

# 19º SIMPÓSIO BRASILEIRO DE BANCO DE DADOS

## I SESSÃO DE DEMOS

19 de Outubro de 2004  
Brasília – Distrito Federal – Brasil

### Promoção

SBC – Sociedade Brasileira de Computação  
Comissão Especial de Banco de Dados



ACM – Association for Computing Machinery  
SIGMOD – Special Interest Group on Management of Data



### Apoio

VLDB Endowment



### Edição

José Palazzo de Oliveira (Universidade Federal do Rio Grande do Sul–UFRGS)  
Carina Friedrich Dorneles (Universidade Federal do Rio Grande do Sul–UFRGS)  
Eduardo Kroth (Universidade de Santa Cruz–UNISC)

### Organização

Murilo S. de Camargo (Universidade de Brasília – UnB)



### Realização

Departamento de Ciência da Computação da Universidade de Brasília - UnB  
Departamento de Engenharia Elétrica da Universidade de Brasília - UnB

## CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Sessão de Demos do Simpósio Brasileiro de Banco Dados(01.:2004 out. 19: Brasília).

Anais/Edição José Palazzo de Oliveira, Carina Friedrich Dorneles, Eduardo Kroth – Porto Alegre: Universidade Federal do Rio Grande do Sul, 2004.  
61 p.: il.

ISBN 85-7669-006-3

Conhecido também como Sessão de Demos SBBD 2004.

1. Banco de Dados. I. Palazzo, José; Fridrich, Carina; Kroth Eduardo. II. SBBD (19.: 2004: Brasília).

“Esta obra foi impressa a partir de originais entregues, já compostos pelos autores”

Capa: Fernando Ribeiro

Editoração: Diogo Rispoli, Hugo Lousa, Wantuil Firmiano Jr.

# 19th BRAZILIAN SYMPOSIUM ON DATABASES

## 1st Tools Session

October 19th, 2004  
Brasília – Distrito Federal – Brazil

### Promotion

SBC – Brazilian Computing Society  
Special Committee on Databases



ACM – Association for Computing Machinery  
SIGMOD – Special Interest Group on Management of Data



### Support

VLDB Endowment



### Editor

José Palazzo de Oliveira (Universidade Federal do Rio Grande do Sul–UFRGS)  
Carina Friedrich Dorneles (Universidade Federal do Rio Grande do Sul–UFRGS)  
Eduardo Kroth (Universidade de Santa Cruz–UNISC)

### Organization

Murilo S. de Camargo (Universidade de Brasília – UnB)



### Realization

Departamento de Ciência da Computação da Universidade de Brasília - UnB  
Departamento de Engenharia Elétrica da Universidade de Brasília - UnB

## **I Sessão de Demos/SBBD 2004**

### **Apresentação**

A Sessão de Demonstrações de Ferramentas para Banco de Dados, em sua primeira edição, foi organizada com o intuito de proporcionar um fórum para intercâmbio de experiências e de soluções automatizadas entre pesquisadores da área de Banco de Dados no Brasil. O público-alvo inclui membros da comunidade acadêmica, que podem demonstrar os resultados de seus projetos de pesquisa aplicados à área de Banco de Dados.

O processo de seleção e avaliação das ferramentas aconteceu em duas etapas. Na primeira etapa, os autores submeteram artigos descrevendo as principais características de suas ferramentas, para análise pelos membros do Comitê de Avaliação. Na segunda etapa, as ferramentas selecionadas foram apresentadas durante o evento, quando então foram novamente avaliadas in loco, por membros do Comitê. Esta segunda avaliação visou garantir que as ferramentas selecionadas fossem operacionais e funcionalmente compatíveis com a descrição contida nos documentos avaliados na primeira etapa. Os resultados das duas avaliações forneceram subsídios para que o Comitê de Avaliação pudesse escolher e premiar os autores da Melhor Demonstração de Ferramenta de Banco de Dados. O Comitê de Avaliação foi montado com pesquisadores de Universidades com grupos de reconhecida competência do Brasil que se prontificaram em colaborar com a primeira edição da Sessão de Demos. Além deles, tivemos a participação de um professor estrangeiro no Comitê de Avaliação.

Em sua primeira edição, a Sessão de Demos 2004 incluiu a apresentação e a demonstração de 10 ferramentas selecionadas pelo Comitê de Avaliação, a partir de um total de 26 submissões (taxa de aceitação: 38%). Com base nos dados enviados pelos autores dos artigos submetidos, percebe-se que 50% das ferramentas foram definidas como ferramentas de uso acadêmico e comercial. Além disso, foi constatada uma forte participação de ferramentas desenvolvidas com o objetivo final de validação de projetos de pesquisa, cerca de 25% delas. Do total de submissões, 74% está inserida diretamente em trabalhos de mestrado e doutorado. Mais de 50% das ferramentas submetidas encontram-se completamente concluídas, demonstrando ativo trabalho dos grupos na preparação de protótipos para validação de idéias. Sistemas multiplataformas também tiveram uma participação boa, apesar de ainda se sobressaírem ferramentas para plataforma Windows, cerca de 40%. Pôde-se observar certa preocupação por parte dos pesquisadores em desenvolver ferramentas com suporte a Web, as quais somaram 28% das submissões. Por fim, vale ressaltar que as ferramentas submetidas estavam, em sua totalidade, dentro das grandes áreas de Banco de Dados, mostrando que existe espaço no Brasil para a realização de um evento como a Sessão de Demos. De forma geral, ferramentas para consultas, mineração de dados e integração BD/XML somaram um total de 38% as submissões. O restante das submissões se enquadra em assuntos como geração de índices, BD móveis, BD temporais, alocação de dados, entre outros.

Finalmente, desejamos que o evento atenda às expectativas de todos os participantes, tanto pela troca de experiências com colegas quanto pelas possibilidades de crescimento e aperfeiçoamento das ferramentas apresentadas.

José Palazzo de Oliveira  
Coordenador

Carina Friedrich Dorneles  
Eduardo Kroth  
Organizadores

## **I Sessão de Demos / SBBD 2004 – Comitê**

### **1st Tools Session / SBBD 2004 - Committee**

#### **Coordenador do Comitê de Programa**

José Palazzo de Oliveira - UFRGS

#### **Comitê de Programa – 1ª Etapa**

Altigran da Silva - UFAM

Caetano Traina - USP

Décio Fonseca - UFPE

Luis Fernando Bessa Seibel - PUC-RJ

Mariano Consens - University of Toronto

Marta Mattoso- COPPE/UFRJ

Wagner Meira Jr. - UFMG

#### **Avaliadores Externos**

Flávia Delicato

Geraldo Bonorino Xexéo

Gabriela Ruberg

Paulo F. Pires

#### **Comitê de Organização**

Carina Friedrich Dorneles - UFRGS

Eduardo Kroth - UNISC

#### **Comitê de Organização do SBBD / SBES 2004**

Murilo S. de Camargo, UnB (Coordenador Geral)

Alba Cristina M. A. de Mello - UnB

Célia Ghedini Ralha - UnB

Daniel Arruda Santos Anjos - UnB

Diogo de Carvalho Rispoli - UnB

Hugo Antônio de Azevedo Lousa - UnB

João José Costa Gondim - UnB

José Carlos Ralha - UnB

Maria Emília Walter - UnB

Rafael de Timóteo de Souza - UnB

Ricardo P. Jacobi - UnB

Ricardo Puttini - UnB

Robson de Oliveira Albuquerque - UnB

Soemes Castilho Dias - UnB

Wantuil Firmiano Júnior - UnB

## Sumário

### Contents

1 – “SAXES: A Software for Exchanging Data Between XML and Databases” _____	1
Sérgio Mergen e Eduardo Kroth - UFRGS e UNISC	
2 – “XML Publisher: Um Framework para Publicação de Dados Armazenados em Banco de Dados Relacional ou Objeto Relacional como XML” _____	7
Vânia Vidal, Wamberg Oliveira, Lineu Santos, Fernando Cordeiro e Denis Lima - UFC	
3 – “Eris: Um protótipo para consulta por similaridade a bases de dados XML” _____	13
Andrei Eneas Nunes Lima, Carina F. Dorneles e Carlos A. Heuser - UFRGS	
4 – “iDFQL - interactive Data Flow Query Language” _____	19
Ana Paula Appel, Caetano Traina Júnior - USP São Carlos	
5 – “ROSA: Um Ambiente de Desenvolvimento e Armazenamento de Conteúdos Didáticos com Acesso Semântico” _____	25
Ana Maria de C. Moura, Fábio Porto, Abílio Fernandez, Adriana Fernandez e Fábio Coutinho - IME Rio	
6 – “XAlloc: Uma Ferramenta para Avaliar Algoritmos de Alocação de Dados” _____	31
Matheus Wildemberg, Melise M. V. Paula, Fernanda Baião, Marta Mattoso - COPPE-UFRJ	
7 – “Um agente de Software para Criação de Índices no PostgreSQL” _____	37
Marcos Antonio Vaz Salles e Sérgio Lifschitz - PUC-Rio	
8 – “PDM - Palm Database Manager” _____	43
Dorotéia Pitombeira, Angelo Brayner e Nabor das Chagas Mendonça - UNIFOR	
9 – “ClockTool: Uma Ferramenta Web para Desenvolvimento e Manutenção de Banco de Dados Temporais” _____	49
Eugênio O. Simonetto, Duncan D.A.Ruiz e Márcio T. Nazari - UNIFRA e PUC-RS	
10 – “TVM Web - Uma Interface Visual para o Modelo Temporal de Versões” _____	55
Vincent Nelson Kellers da Silveira, Nina Edelweiss e Renata de Matos Galante - UFRGS	

## **SAXES: A Software for exchanging data between XML and databases**

**Sergio Luis Sardi Mergen<sup>1</sup> , Eduardo Kroth<sup>2</sup>**

<sup>1</sup>Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)  
Caixa Postal 15064 – 90501-970 Porto Alegre, RS

<sup>2</sup>Departamento de Informática – Universidade de Santa Cruz do Sul (UNISC)  
Caixa Postal 188 – 96815-900 Santa Cruz do Sul, RS

mergen@inf.ufrgs.br, kroth@inf.ufrgs.br

***Abstract.** A great deal of companies is bringing XML into their day to day activities. A common scenario involves representing business data as XML for the publishing of data on the web and for data exchange purposes. Given most of the data is currently stored in relational databases, it is necessary to convert data between a relational representation and a XML representation. In this paper we present SAXES, a system that allows the conversion of data between these two data formats by means of mappings between the schemas.*

### **1. Introduction**

A great deal of companies is bringing XML into their day to day activities. The benefits of using XML technology have manifold reasons. At first, XML is a standard for describing data in a structured fashion. Besides, some services have being created with the purpose of leverage XML power even further. Some of the services surrounding XML allows transforming(XSLT), querying(XQuery) and handling XML documents in a standardized way(DOM and SAX APIs).

