classified ads, Q&A systems, dating services, and many others.

- *Multiple negotiation dimensions, and strategies*
  In Kasbah, there is one negotiation dimension (price) and three negotiation strategies for this specific dimension: anxious, greedy/frugal, cool-headed. These negotiation strategies are basically price decay functions over time if you are dealing with a selling agent, or price increase functions if you are dealing with buying agents. In the case of the anxious strategy the price varies following a linear function, for the cool-headed it is a quadratic function, and for the greedy/frugal strategies it is a cubic function.
  Although these negotiation strategies are quite simple and not necessarily will obtain the best deals in a given virtual marketplace, they were chosen that way because after some experiments it was identified that users normally don't fully trust their agents if they don't understand completely what is their rationale. This way, these strategies were made simple for the sake of user understanding and trust.

Once these trust barriers are overcome, it will be important to experiment with different and more complex strategies; some of them could take market statistics and other info, attempting to optimize each deal. For this to be possible, the agent negotiation needs to be specified as a separate entity in the agent component making it easier to "plug" different negotiation strategies, even at runtime. This type of approach would also allow the creation of negotiation processes for systems that negotiate over more than one dimension, such as T@T, as well as to create many different strategies for them.

- *Multiple agent's communication protocols:*
  Kasbah agent communication protocol is fairly simple. Both buying and selling agents can make a buying or selling offer composed basically of a price offer for a specific item and the answer for this offer. The answer can be of two types, positive (in this case the deal is closed), or negative (in this case the agents will keep looking for other agents, and adapting their prices over time). This type of protocol is well suited for simple one-dimension strategies like the one being currently used in Kasbah, but if we start to deal with more then one dimension and more complex strategies, this protocol would probably not work, or at least it would be very inefficient. It is desirable to support a scalable communication protocol, where agents could support more complex protocols where more elaborated proposals and counter-proposals would be possible.
  For this task, there are many different option being considered, such as KQML [28] (Knowledge Query Manipulation Language) or even Java Interfaces coupled by Java's Reflection mechanisms [26], where agents could dynamically query other agents' communications capabilities and use the highest common communication protocol supported by them. Although this dynamical protocol resolution mechanism is not necessary, at its minimum there should be support for easily customizing the system and its agent's protocol capabilities without requiring a major redesign of the system.