Those features makes of XML a good solution for applications such as the publishing of data on the web and the exchange of data among companies. Usually the data is store in a format other than XML, such as relational databases. So, in order to take advantage of XML features, it is necessary to convert data between the XML format and the original format of the data. As a matter of fact, the companies using relational databases represent the biggest slice of the market, so we restrict the scope of our research to the conversion between XML data and relational data. In this paper we shall use the term **XML pulling** meaning the conversion of data from relational databases to XML. Likewise, **XML pushing** means the other way around.

The XML pushing and pulling may be aided by the usage of conversion tools. Some of such tools perform the XML pulling by using fixed pre-defined conversion rules. In a few words, the nesting of elements in the generated XML document resembles the convey structure of the relational schema. Additionally, the elements names are extracted from the names of the tables and columns of the relational schema. The XML pushing follows the same principle. It means that the XML document can only be pushed inside

the database if the structure and the name of the elements in the XML document copes with the relational schema.

However, the ability to control the way the output is generated is necessary, so the whole potential of XML can be explored. For instance, it is highly probable the company uses a standard XML schema to publish their data or to communicate their data with other applications. In order to convert relational data to that specific XML format, the tools should express the conversion rules in a flexible matter, given the user the freedom to choose how the data should be converted.

The most common and practical way to make the conversion customizable is to allow the mapping between the XML schemas and the relational schemas. Many software vendors follows the mapping approach to handle data conversion [1, 2, 3]. Such systems differ in the way the user defines the mappings. The schema mappings can go from a complete abstract process, in which only the matchings<sup>1</sup> between the schema elements are required, until the specification of complex conversion rules regarding the translation of elements from one schema into elements of another schema.

The former kind of mapping eases the user work, but it's usage is limited to simple cases. For instance, it may not be possible to translate relational data into a XML document with a complex nesting of elements, or when several relational tables must be joined. In the other hand, by defining conversion rules such cases can be solved, with the cost of needing to deal with the low level requirements of the conversion. There are also alternatives that mix features of both kinds of mappings in a hybrid approach.

In this paper we present a system capable of performing the conversion between XML and relational schemas. The novel aspect of the system is that it handles complex data conversions while the user is provided with a high level of abstraction. The system uses reasoning on the schemas to automatically derive the necessary conversion rules. If, however, the system assumptions on the conversion rules are incorrect, the user can still adapt them to reflect the true conversion desires.

This paper is organized as follows. Section 2 describes some overall aspects of our system. Section 3 and 4 respectively show some details on the XML pulling and XML pushing. In section 5 we present details concerning the reasoning processes. Finally we summarize our work in section 6.

## **2. SAXES**

SAXES<sup>2</sup> is a middleware developed in Java capable of communicating relational data with XML. It carries out the data exchange by the definition of matchings between the columns of the relational schema and the elements and attributes of the XML schema.

The tool allows the opening of a database schema and an XML schema, and let the user supplies matchings between both schemas. Because SAXES uses a standard JDBC connection, it handles all major relational database vendors, as well as popular open-source database systems. The system provides full support for DTDs. There is also a limited support for XML Schemas.

---

<sup>1</sup>Schema matching is the correspondence of elements from two schemas

<sup>2</sup>SAXES stands for Semi-Automatic XML Exchange System



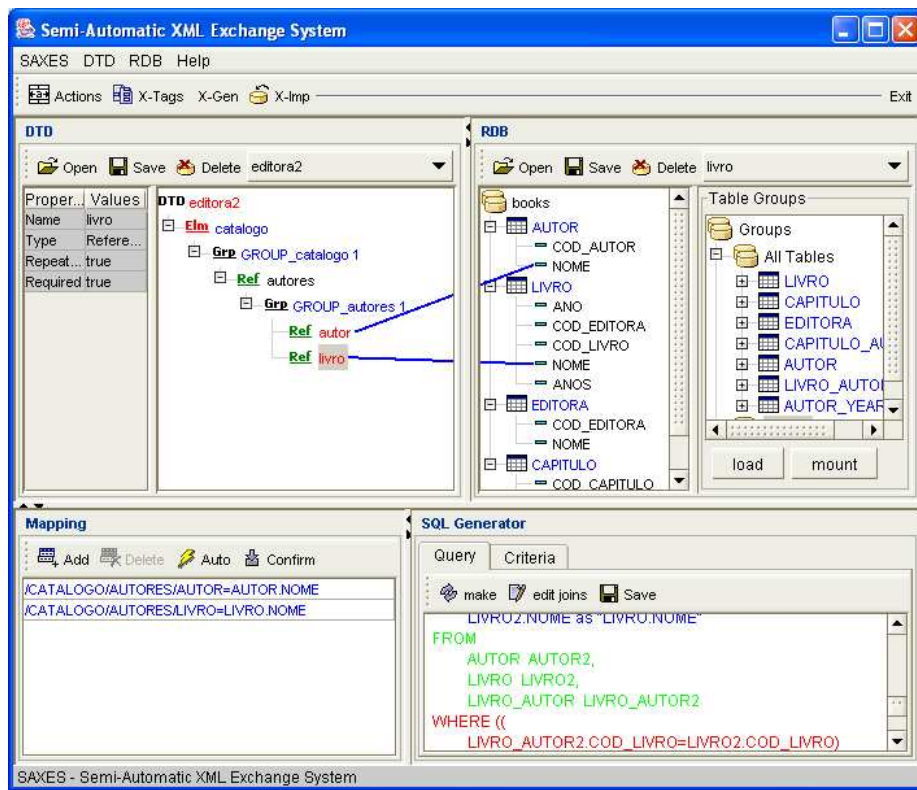


Figure 1: SAXES main window

The schemas are represented as trees, where the XML schema tree has several levels of nesting, while the database schema has only two: The outer level for the tables and the inner level for the columns. The leaf nodes of both trees represent the nodes that can be matched. The matchings can be visualize either in a text format or as lines connecting the matched nodes.

The XML schemas, the relational schemas and the matchings among the schemas can be store in a metamodel for later usage. This prevents the user to reconstruct the matchings each time he desires to convert data from one format to another. The software also presents a DTD editor, that allows the user the create DTDs without leaving the tool.

### 3. XML Pulling

The process of pulling XML from a database can be briefly described in three steps:

1. Creation of an SQL query.
2. Execution of the SQL query: the generated query is executed and its result set is pipelined to the XML generation algorithm (*tager*).
3. XML generation: the *tager* transforms the result set into XML.

In order to convert relational data into XML properly, the *tager* requires the result set to be disposed in a specific way. Otherwise the XML generation can be compromised. Based on the user schema matchings and the schemas themselves, SAXES uses query discovery techniques [4] to automatically derive an SQL query that covers, at the extent possible, the user desires.

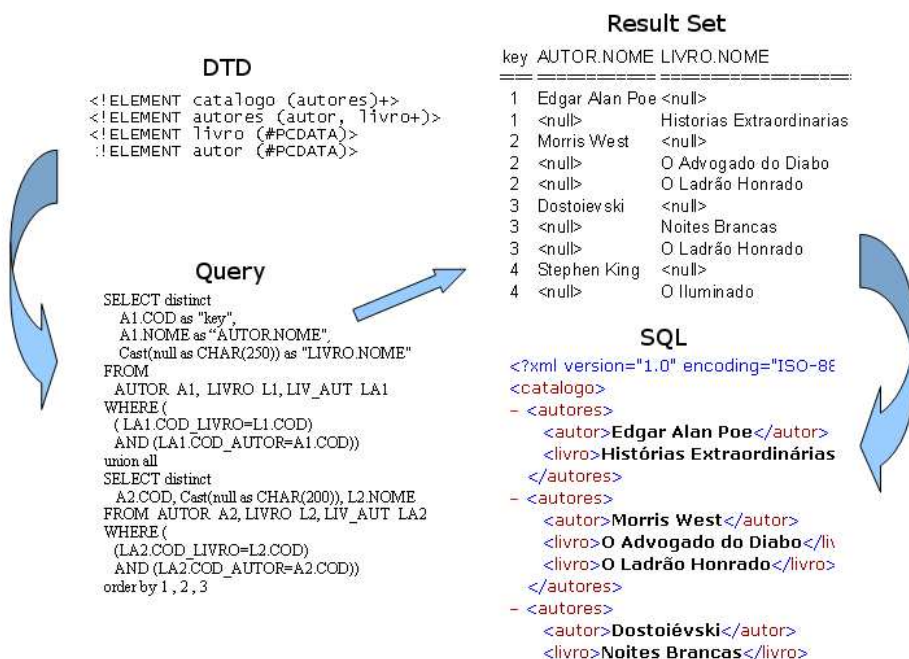


Figure 2: Overall Pulling Process

The generated query is a collection of subqueries, combined using the UNION ALL operator. The union of several queries for XML generation purposes is often called in the literature as *SORTED OUTER UNION* approach [5]. The effect of using this type of query is on the result set. Results sets are traditionally flat, since most relational database lack the concept of nested tuples. Sorted Outer Union queries can make the result appears "disguised" as nested data. This approach is particularly useful when the XML schema represents data with a complex nesting. The tagger can naturally create nested content by traversing a result set whose structure resembles the XML hierarchy desired. We depict the overall pulling process in figure 2. For this example we assume the DTD is mapped to a book database. Observe in the query that the joins between tables were already resolved by the software.

**Query Edition:** Usually it is necessary to edit the generated query to add personal criteria on the selection or to change the way the tables are joined. To prevent query misconstructions (queries not compatible with the tagger) the user can edit the query only through SAXES facilities. SAXES provides a high level of abstraction when editing the query. For instance, the user can readily add criteria without concerning how the criteria fits the final query.

**Pulling Customization:** The conversion of relational data to XML was designed in a straightforward way, by the definition of an extensible framework based on a dispatcher that notifies a handler class whenever some particular generation action should be executed on the relational data. SAXES comes with a default handler that generates XML documents. However, the user can implement a custom handler to treat the incoming relational data in a particular way. For instance, a user could implement a handler that turns

the relational data into a CSV format.

#### 4. XML Pushing

The pushing process can be basically described as the conversion of the XML document into a set of DML statements that are executed against the database engine. Next we describe the features supported by our system:

**Definition of DML actions:** The default behavior of the software is to generate insert statements as the XML document is being traversed. Nevertheless, it is possible to perform other kinds of statements rather than inserts. This is done by setting the behavior of each table. The behaviors comprise: (i)*INSERT(the default)*, (ii)*INSERT\_STRICT*, (iii)*INSERT\_OR\_UPDATE*, *UPDATE*(iv), (v)*UPDATE\_STRICT* and (vi)*DELETE*.

**Automatic Key generation:** If the primary keys are not mapped to any element or attribute of the XML schema, the user can configure the tool to automatically generate the keys as the data is pushed in the database. The key generation strategy we adopted is based on the maximum existing value for the primary key of the table where the data is being pushed. This prevents the generated key to conflict with any existing record.

**Definition of tables joins:** Another question that arises when it comes to converting data to a database is to resolve the tables joins. SAXES automatically resolves foreign keys relationships between mapped tables<sup>3</sup>, and updates such foreign keys accordingly. Additionally, SAXES identifies all associative tables that should be updated.

**Pushing Customization:** The pushing process were design in the same way of the pulling process, which allows the user to customize the handling of the XML data. For instance, besides inserting the incoming data into a relational table, the user can write code to pass the data to specialized business rules for further processing.

#### 5. SAXES Automation

The process of pulling and pushing XML from a database demands more information than just the matchings between the schemas. Based on the schemas and the matchings provided by the user, SAXES uses reasoning on the schemas to infer the missing information that is not directly argued by the matchings (an example of missing information is the join between tables). The whole inference process is performed behind the scene, and in many cases the user doesn't even know there is more in a conversion operation than the matchings he/she provides.

Complete automation is far from becoming a reality though. Generally speaking, user interference is still required for deciding what to do in some particular cases, specially when ambiguity gets in the way. When it comes to the mapping of XML schemas and relational schemas, one of the most challenging issues is the problem of resolving table joins.

The pulling and pushing process requires the mapped tables to be joined. SAXES identifies all possible ways to join the mapped tables. In large database schemas, there

---

<sup>3</sup>tables in which at least one column is mapped

can be many paths between tables. For instance, to join tables **customer** and **movie**, one possible path is through **rented**, which would represent the movies currently rented by each customer, while another path could go through **history**, which would return the movies already rented by each customer. In such cases, the user must select if whether the first path or the second best fits the output desired.

Another area of automation exploited by SAXES gives respect to the matchings definitions. The manual matching of the schemas can become a time consuming and error-prone task, specially when the schemas are large. To leverage the user performance SAXES can suggest the matchings automatically. The SAXES suggested matchings appear in a different color than the user confirmed matchings. If the matchings are correct, the user turns them into confirmed matchings.

## 6. Conclusion

The conversion between XML data and relational data is a challenging task, since it deals with two different structural paradigms. Conversion tools use mapping between the schemas so the conversion can take place. In most tools, to resolve the heterogeneities and the complex structures that can be found in the schema involved, the user is required to inform a great deal of information in the mappings.

With this drawback in mind, we developed SAXES not only to communicate XML with relational databases, but also to hide from the user the complexity inherently involved in this problem. Our main effort was to automatically derive the mappings between the schemas based simply on the matchings specified by the user. If needed though, the user has the power to change the mappings as he / she sees fit.

We are currently working on a scheduler responsible for executing the XML pulling / pushing from fixed time periods, stipulated by the user. The scheduler can be used for fixed tasks, such as the dumping of relational data into a backup repository. For the future we intend to give support for data exchange between databases and EDI.

## References

- [1] [http : //www.hitsw.com/products\\_services/xmlplatform.html](http://www.hitsw.com/products_services/xmlplatform.html).
- [2] [http : //www.altova.com/products\\_mapforce.html](http://www.altova.com/products_mapforce.html).
- [3] Lucian Popa et al. Mapping xml and relational schemas with clio, demo. In *In ICDE*, 2002.
- [4] Renée J. Miller, Laura M. Haas, and Mauricio A. Hernandez. Schema mapping as query discovery. In *Proceedings of the 26th VLDB Conference*, pages 77–88, Cairo, Egypt, 2000.
- [5] Jayavel Shanmugasundaram and Eugene Shekita. Efficiently publishing relational data as xml documents. In *The VLDB Journal*, 2001.

## **XML Publisher: Um Framework para Publicação de Dados Armazenados em Banco de Dados Relacional ou Objeto-Relacional como XML**

**Vânia Vidal, Fernando Lemos, Lineu Santos, Wamberg Oliveira, Denis Lima, Valdiana Araújo**

**Departamento de Computação – Universidade Federal do Ceará (UFC)**

{vvidal, fernandocl, lineu, wamberg, denis, valdiana}@lia.ufc.br

**Resumo.** Nos últimos anos, a Web se tornou o maior ambiente capaz de fornecer acesso a fontes de dados heterogêneas. Nesse contexto, a linguagem XML se firmou como um formato universal para publicação e troca de dados na Web. Como a maioria dos dados está armazenada em bancos de dados relacionais ou objetos-relacionais, surgiu a necessidade de definir mecanismos para publicá-los no formato XML. Neste trabalho, apresentamos o XML Publisher, um framework para publicação de dados armazenados em banco de dados relacionais ou objetos-relacionais no formato XML. O framework proposto permite a publicação de visões XML, de forma que a base de dados pode ser consultada e atualizada através dessas visões, usando a linguagem XQuery.

### **1. O Framework Proposto**

#### **1.1 Arquitetura**

No XML Publisher os dados são descritos em três níveis de Esquemas como ilustrado na Figura 1. Para publicar uma visão XML, o projetista deve definir uma Visão de Objeto [9], denominada Visão de Objeto Default (VOD), de modo que os objetos da VOD possuem a mesma estrutura dos elementos da visão XML. Visões de objeto provêem uma técnica poderosa para determinar visões lógicas, ricamente estruturadas, sobre banco de dados já existente. No Oracle 9i, uma visão de objeto é definida através de uma consulta SQL:1999, a qual especifica como os objetos da visão são sintetizados a partir dos dados do banco de dados; e, caso atualizações sejam permitidas, devem ser definidos tradutores (*triggers INSTEAD OF*) os quais especificam como atualizações definidas sobre a visão de objeto são traduzidas em atualizações especificadas sobre o banco de dados.

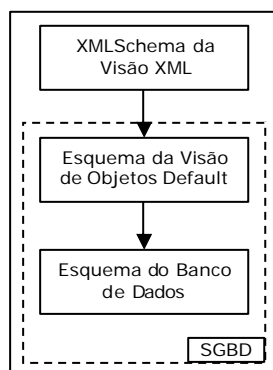


Fig. 1 – 3 camadas de esquema

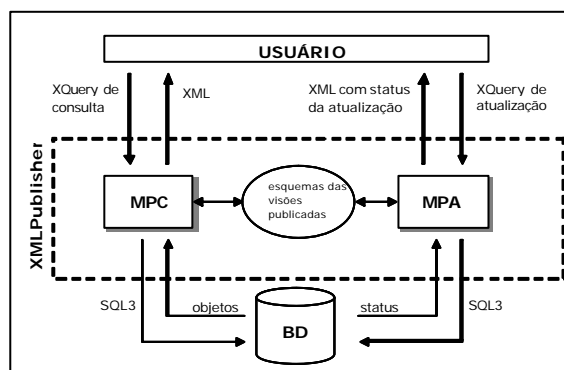


Fig. 2 – Arquitetura do XML Publisher

No framework proposto, consultas e atualizações definidas sobre o esquema da visão XML são traduzidas em consultas e atualizações sobre o esquema da visão de objeto, e o que por sua vez são traduzidas, pelo mecanismo de visão do SGBD, em consultas e atualizações definidas sobre o esquema do banco. Figura 2 mostra os principais componentes do *XML Publisher*. O Módulo de Processamento de Consultas (MPC) é responsável pelo processamento de consultas *XQuery* definidas sobre a Visão XML e o Módulo de Processamento de Atualização (MPA) pelo processamento de atualizações definidas sobre a Visão XML.

## 1.2 Processamento de consultas no *XML Publisher*

Consultas sobre as visões XML publicadas no *XML Publisher* são definidas através da linguagem *XQuery* [14]. Como apresentado na Figura 2, cada consulta *XQuery* submetida ao *XML Publisher* é processada pelo Módulo de Processamento de consulta (MPC) seguindo os seguintes passos. Inicialmente, (i) a consulta *XQuery* é traduzida em uma consulta SQL:1999 sobre a VOD. Dado que a estrutura dos objetos da VOD é compatível com os elementos da visão XML, a tradução é simples e pode ser feita com base apenas no esquema da visão XML. Em seguida, (ii) a consulta SQL:1999 gerada é processada pelo SGBD. Dessa forma, toda a complexidade do processamento dos dados é delegada ao SGBD. Finalmente, (iii) o resultado da consulta SQL:1999 é transformado, baseado na consulta *XQuery* e no esquema da visão XML, no fragmento XML requisitado pela consulta.

## 1.3 Processamento de Atualizações no *XML Publisher*

Atualizações sobre as visões XML publicadas no *XML Publisher* são definidas através de uma extensão da linguagem *XQuery*[12]. Como apresentado na Figura 2, uma atualização submetida ao *XML Publisher* é processada pelo Módulo de Processamento de Atualização (MPA) seguindo os seguintes passos. (i) A atualização *XQuery* é traduzida em uma atualização SQL:1999 definida sobre a VOD correspondente. (2) as atualizações na visão de objeto default são por sua vez traduzidas pelos *triggers INSTEAD OF*, em atualizações definidas sobre o esquema do banco de dados. Em [8] é proposto uma ferramenta para automatizar o processo de geração dos tradutores de atualizações de visões de objeto.

## 1.4 Acessando o *XML Publisher* através da Web:

O *XML Publisher* é disponibilizado como uma aplicação *web*, de forma que seus serviços podem ser acessados por um cliente através de requisições HTTP GET. Essas requisições têm o seguinte formato: `http://host/path?REQUEST=operação&param=value`, onde *host* denota o endereço da aplicação *web* que hospeda o *XML Publisher*, *path* é o caminho da aplicação *XML Publisher* na aplicação *web* e *REQUEST* é o parâmetro da requisição que indica a operação submetida ao *XML Publisher*. O nome (*param*) e o valor (*value*) para o segundo parâmetro da requisição, variam de acordo com a operação. As operações permitidas no *XML Publisher* são:

1. **GetCapabilities:** requisita os nomes das visões XML publicadas, e ações suportadas (consultas e/ou atualizações) por estas.
2. **GetSchema:** requisita o *XML Schema* de uma visão XML publicada.
3. **ExecuteQuery:** requisita consultas *XQuery* definidas sobre o esquema da visão XML publicada.
4. **ExecuteUpdate:** requisita atualizações *XQuery* definidas sobre o esquema da visão XML publicada.

## 2. Ambiente para publicação de visões XML no *XML Publisher*

Para o *XML Publisher* desenvolveu-se um ambiente que dá suporte ao processo de publicação de visões XML. O processo de publicação compreende os seguintes passos:

- (i) O usuário define, através de uma interface gráfica, o tipo dos elementos da visão XML;

- (ii) A ferramenta CriaEsquemaVOD gera automaticamente o esquema da Visão de Objeto Default;
- (iii) O usuário deve então carregar o esquema do banco e, através de uma GUI, definir as assertivas de correspondência do esquema VOD com o esquema do banco de dados [7, 10]. Com base no conjunto de assertivas de correspondência da visão, a ferramenta VBA (ViewByAssertion) gera automaticamente a consulta SQL:1999 e os tradutores de atualização da visão de acordo o mapeamento [2] definido pelas assertivas. Como mostramos em [11] o VBA lida com o problema da heterogeneidade semântica [13], e permite que mapeamentos complexos possam ser especificados de forma simples;
- (iv) os arquivos de configuração do *XML Publisher* são atualizados, de forma que a visão XML é finalmente publicada.