References

1. Moukas, R. Guttman, and P. Maes. "Agent-mediated Electronic Commerce: An MIT Media Laboratory Perspective." *Proceedings of the First International Conference on Electronic Commerce (ICEC'98)*, Seoul, Korea, April 1998.
2. R. Guttman, A. Moukas, and P. Maes. "Agent-mediated Electronic Commerce: A Survey." To appear, *Knowledge Engineering Review*, June 1998.
3. J. Terpsidis, A. Moukas, B. Pergioudakis, G. Doukidis, and P. Maes. "The Potential of Electronic Commerce in Re-engineering Consumer-Retailer Relationships." *Proceedings of the European Conference on MM & E-Commerce*, Florence, Italy, November 1997.
4. Sarkar, et al. "Intermediaries and Cybermediaries: A Continuing Role for Mediating Players in the Electronic Marketplace"
5. Object Space's Voyager URL:<http://www.objectspace.com/voyager>
6. IBM's Aglets URL:<http://www.trl.ibm.com>
7. General Magic's Odyssey URL:<http://www.genmagic.com/agents/>
8. Mitsubishi Electric ITA Horizon Systems Laboratory's Concordia URL: < http://www.meitca.com/HSL/Projects/Concordia >
9. Oracle's Mobile Agent White Paper URL:<http://www.oracle.com/products/networking/mobile_agents/html/whitepaper.html>
10. R. Guttman and P. Maes. "Agent-mediated Integrative Negotiation for Retail Electronic Commerce." *Proceedings of the Workshop on Agent Mediated Electronic Trading (AMET'98)*. May 1998.
11. R. Guttman, P. Maes, A. Chavez, and D. Dreilinger. "Results from a Multi-Agent Electronic Marketplace Experiment." *Poster Proceedings of Modeling Autonomous Agents in a Multi-Agent World (MAAMAW'97)*. Ronneby, Sweden, May 1997.
12. Chavez, D. Dreilinger, R. Guttman, and P. Maes. "A Real-Life Experiment in Creating an Agent Marketplace." Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97). London, UK, April 1997.
13. Chavez, A. Moukas, and P. Maes. "Challenger: A Multiagent System for Distributed Resource Allocation: A Closed CPU-Time Market." *Proceedings of the International Conference on Autonomous Agents*, Marina Del Ray, California, ACM Press, February 1997.
14. Chavez and P. Maes. "Kasbah: An Agent Marketplace for Buying and Selling Goods". Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96). London, UK, April 1996.
15. P. Alencar, D. Cowan, C. J. Lucena, and L. C. Nova, The Views Relationship in Object-Oriented Analysis and Design" Technical Report, CS 97-13, University of Waterloo, Ontario, Canada, 1997
16. D. Cowan and C. J. Lucena "Abstract Data Views: An Interface Specifications Concept to Enhance Design for Reuse", IEEE Transactions on Software Engineering, March 1995
17. E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, "Design Patterns, Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995
18. Yariv Aridor, Danny B. Lang "Agent Design Patterns: Elements of Agent Application Design", International Conference on Autonomous Agents 1998.
19. MIT Media Laboratory's Kasbah. https://kasbah.media.mit.edu
20. Alexander Chislenko "Hipereconomy" Dow Jones' MaRKETS magazine April 30, 1998.
21. R. Guttman. "Merchant Differentiation through Integrative Negotiation in Agent-mediated Electronic Commerce. MIT Media Laboratory Thesis Proposal
22. R. Guttman and P. Maes. "Cooperative vs. Competitive Multi-Agent Negotiations in Retail Electronic Commerce." To appear, *Proceedings of the Second International Workshop on Cooperative Information Agents (CIA'98)*. Paris, France, July 1998.
23. "Extensible Markup Language (XML)." World Wide Web Consortium (W3C) Working Draft, 17 November 1997.
24. D. Rogers. Inside COM: Microsoft's Component Object Model. Microsoft Press, 1997.
25. Tete-a-Tete URL: <http://ecommerce.media.mit.edu/Tete-a-Tete/>
26. T. Gruber. "Ontolingua: A Mechanism to Support Portable Ontologies." Stanford University, Knowledge Systems Laboratory, Technical Report KSL-91-66, March 1992.
27. "RMI and Java Distributed Computing." SunSoft white paper, 1997. URL:<http://www.javasoft.com/features/1997/nov/rmi.html>
28. T. Finin, Y. Labrou, and J. Mayfield. "KQML as an Agent Communication Language." J. Bradshaw (ed.), Software Agents, MIT Press, 1997.
29. MIT Media Lab AmEC (Agent mediated Electronic Commerce) Infrastructure Project. URL:< http://ecommerce.media.mit.edu/Infrastructure/infra.html >
30. Firefly Inc. URL:< http://www.firefly.com>
31. Ebay Inc. URL:http://www.ebay.com

SpinOff: a Component-Based Transformation System

Marcelo Sant'Anna, Julio Cesar Sampaio do Prado Leite
{santanna, julio}@inf.puc-rio.br

Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introduction

Transformation systems are very flexible tools that have been successfully helping software engineers on diverse tasks such as software maintenance, compiler construction and the development of software metrics tools. Typically, transformation systems work around rule-based languages that are supported by batch compilation operating over textual files. While such approach for transformational technology has shown success along several years of development, it provides very limited support for interoperability with modern software engineering component-oriented systems. Based upon our previous research work with the Draco-PUC transformation system, we now present the SpinOff environment, a component-based development workbench for the construction of transformational solutions. SpinOff uses an analogy to VLSI electronic circuits as a design approach to software transformers. By providing support for the design and simulation of transformation circuits (TCs), SpinOff is targeted to the construction of reusable transformational parts that can be integrated to other components on a shared software bus. In this paper, we present the concept of component-based transformation systems and then we focus on our particular implementation by showing how we represent and execute SpinOff TCs.

Software transformation techniques have been proved to really add value to important software engineering tasks, such as software maintenance, retargetable compiler construction and the development of software metrics tools. The achieved results are mainly due to the flexible automated support provided by software transformation systems (TSs). These tools capture essential knowledge about transformational processes, easing the task of building common assets for a transformational solution. Software development tools such as Refine [9], Draco-PUC [4][6][7][10], TXL [5] and Popart [12] are some examples of representative TSs.

Despite the success along several years of research, development and usage, right now the typical rule-based batch-compiler software architecture adopted by TSs need to be rethought, so they can keep up with current advancements on software engineering practice. TS implementors are right now very focused on how to evolve their systems architecture so they can provide better, flexible tools for the needs of modern software engineering. Along with the mentioned TS projects, some very important activities in the area can be found at the DMS [2], IP [1], AO [3] and JTS [11] projects. In spite of tackling distinct aspects of TSs, all these projects share the common goal of putting transformational technology a step beyond, so that it can become more useful to software engineers.