### 3. Estudo de Caso

Considere a base de dados Pedidos cujo esquema relacional é mostrado na figura 3. Suponha que seja publicada a visão XML Pedidos cujo esquema é mostrado na Figura 4. O esquema da visão de objeto default Pedidos\_v gerada para a visão Pedidos\_v é mostrado na figura 5. A definição SQL da VOD Pedidos\_v é mostrada na figura 6. O *trigger* mostrado na Figura 7 define a tradução para inserções na *NESTED TABLE* listatens da visão Pedidos\_v. A seguir apresentamos exemplo de processamento de uma consulta e uma atualização na visão Pedidos\_v usando *XML Publisher*.

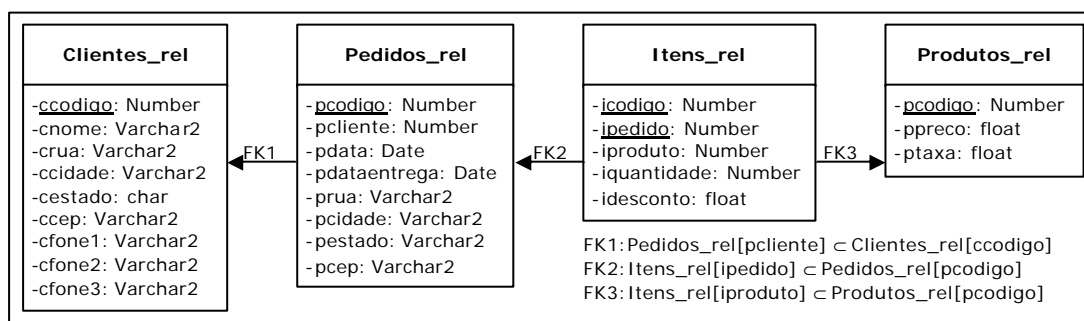


Fig. 3 – Esquema do Banco de Dados Pedidos

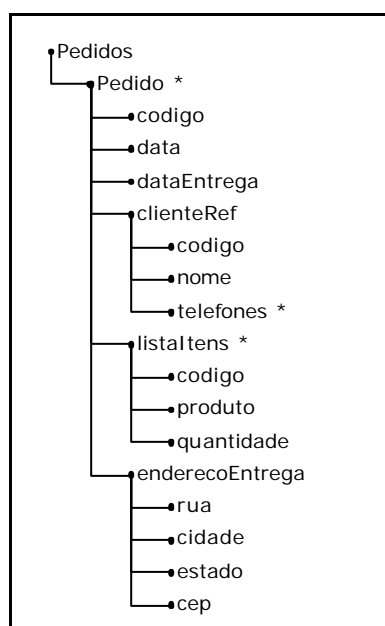


Fig. 4 – Visão XML Pedidos

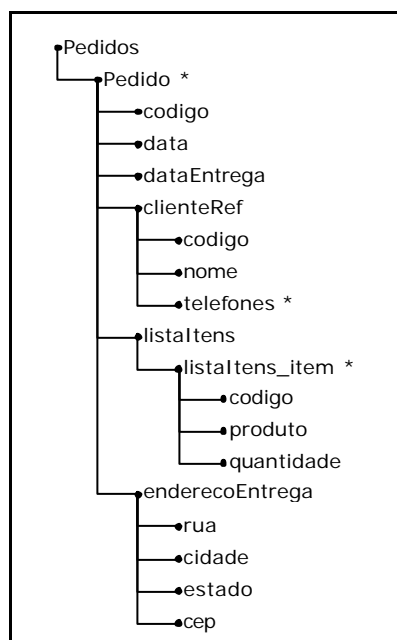


Fig. 5 – VOD Pedidos\_v

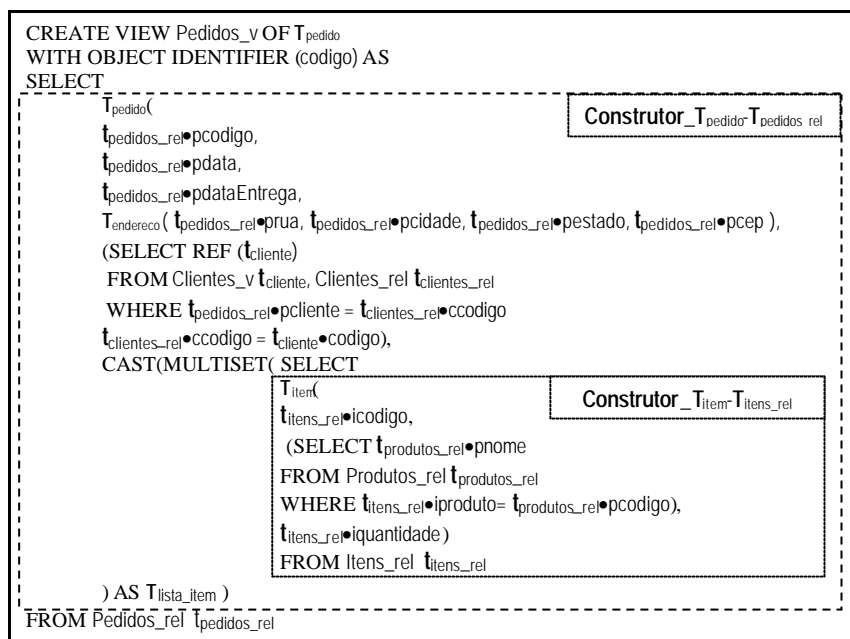


Fig. 6 – Definição da visão de objetos Pedidos\_v

```

CREATE TRIGGER AdicaoEmListaItens
INSTEAD OF INSERT ON NESTED TABLE listaItens OF Pedidos_v
BEGIN

INSERT INTO Itens_rel(icodigo, iquantidade, iproduto, ipedido)
VALUES ( :new.codigo,
        :new.quantidade,
        :new.produto,
        :parent.codigo);

END ;

```

Fig. 7 – Trigger de tradução para inserções na NESTED TABLE listaItens da visão Pedidos\_v

### 3.1 Exemplo de processamento de Consulta no XML Publisher

Considere a consulta *XQuery* *Q* na Figura 8, a qual obtém a data de entrega e os itens com quantidade maior ou igual a dois do pedido do cliente “Pedro”. A seguir discutimos cada passo do processamento dessa consulta no *XML Publisher*.

1. A consulta *Q* é traduzida pelo MPC na consulta *SQL:1999 Q'* sobre a VOD *Pedidos\_v* mostrada na Figura 9. Dado que os tipos dos objetos da VOD *Pedidos\_v* têm a mesma estrutura do tipo da visão XML, essa tradução é simples e pode ser realizada com base apenas no XML esquema da visão *Pedidos\_v*.

```

<PedidoPedro>{
  for
    $i in view("Pedidos_v")/Pedidos/Pedido
    where $i/cliente = "Pedro"
  return
    $i/dataEntrega
    <itens>{
      for $j in $i/listaItens
      where $j/quantidade >= 2
      return
        <item>{
          <descricao>{
            $j/produto/text()
          }</descricao>
          $j/quantidade
        }</item>
    }</itens>
}</PedidoPedro>

```

Fig. 8 – Consulta XQuery *Q*

```

SELECT I.DATAENTREGA,
       CURSOR(SELECT J.PRODUTO,
                    J.QUANTIDADE
              FROM TABLE(I.LISTAITENS) J
              WHERE J.QUANTIDADE >= 2)
FROM PEDIDOS_V I
WHERE I.CLIENTE = 'Pedro'

```

Fig. 9 – Consulta *SQL3 Q'*



DATAENTREGA	CURSOR	
12/03/2004	PRODUTO	QUANTIDADE
	Teclado	3
	Mouse	5

Fig. 10 – Resultado da consulta Q'

```

<PedidoPedro>
  <dataEntrega>12/03/2004</dataEntrega>
  <itens>
    <item>
      <produto>Teclado</produto>
      <quantidade>3</quantidade>
    </item>
    <item>
      <produto>Mouse</produto>
      <quantidade>5</quantidade>
    </item>
  </itens>
</PedidoPedro>

```

Fig. 11 – Documento XML resultante

2. A consulta Q' é executada pelo SGBD com base na definição da VOD Pedidos\_v. Suponha que o resultado da consulta contém os objetos mostrados na Figura 10.
3. O resultado da consulta é convertido no documento XML da Figura 11. A conversão é realizada de forma simples com base na cláusula RETURN da consulta Q e no XML esquema da visão Pedidos.

### 3.2 Exemplo de processamento de Atualização no XML Publisher

Considere o pedido de atualização u na Figura 12, a qual requisita a inserção do elemento item como filho dos nós \$j satisfazendo as seguintes condições: \$j ∈ \$i/listaItens onde \$i ∈ view("Pedidos\_v")/Pedidos/Pedido e \$i/cliente = "Pedro". A seguir discutimos cada passo do processamento dessa consulta no XML Publisher.

1. O pedido de atualização u é traduzido pelo MPA em uma sequência de atualizações definidas sobre a VOD Pedidos\_v mostrada na Figura 13. Da mesma forma que no caso da consulta, essa tradução é simples e pode ser realizada com base apenas no XML esquema da visão Pedidos.
2. As atualizações definidas sobre a VOD Pedidos\_v são traduzidas pelo *trigger INSTEAD OF* da figura 7 em atualizações sobre o esquema do banco, sendo então processadas pelo SGBD.

```

for $i in view("Pedidos_v")/Pedidos/Pedido,
  $j in $i/listaItens
where $i/cliente = "Pedro"
update $j
insert
<item>
  <codigo>10</codigo>
  <produto>Impressora</produto>
  <quantidade>100</quantidade>
</item>

```

Fig. 12 – Atualização XQuery u

```

DECLARE
  COD_PED NUMBER;
  CURSOR PED_CUR IS SELECT P.CODIGO
                    FROM PEDIDOS_V P
                    WHERE P.CLIENTE_REF.NOME = 'Pedro';

BEGIN
  OPEN PED_CUR;
  LOOP
    FETCH PED_CUR INTO COD_PED;
    EXIT WHEN PED_CUR%NOTFOUND;
    INSERT INTO TABLE(SELECT PLISTAITEM
                      FROM PEDIDOS_V P
                      WHERE P.CODIGO = COD_PED)
          VALUES(TITEM_V(10, 'Impressora', 3));
  END LOOP;
END;

```

Fig. 13 – Tradução da atualização u

## 4. Trabalhos Relacionados

Já existem alguns sistemas como o XPeranto [3], o Silkroute [5] e o SQL Server 2000 [6] que dão suporte a publicação de dados relacionais através de visões XML. Esses três sistemas apresentam algumas limitações: (i) aplica-se somente a bancos de dados relacionais; (ii) existe uma forte dependência entre a visão XML Canônica/Esquema Anotado e o banco de dados de modo que uma mudança no banco de dados requer a redefinição das Visões XML; (iii) no XPeranto e no Silkroute, a consulta XQuery que mapeia os elementos da Visão XML Canônica em elementos da Visão XML geralmente é complexa, o que requer conhecimentos avançados de XQuery do projetista; (iv) no XPeranto e SQL Server 2000, não há garantia de processamento

eficiente das consultas e não tratam de atualização; e (v) no SQL Server 2000, a capacidade de consulta é bastante limitada, pois XPath suporta apenas seleção e projeção de elementos, mas não reestruturação, como em *XQuery*. Além desses sistemas, podemos citar [1] e [4] como soluções relacionadas ao trabalho proposto.

## 5. Conclusões

O *XML Publisher* é uma solução simples e eficiente para consultar dados relacionais ou objeto-relacionais através de uma visão XML. Conforme observado anteriormente, a publicação de dados objeto-relacionais através de visões XML não é suportada por outros sistemas já existentes. Outra vantagem do Framework proposto é que permite também a atualização de banco de dados relacionais e objeto relacionais através de visões XML. No momento, o framework suporta somente bases de dados Oracle. O passo seguinte é estender suas funcionalidade para outros bancos de dados.

## 6. Referências

- [1] BRAGANHOLO, Vanessa; DAVIDSON, Susan; HEUSER, Carlos. UXQuery: building updatable XML views over relational databases. In: Proceedings of SBBD - Brazilian Symposium on Databases, Manaus, AM, Brazil, 2003
- [2] Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data Exchange: Semantics and Query Answering. In ICDT, pages 207–224, 2003.
- [3] J.Shanmugasundaram, et al. “XPERANTO: Bridging Relational Tecnology and XML” IBM Research Report, June 2001.
- [4] L. Wang, M. Mulchandani, and E. A. Rundensteiner. Updating XQuery Views Published over Relational Data: A Round-trip Case Study. In Proc. of XML Database Symposium, Berlin, Germany, Sept. 2003.
- [5] M. Fernández, Y. Kadiyska, D. Suciu., A. Morishima and W. Tan. “SilkRoute : A Framework for Publishing Relational Data in XML”. ACM Transactions on Database Systems, Vol. 27, Nº. 4, December 2002, Pages 438-493.
- [6] M. Rys. “Bringing the Internet to Your Database: Using SQL Server 2000 e XML to Build Loosely-Coupled Systems”. In Proceedings of the International Conference on Data Engineering. Heidelberg, Germany, 2001, pp. 465-472.
- [7] Madhavan, J., Bernestein, P., Rahm, E.: Generic Schema Matching with Cupid. In: Proc. Of VLDB, p. 49-58, 2001.
- [8] Oliveira, Wamberg Claucon Chaves de: Atualizando Banco de Dados Objeto Relacionais através de Visões XML. Dissertação de Mestrado, Universidade Federal do Ceará. Em andamento.
- [9] Oracle Corporation. Disponível em: <http://technet.oracle.com>. (Acessado: 15 de julho 2004)
- [10] Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. VLDB Journal, 10(4):334–350, 2001.
- [11] Santos, L.A.L.: *XML Publisher*: Um framework para Publicar Dados Objeto Relacionais em XML. Dissertação de Mestrado, Universidade Federal do Ceará.
- [12] Tatarinovi, I.; Ives, Z.G.; Halevy, A .Y.; Weld, D.S. "Updating XML". In Proceedings of SIGMOD 2001, Maio de 2001.
- [13] Vidal, V.M.P, Lóscio, B.F.: Solving the Problem of Semantic Heterogeneity in Defining Mediator Update Translators. In Proc. of 18th Intern. Conf. on Conceptual Modeling, Paris, France, p.293-308, 1999.
- [14] World Wide Web Consortium, “XQuery 1.0”, W3C Working Draft, 12 November 2003.

## Eris: Um protótipo para consulta por similaridade a bases de dados XML<sup>1</sup>

Andrei Enéas Nunes Lima    Carina F. Dorneles    Carlos A. Heuser

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)  
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

[aenlima@inf.ufrgs.br](mailto:aenlima@inf.ufrgs.br) | [dorneles@inf.ufrgs.br](mailto:dorneles@inf.ufrgs.br) | [heuser@inf.ufrgs.br](mailto:heuser@inf.ufrgs.br)

**Resumo.** Este artigo descreve o Eris, uma ferramenta destinada a realizar consultas por similaridade em bases de dados XML. A ferramenta foi implementada em Java e possibilita a escolha das funções de similaridade, bem como o threshold usado para a criação do ranking dos resultados. Todas as instâncias constantes em uma base XML são processadas e comparadas com uma dada consulta, na tentativa de encontrar aquelas instâncias que mais se aproximam da requisição do usuário.

### 1 Introdução

A intensificação do uso da Internet tem feito com que milhões de dados sejam trocados diariamente. Tais dados são, em sua maior parte, criados e manipulados diretamente por pessoas, contendo erros de vários tipos. Ou seja, o formato não é rígido e permite grandes variações: por exemplo, datas podem ser escritas de várias maneiras diferentes, como *dia/mês/ano* ou *mês/dia/ano*, ou *ano* com quatro ou dois dígitos, ou *mês* como *12* ou *dezembro*. Todos esses formatos estão corretos, mas são difíceis de serem pesquisados por uma busca padrão. Outro exemplo seria os diferentes formatos para nomes, principalmente de pessoas, que podem apresentar iniciais, abreviações, particularidades de cada idioma, etc.

A solução mais simples seria aplicar regras restringindo as possibilidades de armazenar uma informação, como normalmente feito em bancos de dados, onde existe um formato fixo para o armazenamento de datas, por exemplo. No entanto, mesmo em bancos de dados tradicionais o armazenamento dos dados é uma tarefa que exige intenso planejamento e várias estruturas de controle. Uma solução desse tipo se torna praticamente impossível quando direcionada para um campo tão vasto e dinâmico como a Internet, dada a dificuldade que seria manter um formato fixo para os dados existentes.

Outra possibilidade é permitir o uso de consultas com argumentos de busca vagos. O resultado de tais consultas são valores **similares** aos solicitados pelo usuário. As funções que calculam a similaridade entre dados usualmente retornam um valor numérico indicativo dessa similaridade. Este valor pode ser usado para ordenação e até mesmo para restringir o conjunto de resultados, através do uso de *threshold*<sup>2</sup>.

---

<sup>1</sup> Este trabalho foi desenvolvido no Projeto Gerindo, processo Nr. 55.2087/02-05 - CTInfo/CNPq

<sup>2</sup> Ponto de corte para o *ranking* dos resultados.

Uma das características do **Eris** é permitir consultas com argumentos vagos. O cálculo de similaridade é realizado tendo em vista dois elementos distintos: elementos **atômicos** e elementos **complexos**, tais como encontrados em documentos XML. Os elementos atômicos contêm como valor *strings* (nomes, datas, endereços, etc), já os elementos complexos contêm como valor outros elementos. A definição das métricas para o cálculo da similaridade entre esses elementos pode ser encontrada em [1]. Para o cálculo de elementos atômicos, definiram-se métricas para cada tipo de elemento; o cálculo de elementos complexos leva em conta a estrutura do elemento e o que o usuário deseja com a consulta. Os elementos complexos podem ser divididos em **tuplas** ou **coleções**. Tuplas são os elementos que contêm elementos com nomes diferentes, enquanto coleções contêm elementos de mesmo nome. As coleções podem ser tratadas como **listas**, se a ordem em que as informações aparecem são importantes para o usuário, ou como **conjuntos**, caso a ordem não importe. Também foi definido o conceito de **sub-estrutura**, onde informações parciais podem ser utilizadas na consulta. Um exemplo seria um **sub-conjunto**. Neste caso a similaridade é calculada usando como parâmetros um elemento XML da base, considerado conjunto, e a consulta como parte dos membros deste conjunto, por exemplo, contendo o nome de alguns autores de um artigo, mas não todos.

<pre> - &lt;REFERENCES&gt; - &lt;REF&gt;   &lt;TYPE&gt;BOOK&lt;/TYPE&gt; - &lt;AUTHORS&gt;   - &lt;AUTHOR&gt;     &lt;LAST&gt;Abiteboul&lt;/LAST&gt;     &lt;PREF&gt;S.&lt;/PREF&gt;   &lt;/AUTHOR&gt;   - &lt;AUTHOR&gt;     &lt;LAST&gt;Buneman&lt;/LAST&gt;     &lt;PREF&gt;P.&lt;/PREF&gt;   &lt;/AUTHOR&gt;   - &lt;AUTHOR&gt;     &lt;LAST&gt;Suciu&lt;/LAST&gt;     &lt;PREF&gt;D.&lt;/PREF&gt;   &lt;/AUTHOR&gt; &lt;/AUTHORS&gt; - &lt;DATE&gt;   &lt;YEAR&gt;1999&lt;/YEAR&gt; &lt;/DATE&gt; &lt;TITLE&gt;'`Data on the Web: From Relations to Semistructured Data and XML'&lt;/TITLE&gt; &lt;PUBLISHER&gt;Morgan Kaufmann Pub.&lt;/PUBLISHER&gt; &lt;REFNUM&gt;BOOK:ABS:99&lt;/REFNUM&gt; &lt;/REF&gt; &lt;/REFERENCES&gt; </pre>	<pre> - &lt;REFERENCES&gt; - &lt;REF&gt;   &lt;TYPE&gt;BOOK&lt;/TYPE&gt; - &lt;AUTHORS&gt;   - &lt;AUTHOR&gt;     &lt;LAST&gt;Abiteboul&lt;/LAST&gt;     &lt;PREF&gt;Serge&lt;/PREF&gt;   &lt;/AUTHOR&gt;   - &lt;AUTHOR&gt;     &lt;LAST&gt;Buneman&lt;/LAST&gt;     &lt;PREF&gt;Peter&lt;/PREF&gt;   &lt;/AUTHOR&gt; &lt;/AUTHORS&gt; - &lt;DATE&gt;   &lt;YEAR&gt;2000&lt;/YEAR&gt; &lt;/DATE&gt; &lt;TITLE&gt;Data on the Web: from relations to semistructured data and XML&lt;/TITLE&gt; &lt;PUBLISHER&gt;Morgan Kaufmann Publishers Inc.&lt;/PUBLISHER&gt; &lt;REFNUM&gt;324746&lt;/REFNUM&gt; &lt;/REF&gt; &lt;/REFERENCES&gt; </pre>
--	---

Figura 1. Exemplo de duas instâncias de arquivos XML.

Algumas ferramentas já foram propostas, entre as quais podemos destacar o **dtSearch** [3] e o **Lucene** [4]. O Lucene é uma *API* que permite a indexação e busca de informações em vários tipos de dados, ele utiliza o modelo vetorial. O **dtSearch** é uma ferramenta comercial que permite a indexação e consulta de vários tipos de documentos, não somente XML. Embora as duas ferramentas atendam à maioria das necessidades de consulta, elas não tratam dois pontos: (i) as duas fazem consultas por palavras-chave, sem tolerar variações na grafia das palavras digitadas; (ii) não tratam similaridade de coleções de valores, como listas, sublistas, conjuntos, etc.

A partir da definição das métricas acima descritas, foi criado o **Eris**, uma ferramenta desenvolvida em *Java* que retorna as instâncias mais similares a uma

consulta enviada pelo usuário. Com o Eris é possível descobrir o número de instâncias similares existentes em uma base e ainda definir qual o *threshold* a ser aplicado.