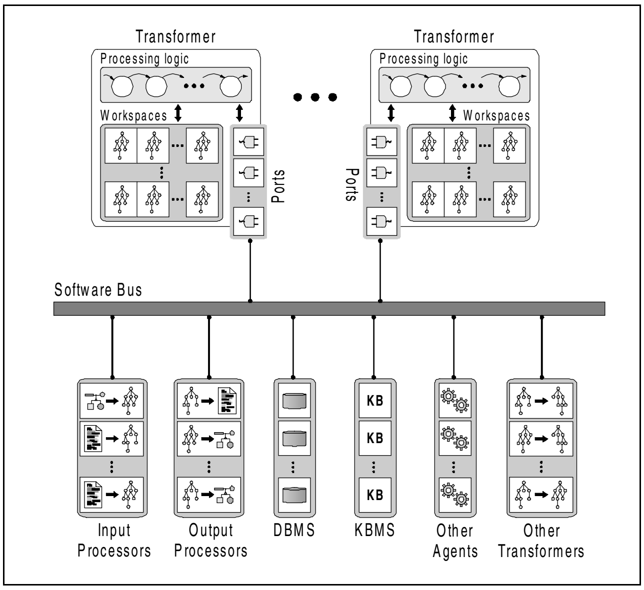
In our case, inside the Draco-PUC project, current efforts have been put into the development of a new version of our transformational workbench with special emphasis on component-based support for transformational solutions. The under development prototype is being called SpinOff and it is firmly based on the achievements gathered so far with our previous Draco-PUC implementations and the experiments concerning the Draco paradigm [8].

2. Component-Based Transformation Systems

As we have stated earlier, typically, transformation systems operate as batch systems both for processing languages and transformers, as well as for manipulating software models. Often, input and output processors, as well as transformation assets, are kept in a file system and software models are just transients to the transformation engine, without the concept of a repository for them. The explanation for such architectural approach lies in the roots of the standard solutions adopted by the compilers community. For such reason, transformation systems are typically monolithic systems working as black boxes with very limited interfaces to external agents. This interface, in general, is provided by textual files stored in a shared file system.

Recent advancements on distributed systems technology have been showing the advantages of interoperability in a component-based approach. Success has been shown even in highly heterogeneous environments such as the Internet. Inspired by such achievements, we have been looking for a way to spread transformation system concepts along a set of communicating components. Such approach is headed towards the development of what we have been calling a *C*omponent-*B*ased *T*ransformation *S*ystem (CBTS). In this kind of system, transformational knowledge is captured in a set of reusable interoperable components, as opposed to being embedded on a large compiler-compiler-like monolithic system.

There are several ways how this very essential idea can be followed. In our case, we have defined a standard architectural model for the class of CBTSs that we are looking for. In this standard model, input and output processors, transformers, and external components are able to interact through a shared software bus (see Figure 1). Transformers communicate over ports that can transfer in and out software models or information concerning these models. Internally, transformers can capture software models inside storage cells, called workspaces, so that specific processing logic can operate over them locally, if needed. We term as an external component any available component in the software bus that is not directly associated to the manipulation of software models. Examples of external components are databases, knowledge bases and web services. Among other things, such components can provide additional information for the decision process when a software model is being transformed. Visualization or user-interface components represent other categories of very useful external components.



SpinOff standard architectural model for a CBTS

Following this model, a software system making use of transformational technology is assembled from reusable available components belonging to the categories we have shown. If, for example, a compiler from Smalltalk to Java bytecode is to be assembled, we can choose among different implementations from the available components that we have over the bus.

Suppose a scenario where there are some available components like a Smalltalk parser as an input processor for Smalltalk programs, a Smalltalk to Java transformer, a Java to Java VM bytecode transformer, a Java VM bytecode output processor, a Java VM bytecode transformer acting as a profiler that generates relational data, some Java VMs and several GUI components. If the transformers hold the property of traceability, we can assemble a performance-tuning software reengineering tool for Smalltalk systems targeting Java VMs. If we are not satisfied on having to first transform Smalltalk to Java and just then being able to generate bytecode, we can build a transformer that act as a kind of Smalltalk compiler directly mapping Smalltalk programs to bytecode. If control-flow analysis components for OO programs are available in the bus, again, our job can be facilitated through component reuse.