## 2 Eris

As consultas são executadas sobre bancos de dados XML, um exemplo é apresentado na Figura 1, onde observa-se duas instâncias representando o mesmo livro, mas de fontes diferentes. Nota-se a diferença entre os títulos e, principalmente, na lista de autores, onde nem todos os autores que aparecem na instância da esquerda estão relacionados na instância à direita. O Eris permite tratar isso através do uso de listas e coleções.

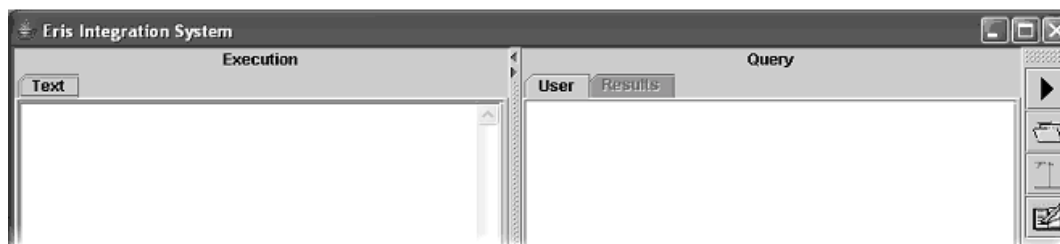


Figura 2. Visão parcial da tela principal do Eris

A tela principal do Eris é mostrada na Figura 2. A área à esquerda serve para informar o andamento da execução e a área à direita é destinada à construção da consulta a ser executada. A barra de ferramenta à direita apresenta ícones para as seguintes tarefas, respectivamente: i) execução da consulta; ii) localização da base XML; iii) definição do threshold e; iv) apresentação do log da execução da consulta.

```
- <XML>
- <REFERENCES>
- <REF>
- <AUTHORS weight="0.2" searchAs="SUBSET">
- <AUTHOR>
- <LAST measure="EDIT_DISTANCE">widom</LAST>
</AUTHOR>
- <AUTHOR>
- <LAST measure="EDIT_DISTANCE">goldman</LAST>
</AUTHOR>
</AUTHORS>
<TITLE weight="0.5" measure="EDIT_DISTANCE">DataGuides: Enabling Query Formulation and Optimization in
Semistructured Databases</TITLE>
- <DATE weight="0.2">
- <YEAR measure="DATE_SIMILARITY">1997</YEAR>
<MONTH>aug</MONTH>
</DATE>
<BOOKTITLE weight="0.1" measure="EDIT_DISTANCE">Proceedings of 23rd International Conference on Very Large
Data Bases</BOOKTITLE>
</REF>
</REFERENCES>
</XML>
```

Figura 3. Exemplo de consulta.

### 2.1 Formato das consultas

O formato das consultas segue a estrutura das instâncias da base de dados XML. Um exemplo de consulta é apresentada na Figura 3. Como pode ser observado, foram criados alguns atributos especiais para indicar como a consulta deve ser processada. Esses atributos possuem diversas funcionalidades e são explicados na Tabela 1.

Atributo	Formato	Valores possíveis para	Valor default
<i>searchAs</i>	<elemento searchAs= "x">	<p>atom: indica um elemento folha, ou seja, um elemento que contém dados textuais;</p> <p>list: permite fazer pesquisas com elementos do mesmo tipo e ordenados, seguindo a ordenação contida na base de dados;</p> <p>sublist: permite fazer pesquisas com elementos parciais do mesmo tipo e ordenados, seguindo a ordenação contida na base de dados;</p> <p>set: fazer pesquisas com elementos do mesmo tipo, sem levar em conta sua ordenação;</p> <p>subset: permite fazer pesquisas com elementos parciais do mesmo tipo, sem levar em conta sua ordenação;</p> <p>tuple: permite fazer consultas com elementos parciais de tipos diferentes, sem levar em conta sua ordenação.</p>	o valor <i>default</i> de cada elemento, caso não seja explicitamente indicado, é atom para elementos atômicos e tuple para os demais.
<i>Weight</i>	<elemento weight="x" >	quaisquer valores numéricos, inteiros ou não, especificando o peso associado ao objeto (utilizar ponto como separador decimal).	1
<i>Measure</i>	<elemento measure="x" >	<p>ngrams: computa similaridade entre duas <i>strings</i> que têm rearranjo de palavras;</p> <p>edit_distance: computa similaridade entre duas <i>strings</i> que têm somente erros de digitação;</p> <p>date_similarity: computa a similaridade entre datas;</p> <p>numSimPos: computa a similaridade entre dois números positivos;</p> <p>numSimNeg: computa a similaridade entre dois números negativos.</p>	Ngrams
<i>Position</i>	<elemento position="x">	qualquer valor inteiro especificando a posição do elemento na instância da base a ser comparada, se a posição especificada não existir, ele é ignorado.	

Tabela 1. Atributos especiais para a consulta.

## 2.2 Interface

Os ícones da barra de ferramentas podem ser explicados como segue.

**Execução:** este ícone executa a consulta do usuário, ou seja, a consulta é enviada ao sistema para que sejam procuradas na base instâncias semelhantes. Caso qualquer erro

seja encontrado, estes são colocados no arquivo de *log*. Se a consulta foi executada com sucesso, os resultados são mostrados.

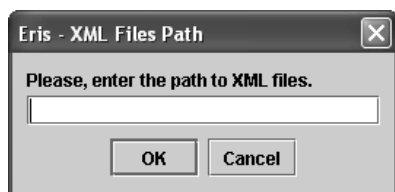


Figura 4. Caminho para os arquivos XML.

**Diretório:** o segundo ícone abre a janela (Figura 4) onde o usuário deve indicar o caminho onde estão os arquivos XML. O Eris aceita o protocolo http.



Figura 5. Escolha do threshold.

**Threshold:** o terceiro ícone abre a janela (Figura 5) onde o usuário indica o *threshold* a ser aplicado nas instâncias XML. O *threshold* pode variar de **0,2** a **0,9**; sendo que o valor *default* é de **0,6**. Valores mais altos de *threshold* são mais restritivos; as instâncias apresentadas nos resultados são mais similares à consulta, mas essa restrição também limita o número de respostas a tal ponto que valores muito elevados (0,9) podem não trazer resultados caso não exista na base instâncias muito parecidas com a consulta apresentada. Os valores mais baixos trazem mais resultados, mas isso tem um preço: caso a base seja muito grande, a máquina virtual Java pode simplesmente não suportar a quantidade de respostas, e o resultado disso é que uma exceção será disparada e nenhum resultado será mostrado.

**Log:** o quarto ícone do Eris mostra o log de execução, se houve algum erro durante o processamento das consultas, o erro estará indicado nesta janela, no formato retornado pela máquina virtual Java.

### 3 O cálculo da similaridade

Para construir o *ranking* de similaridade de documentos XML, é utilizada a similaridade do elemento raiz do documento XML. A partir de uma árvore DOM de um documento XML, a métrica de similaridade é aplicada aos elementos em um processo recursivo iniciando no elemento folha localizado mais à esquerda da árvore. Quando todos os filhos de um elemento forem processados, a métrica associada àquele elemento é aplicada, calculando a similaridade para aquele elemento. O processo continua até chegar à raiz. Dessa forma, as diversas métricas de similaridade são combinadas em um processo *bottom-up* a fim de gerar o *ranking*.

### 4 Experimentos

Para verificar a eficácia do Eris, comparou-se ele com uma implementação que utiliza a API Lucene; o mesmo conjunto de consultas foi submetido às duas implementações, funcionando sobre a mesma base de dados. A base de dados utilizada nos testes foi montada a partir de citações de arquivos bibtex, oriundas do *site The Collection of Computer Science Bibliographies*<sup>3</sup> e dos membros do **Grupo de Banco de dados da UFRGS**<sup>4</sup> somando 16.519 entradas de citações bibliográficas. As citações foram

<sup>3</sup> <http://liinwww.ira.uka.de/bibliography/Database/index.html>

<sup>4</sup> <http://metropole.inf.ufrgs.br/DBGGroup/>

convertidas para XML utilizando a ferramenta *bib2XML*<sup>5</sup>. Foram realizadas oito consultas, construídas a partir de instâncias existentes na base de dados. Por questões de espaço não é possível mostrar mais resultados, os quais podem ser encontrados em [1].

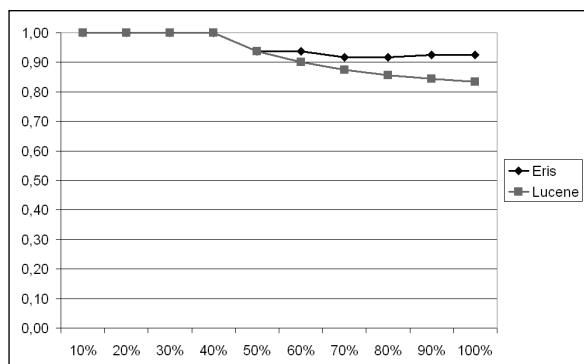


Figura 6. Comparação Eris x Lucene

Os resultados são apresentados através de uma curva de precisão/revocação. Como apresentado na Figura 7, pode-se observar que o Eris obteve resultados mais precisos à medida que a revocação aumentava, isso devido à utilização de similaridade, ao invés de comparações exatas. Além disso, o Eris se apresenta estável em instâncias onde foram usadas funções de similaridade para coleções de valores.

## 5 Conclusão e Trabalhos futuros

Este artigo apresentou o Eris, uma ferramenta funcional implementada em Java que foca no problema de consultas por similaridade em bases de dados no formato XML, onde o conteúdo de várias instâncias pode aparecer em formatos similares. Os testes realizados mostram que o Eris pode obter bons resultados. O conceito de pesos (de elementos) também foi implementado e alguns testes realizados, auxiliando na obtenção de respostas mais precisas. Por restrições de espaço não foram apresentados nesse artigo.

Como trabalhos futuros, visa-se a inclusão de outras funções de similaridade para elementos atômicos, permitindo uma flexibilidade maior na hora do usuário escolher qual o algoritmo a ser utilizado na consulta. Uma boa alternativa são as funções de similaridade apresentadas em [5]. Pesquisas envolvendo a configuração automática da consulta determinando automaticamente o tipo de algoritmo a ser utilizado e os pesos adequados também são um tópico a ser explorado.

## 6 Referências

- [1] DORNELES, Carina Friedrich; LIMA, Andrei Enéas Nunes; Heuser, Carlos Alberto; da Silva, Altigran; de Moura, Edleno S. Accessing XML data by allowing imprecise query arguments. Technical Report, RP-342, Universidade Federal do Rio Grande do Sul, 2004.
- [2] LIMA, Andrei Enéas Nunes. **Similaridade para elementos XML**. Porto Alegre: Universidade Federal do Rio Grande do Sul, 2002. (Relatório Técnico).
- [3] **dtSearch**. <http://www.dtsearch.com/>.
- [4] **Jakarta Lucene**. <http://jakarta.apache.org/lucene/docs/index.html>
- [5] Cohen, William. Ravikumar, Pradeep. Fienberg, Stephen. **A Comparison of String Metrics for Matching Names and Records**. KDD Workshop on Data Cleaning and Object Consolidation, 2003.

<sup>5</sup> <http://www.cs.duke.edu/~sprenkle/bibtex2html/>



## iDFQL - Uma ferramenta para auxílio á preparação de consultas complexas

Ana Paula Appel<sup>1</sup>, Caetano Traina Júnior<sup>1</sup>

<sup>1</sup>Departamento de Computação – Universidade de São Paulo  
Av. do Trabalhador São Carlense, 400, centro  
Caixa Postal 668, CEP 13560-970, São Carlos - SP, Brasil

anaappel@icmc.usp.br, caetano@icmc.usp.br

**Resumo.** Este artigo apresenta uma ferramenta chamada iDFQL - *interactive Data Flow Query Language* - baseada na álgebra relacional para o auxílio ao desenvolvimento de expressões de consultas complexas e apoio ao ensino da linguagem SQL e da Álgebra Relacional. Esse ambiente permite a criação e execução de consultas de forma interativa, permitindo que o processo de desenvolvimento seja acompanhado passo a passo, o que torna mais estimulante e intuitivo o aprendizado dos conceitos desta linguagem e mais fácil o desenvolvimento de consultas complexas envolvendo grande quantidades de tabelas, atributos e predicados.

### 1. Motivação

Grande parte do sucesso dos SGBDR (Sistema Gerenciador de Banco de Dados) se deve á recuperação eficiente da informação que é possível a partir de um conjunto de restrições que as linguagens de consultas definem para selecionar a informação desejada.

Um consulta pode ser construída em uma linguagem textual ou visual. Para usar uma linguagem de consulta textual como a linguagem SQL (*Structure Query Language*)[Chamberlin and Boyce, 1974], o usuário precisa aprender a sintaxe completa da linguagem (ou parte significativa dela) para expressar qualquer consulta simples ou complexa, um aspecto que em determinadas aplicações pode ser negativo. Além disso, é necessário que ele conheça como as informações estão organizadas no banco de dados. Por outro lado, as linguagens de consultas visuais [Catarci et al., 1997] apresentam um ambiente mais amigável para a recuperação de informações, permitindo que os usuários, não familiarizados com a sintaxe da linguagem de consulta textual formulem suas consultas de forma simples e intuitiva.

Uma característica que ocorre frequentemente em aplicações centradas em um SGBDR é que diversos tipos de usuários, experientes ou não, precisam recuperar informações da base de dados. Esta facilidade para o acesso dos dados requer uma grande quantidade de trabalho antes dos usuários finais poderem fazer suas consultas. Uma forma de ajudar essa classe de usuários a construir consultas mais eficientes, é oferecer um ambiente que permite ao usuário construir suas consultas de uma forma mais intuitiva.

A álgebra relacional [Codd, 1970] é uma linguagem de consulta que consiste em um conjunto de operadores que têm como entrada uma ou duas relações (tabelas) e produzem como resultado uma nova relação [O'Neil and O'Neil, 2001]. Existem muitas razões para se usar a álgebra relacional como uma linguagem de consulta, entre elas: ela é uma linguagem compacta, independente de plataforma, conceitualmente definida (base formal dos operadores do modelo relacional) e possui um alto poder de expressão.

Além disso, e o mais importante, é que ela é a base para a implementação e otimização das consultas em um SGBDR.

No contexto do ensino de base de dados, as ferramentas disponíveis estão centradas principalmente na linguagem SQL, como é o caso do SQL-Tutor [Seyed-Abbassi, 1993]. Outro aspecto bastante explorado é o suporte ao processo de projeto do modelo entidade-relacionamento tal como o DBTool [Lim and Hunter, 1992] e outras ferramentas comerciais como, Microsoft Visio 20000®, DIA®, etc. Um experimento interessante no ensino da álgebra relacional é mostrado em [Matos and Grasser, 2000], no qual o autor mostra a implementação de um parser para a álgebra relacional - o RELAX (*Relational Algebra Explorer*) feita pelos alunos como trabalho na disciplina de base de dados. Usando uma interface gráfica, ele possibilita a escolha dos operadores algébricos e a definição da consulta (mostrada em formato textual) que pode ser analisada e resolvida.

Este artigo apresenta uma ambiente interativo para a construção de consultas em álgebra relacional chamado iDFQL - *Interactive Data Flow Query Language*. A ferramenta iDFQL oferece um ambiente interativo para a construção de consultas baseadas em álgebra relacional usando elementos gráficos. A principal característica da iDFQL é o uso do paradigma icônico e de diagrama de fluxo para representação da consulta. Neste artigo a seção 2 apresenta a ferramenta iDFQL, a seção 3 apresenta as conclusões deste trabalho.

## 2. A Ferramenta iDFQL

A ferramenta iDFQL combina o uso de ícones para representar os operadores da álgebra relacional e diagramas de fluxo para representar a consulta. Um diagrama de fluxo é uma representação gráfica de um processo (ação) descrito, ou analisado. Esses processos são conectados entre si através de linhas (formando uma cadeia) que representa um relacionamento seqüencial ou de dependência.

Cada operação de recuperação de informação pode ser considerada um processo em um SGBDR. Desta maneira os operadores da álgebra relacional podem ser representados como processos identificados por ícones em um diagrama de fluxo. A conexão entre os ícones produz um diagrama de fluxo que representa a consulta. Essa representação permite a criação interativa de consultas de uma maneira mais intuitiva.

Além dos operadores tradicionais da álgebra, a ferramenta usa operadores de entrada e saída de dados que funcionam como produtores e consumidores de dados. Além disso, alguns dos operadores tradicionais da álgebra relacional foram conceitualmente modificados para aumentar sua flexibilidade de utilização na ferramenta iDFQL da seguinte maneira: os operadores de junção e o operador de seleção da álgebra relacional incorporam a especificação de suas respectivas condições de junção e de seleção como parte do operador. Na versão para compor os diagramas de fluxo da iDFQL, esses operadores foram separados para que a especificação dessas condições seja feita externamente, a criando um novo operador - o operador de “condição”. O operador de projeção também teve a sua lista de atributos especificada externamente - criando-se o operador “lista de atributos”.

Assim, enquanto na álgebra relacional todas as variáveis são sempre tabelas, na versão para diagramas de fluxo existem outros tipos de dados que são “conduzidos” por um fluxo (ligação entre dois operadores no diagrama) específico. Por exemplo, o operador de condição produz “condições” e operador lista de atributos produz uma “lista de atributos”. Os conceitos de diagramas de fluxo são usados para conectar vários operadores

e criar um diagrama representando uma consulta. Verificou-se que são necessários três tipos de fluxos para expressar uma consulta em um diagrama de fluxo de dados, que são:

- **Fluxo de Dados** - Esses fluxos representam os dados propriamente ditos que são processados a partir das tabelas armazenadas na base até que se obtenham as respostas à consulta;
- **Fluxo de Condições** - Esses fluxos representam as condições que são utilizadas pelos operadores de seleção e pelos vários operadores de junção;
- **Fluxo de Atributos** - Esses fluxos representam conjuntos de atributos que são utilizados, por exemplo, pelos operadores de projeção;

Cada operador possui um ou mais tipos de fluxo sendo que cada fluxo pode ainda ser de entrada ou de saída. O operador que possui um certo tipo de fluxo de saída só pode ser conectado a um operador que possui o mesmo tipo de fluxo de entrada. Cada fluxo é representado por uma cor diferente: azul para fluxo de tuplas ou dados, vermelho para o fluxo de condições e amarelo para o fluxo de atributos. Após receber os dados dos fluxos de entrada, estes são processados segundo a operação representada pelo ícone e enviados ao próximo operador por meio do fluxo de saída.

Essa ferramenta é composta por um módulo de edição de consultas, que é formado basicamente por uma coleção de operadores e por um painel no qual o usuário pode montar o diagrama de fluxo que representa uma consulta, instanciando, parametrizando e conectando os operadores. Além desse a ferramenta também possui o módulo de execução de consultas, que permite ao usuário fazer uma consulta completa, intermediária ou ainda um *sampling* exploratório.

## 2.1. Módulo de Edição

A iDFQL pode ser definida como uma ferramenta de consulta a um SGBD Relacional baseada na álgebra relacional, que possui um editor e um executor de consultas. O editor possui uma interface de interação com o usuário formado basicamente por uma coleção de operadores e um painel no qual o usuário monta o diagrama de fluxo que representa sua consulta. A tela principal do ambiente é mostrada na Figura 1.

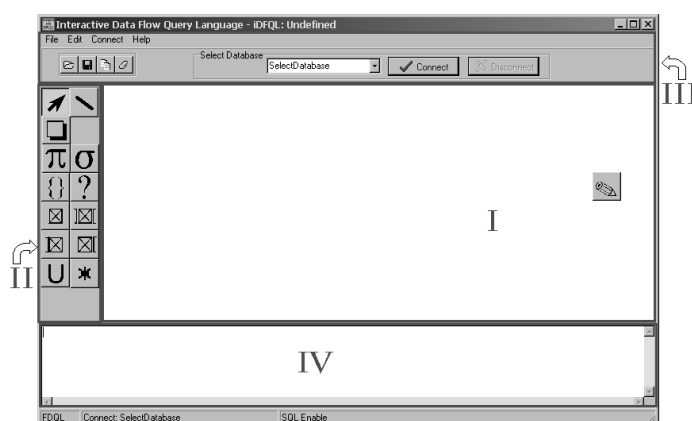


Figura 1: Tela principal da ferramenta iDFQL

A Figura 1 mostra as principais áreas de interação da iDFQL: a área I corresponde ao painel onde a consulta é construída “carimbando-se” os operadores; a área II contém a coleção de operadores disponíveis; a área III mostra os menus e opções para a operação da ferramenta, como por exemplo o comando de conexão com a base de dados que será utilizada para a consulta; a área IV é usada para mensagens da ferramenta (por exemplo, nessa área são mostradas as consultas em  $\text{SQL}$ , montadas automaticamente quando as

mesmas são submetidas ao SGBDR que recupera a expressão representada em álgebra relacional através do comando em SQL).

A representação diagramática da iDFQL permite uma melhor percepção da representação das consultas enquanto mantém todas as propriedades da linguagem. Para construir suas consultas, primeiramente deve se estabelecer a conexão com uma das bases de dados registradas no BDE ou ODBC usando a ferramenta de conexão (área III) apresentado na Figura 1.

Posteriormente, devem ser escolhidos os operadores que irão compor a consulta (área II) e carimbá-los no painel (área I). Os operadores são instanciados clicando-se no seu ícone e carimbando-os como uma instância do mesmo no painel.

Enquanto “monta” a sua consulta, o usuário deve também parametrizar os operadores utilizados. Isso corresponde a definir a tabela que deve ser lida em cada operador de leitura de tabela, definir as condições nos operadores de condição e os atributos nos operadores de lista de atributos. Essa parametrização é feita em uma janela específica, acessível através de um toque duplo sobre o ícone do operador no painel, por exemplo, a escolha da tabela a ser lida num operador de leitura mostrada na Figura 2. Após a parametrização ser feita, o valor definido é mostrado abaixo do operador, como por exemplo, o nome da tabela, atributos ou condições.