Suppose another scenario where someone wants to build a UML CASE tool providing support for model evolution through design patterns. Then, the bus could be populated with an input processor for UML diagrams, design pattern finders and appliers implemented as transformers, a UML to Smalltalk transformer and UML and Smalltalk output processors. A Smalltalk input processor and Smalltalk to UML transformer would be a plus on the construction of a round-trip engineering tool. If the previous Smalltalk to bytecode transformer were available, we would even have a retargetable back-end for the CASE tool.

Is the software engineering community far away from these scenarios? The answer is no, since similar tasks have already been developed using transformational technology [10], but it is still hard to put these achievements together because of the absence of interoperability between developed transformers. The only way to glue them together deals with the use of a shared file system approach, which, definitely, in most cases, is not a reasonable solution. Sure, it is possible to develop some layers on top of the existing transformers, but as one can notice, plug-and-play component reuse would be far away from being a common practice. The CBTS approach is aimed exactly towards this point, by looking for ways to promote higher reusability at the transformer level.

References

[1] W. Aitken et al. *Transformation in Intentional Programming*. Proc. ICSR98.

[2] I. Baxter, and C. Pidgeon. *Software Change Through Design Maintenance*. Proc. ICSM97.

[3] Ted J. Biggerstaff, *Anticipatory Optimization in Domain Specific Translation*, Proc. ICSR98.

[4] A. Brown. *Component-Based Software Engineering: Selected papers from the Software Engineering Institute*. IEEE Press. 1996.

[5] J. Cordy and I. Carmichael. *The TXL programming language syntax and informal semantics*. Technical Report. Queen's University at Kingston, Canada. http://www.qucis.queensu.ca/STLab/TXL

[6] J. Leite, M. Sant'Anna, and F. G. de Freitas. *Draco-PUC: a Technology Assembly for Domain Oriented Software Development*. Proc. ICSR94.

[7] J. Leite, M. Sant'Anna, and A. F. do Prado. *Porting COBOL Programs Using a Transformational Approach*. Journal of Software Maintenance: Research and Practice, January 1997.

[8] James M. Neighbors. *Draco: A Method for Engineering Reusable Software Systems*. In Ted J. Biggerstaff and Alan Perlis (Eds.), Software Reusability, Addison-Wesley/ACM Press, 1989.

[9] Reasoning Systems. *REFINE User's Guide*, Reasoning Systems Inc., Palo Alto, CA. 1992

[10] M. Sant'Anna, J. Leite, and A. Prado. *Draco-PUC: a Workbench for Developing Transformation-Based Software Generators*. Proc. ICSE98 - vol.2.

[11] Y. Smaragdakis and D. Batory. *DiSTiL: a Transformation Library for Data Structures*. USENIX Conference on Domain-Specific Languages, 1997.

[12] D. Wile. *POPART: Producer of parsers and related tools system builders'manual*, Technical Report, USC/Information Sciences Institute, Los Angeles, CA.

An Application Framework for the Parallel Join Operation in Database Systems

Sérgio E.R. de Carvalho, Alberto Lerner, Sérgio Lifschitz
{sergio, lerner, lifschitz}@inf.puc-rio.br

Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225 – Gávea, Rio de Janeiro – RJ
22453-900 Brasil

1. Introduction

Since the introduction of parallel processing architectures, there has been a great deal of effort on the development of Database Management Systems (DBMS) that can benefit from the alleged promises of speed-up and scale-up in processing power [DG92]. As part of this effort, algorithms for the resolution of relational operators have been redesigned to support intra-operation parallelism - one of the available techniques for DBMS parallelization.

In this context, the parallelization of the join operator has gained much attention, due to its high cost and extreme importance in the relational scenario. It is our claim that the literature lacks, so far, a comparison discipline for this class of algorithms - one able to identify which algorithms are best suited for each execution scenario, similar to the profiles of each sequential algorithm variation [ME92]. In our opinion the following issues have greatly influenced this deficiency:

- Proposed algorithms are not always implemented and results come from differently parameterized cost models;
- Load-balanced algorithms often compare their performance with no-load-balanced counterparts, but seldom do it with related algorithms;
- Load balancing strategies differs a great deal on different algorithms;
- Parallel architectures addressed by the algorithms are not consensual.

To address these problems, we have designed a framework for the parallel processing of the join operation, the PJ framework. Basically, it is capable of instantiating many of the algorithms published in the literature, regardless of their parallelization strategy, their target parallel architecture, their load-balancing techniques, and other parameters. The PJ framework provides common support services needed by the algorithms, and promotes the reuse of code, design and domain expertise. It also provides facilities for incorporating *yet-to-be-published* algorithms.

2. The PJ Framework

The parallel processing of the join operation over two relations involves partitioning the relations, and then applying a number of procedures - such as node selection, local join processing, etc - over them. The design of the PJ framework is centered on this partition idea, and is inspired by the observation that join algorithms can be modeled through partitions life-cycles [CLL98]. Moreover, distinct parallel join algorithms employ similar procedures - called characters in the PJ framework context - to process partitions.

Every character in the framework can be played by several distinct actors, each implementing a specific strategy. An algorithm under the framework is free to choose the actor to use on a given occasion. The PJ framework provides a library of reusable actors capable of implementing operations as seen on published parallel join algorithms.

Besides partitions, characters and actors, the PJ framework also supports various issues related to parallelism, namely, distribution on a parallel machine, diversity of parallel architectures, and load balancing needs.

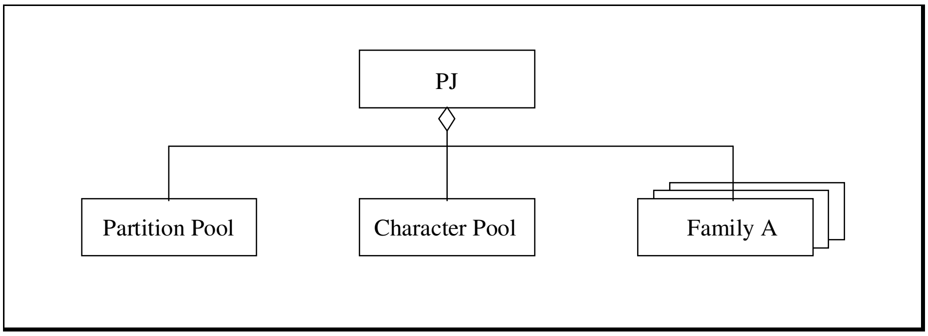


Figure 1 - The High-Level Class Diagram of the PJ Framework

The high-level design of the PJ framework is depicted by the class diagram shown in figure 1. The framework consists of three major groups of classes: the Partition Pool, the Character Pool, and the Algorithms, classified in Families. The description of each of these groups, which we now turn our attention to, is greatly facilitated by the extensive use of Design Patterns [GHJ+94].

The PJ framework is built as a part-of hierarchy, having a mediator object (PJ) to facilitate communications and to decouple its main class groups. Each of the groups' main classes behaves as a Colleague Class and the PJ class is a Mediator pattern. The mediator supports message routing among colleague classes, as well as providing an interface for external contexts the PJ framework may be inserted.

The Partition Pool (PP) is the abstraction treating all partition issues. It is modeled as a Singleton design pattern. The PP consists of a collection of Partition objects that are exclusively accessed via the Pool. It means that anyone wanting to use a specific partition - or a group thereof - must use PP services to do so. Thus, PP provides facilities for addressing its components in diverse ways.

The Character Pool (CP) is the abstraction that captures all the operations that can be performed over partitions. It consists of all possible characters a partition might be submitted to. Each character is represented via a Strategy pattern, and is implemented by a variety of Actors, its concrete subclasses.

The last of the main PJ class groups is the Algorithms group. Algorithms presenting similar behavior are factored out into Families. Algorithm families are implemented via the Template Method pattern. Indeed, it is a common practice in the parallel join literature for a paper to introduce a family of highly similar algorithms (though each directed toward a different execution scenario).

3. Contributions

The contributions of the PJ framework can be summarized as:

- The introduction of the concept of partitions life-cycles and how it is used to capture distinct algorithms behavior - even for load-balanced parallel algorithms;
- The ability to instantiate a great number of algorithms proposed in the literature;
- Extensibily and reusability, achieved via the object-oriented modeling techniques used on the framework. Domain expertise encapsulated in the PJ framework can be reused in new algorithms and the framework itself can be extended;
- Suggestion of a tool for comparing parallel join algorithms and for the profiling of such algorithms;

References

[CLL98] Carvalho, S.E.R., Lerner, A., and Lifschitz, S., "An Application Framework for the Parallel Join Operation in Database Systems", Technical Report PUC-RioInfMCC 40/98, Departamento de Informática, Pontifícia Univerdade Católica do Rio de Janeiro, December, 1998.

[DG92] DeWitt, D. and Gray, J., "Parallel Database Systems: The Future of High Performance Database Systems", Communications of the ACM, 35(6):85-98, 1992.

[GHJ+94] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994.

[ME92] Mishra, P. and Eich, M.H., "Join Processing in Relational Databases", ACM Computing Surveys, 24(1):63-113, 1992.