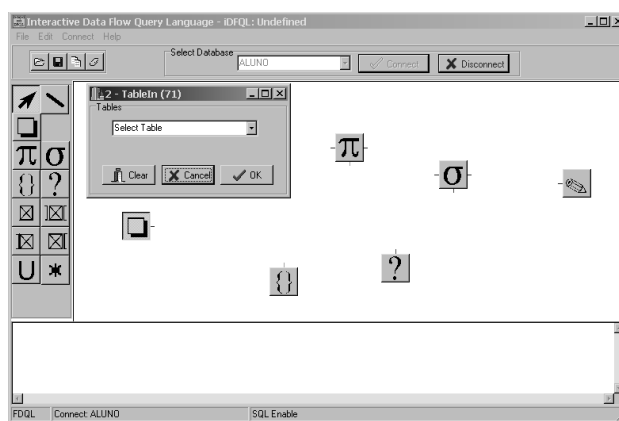


Figura 2: Parametrização do operador “Tabela de Entrada”

Além da parametrização é necessário que os operadores estejam conectados entre si para que uma consulta possa ser executada. Cada conexão entre dois operadores feita pela ferramenta iDFQL é representada por uma linha entre eles usando a cor específica para o tipo de fluxo. A Figura 3 mostra uma consulta completa já construída utilizando a ferramenta iDFQL.

## 2.2. Módulo de Execução

Após serem explicitados os parâmetros e criadas as conexões dos operadores que compõem diagrama de fluxo como mostra a Figura 3, o usuário pode então executar a consulta criada. Existem implementadas três maneiras de solicitar a execução de uma consulta na ferramenta iDFQL: visualização da execução completa, dos resultados intermediários, ou por amostragem.

A visualização de resultados intermediários pode ser solicitada em cada operador do diagrama. Para isso o usuário deve indicar o operador para o qual pretende verificar o resultado. Esse procedimento é feito clicando-se com o botão direito sobre o operador desejado e escolhendo a opção *query* no menu *pop-up*. O resultado desta consulta é apresentado em uma nova janela. Uma visualização intermediária dos resultados, na qual foi indicada a visualização do operador de junção, é mostrada na Figura 4.

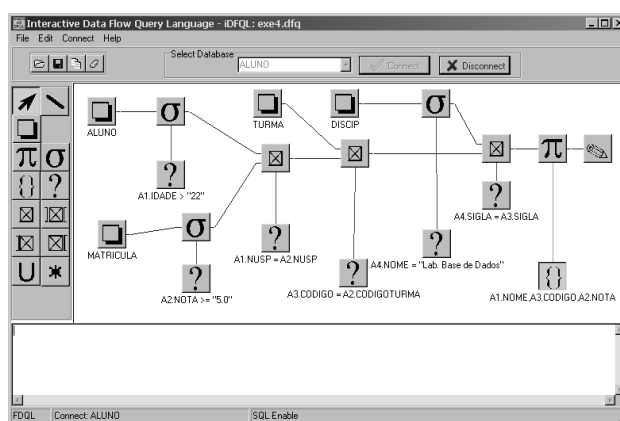


Figura 3: Consulta representada na ferramenta iDFQL

CODIGOTURMA	NUSP	NOTA	NOME	NUSP_1	IDADE
101	8901	10	Carla	8901	23
101	8901	7	Catrina	8901	23
102	5678	7	Catrina	5678	23
102	8901	9	Catrina	8901	23
104	7890	9	Corina	7890	25
104	5678	8	Catrina	5678	23

```

SELECT A2.CODIGOTURMA, A2.NUSP, A2.NOTA, A1.NOME, A1.NUSP, A1.IDADE, A1.CIDADE, A3.SIGLA, A3.NUMERO, A3.CODIGO, A3.NNALUNC
FROM MATRICULA A2 JOIN ALLUNO A1 ON A1.NUSP = A2.NUSP JOIN TURMA A3 ON A3.CODIGO = A2.CODIGOTURMA
WHERE A1.IDADE >= 22 AND A2.NOTA >= 5.0
    
```

Figura 4: Resultado de uma visualização intermediária dos resultados - operador de junção

A visualização da execução completa da consulta é feita de forma análoga à visualização dos resultados intermediários, bastando apenas o usuário fazer o pedido de consulta no operador “saída de dados”. A opção de consulta por amostragem retorna apenas uma pequena amostra do resultado solicitado, isto é uma percentagem do total de tuplas que seria retornado. A opção de execução completa é útil para obter respostas a consultas reais, enquanto as opções de execução intermediária e por amostragem são úteis para auxiliar a depuração e ensino da álgebra relacional em base de dados reais, que em geral tem grande quantidade de tuplas.

Nota-se que, em qualquer uma das visualizações a área IV da ferramenta (Figura 1) mostra a consulta equivalente à efetuada para a execução do diagrama, em um comando SQL. Essa opção pode ser habilitada ou não pelo usuário. Além disso, o resultado pode ser gravado em um arquivo externo ou imprimí-lo.

### 3. Conclusão

Uma consulta é representada (em iDFQL) de uma maneira mais natural por usar uma linguagem de consulta visual, já que a representação por ícones e a navegação em diagramas substituem a complexidade de utilizar comandos numa relação algébrica. Desta forma, linguagens de consultas visuais são mais flexíveis que linguagens de consultas monolíticas tradicionais, tal como SQL.

A iDFQL também permite um maior entendimento sobre o relacionamento entre a álgebra relacional e a linguagem SQL<sub>23</sub> durante o processo de ensino/aprendizado

dessa linguagem. Ela também permite o auxílio à criação de consultas que são extensas e complexas devido a presença de muitas junções e condições, já que permite que o desenvolvimento dessas consultas seja feito de forma interativa, verificando os resultados intermediários produzidos em cada operador, além da possibilidade de verificar diversas alternativas de consultas pela combinação dos operadores e alterar os parâmetros estabelecidos sem ter que reescrever a consulta.

A iDFQL também apresenta benefícios para os professores que podem tornar o processo de ensino mais fácil e agradável do que o tradicional. Apresentando aos alunos experiências práticas, o professor pode reduzir o nível de abstração pelo estímulo da habilidade de quebrar grandes consultas em sub-consultas menores e seguir um processo lógico no encadeamento dessas sub-consultas.

Como desenvolvimento futuro da ferramenta, pretende-se que ela seja estendida para aumentar os recursos de execução por amostragem, suportar as operações de agregados, bem como estendê-la para adição de operadores de processos de mineração de dados.

A ferramenta iDFQL está sendo utilizada como suporte às disciplinas de base de dados do ICMC - USP tendo já se constatado uma receptividade muito boa, e uma melhoria significativa do processo de aprendizado dos alunos, constatado pela aplicação de técnicas de avaliação do aprendizado [Appel et al., 2004]. A ferramenta iDFQL é *free* e está disponível para download no endereço: <http://gbdi.icmc.usp.br/membros/anaappel/idfql.html>.

## Referências

- Appel, A. P., Silva, E. Q., Traina, Jr., C., and Traina, A. J. M. (2004). iDFQL - a query-based tool to help the teaching process of the relational algebra. In *Proceedings of World Congress on Engineering and Technology Education (WCETE2004)*, pages 429–433, Santos, SP, Brasil. COPEC, Council of Researches in Education and Sciences.
- Catarci, T., Costabile, M. F., Levialdi, S., and Batini, C. (1997). Visual query systems for databases: A survey. *J. Visual Languages and Computing*, 8(2):215–260.
- Chamberlin, D. D. and Boyce, R. F. (1974). Sequel: A structured english query language. In Rustin, R., editor, *ACM-SIGMOD Workshop on Data Description, Access and Control*, volume 1, pages 249–264, Ann Arbor, MI. ACM Press.
- Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.
- Lim, B. B. L. and Hunter, R. (1992). Dbtool: a graphical database design tool for an introductory database course. In *Proceedings of the twenty-third SIGCSE technical symposium on Computer science education*, pages 24–27, Kansas City, Missouri, United States. ACM Press.
- Matos, V. and Grasser, R. (2000). Relax - the relational algebra pocket calculator project. *ACM SIGCSE Bulletin*, 32(4):40–44.
- O’Neil, P. and O’Neil, E. (2001). *Database - Principles, Programming and Performance*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, San Francisco, CA, 2nd edition.
- Seyed-Abbassi, B. (1993). A sql project as a learning method in a database course. In *Proceedings of the 1993 conference on Computer personnel research*, pages 291–297, St Louis, Missouri, United States. ACM Press.

## ROSA: Um Ambiente de Desenvolvimento e Armazenamento de Conteúdos Didáticos com Acesso Semântico

Ana Maria de C. Moura<sup>1</sup>, Fabio Porto<sup>2</sup>, Abílio Fernandes<sup>1</sup>, Adriana Fernandez<sup>1</sup>,  
Fábio Coutinho<sup>1</sup>

<sup>1</sup>Instituto Militar de Engenharia  
Depto de Engenharia de Sistemas – Rio de Janeiro

<sup>2</sup>EPFL-IC- Ecole Polytechnique Federal de Lausanne

[anamoura, fabio]@de9ime.eb.br, fabio.porto@epfl.ch,  
adriana@ensino.eb.br

**Abstract.** ROSA é um middleware para o armazenamento e recuperação a Objetos de Aprendizagem(OA) com suporte semântico. OAs são descritos segundo o padrão LOM IEEE, e associados através de predicados nomeados. O sistema utiliza o SGBD nativo XML Tamino para persistência dos OAs e possibilita navegação e pesquisa aos objetos por meio de uma interface amigável via Web.

### 1. Introdução

O middleware ROSA<sup>1</sup> (Repository of Objects with Semantic Access) visa a gerência de documentos com suporte semântico à pesquisas e gerência de metadados. ROSA é um sistema voltado para a área de Ensino a Distância (EAD), que visa auxiliar profissionais da área educacional, ajudando-lhes a descobrir conteúdos didáticos armazenados no sistema, que forneçam subsídios para preparação de suas aulas ou conteúdos instrucionais. Para isso o sistema armazena Objetos de Aprendizagem (OA) (*learning-objects*), que representam de fato os conteúdos instrucionais, acrescidos de um conjunto de características e propriedades, também denominadas de metadados, além de um conjunto de associações (ou predicados) que expressam os relacionamentos que um OA tem com outros OAs.

A partir de um modelo de dados e de uma arquitetura bem definida, consultas de conotação mais semântica podem ser feitas ao sistema, além das mais simples, tais como as de recuperação de OAs a partir de suas propriedades mais específicas. Como exemplo de consultas semânticas, pode-se citar: *que disciplinas são pré-requisitos de Banco de Dados?* Ou ainda, *que tópicos compreendem a disciplina Banco de Dados?* Nesses exemplos, *pré-requisito* e *compreendem* fazem parte de um conjunto de predicados pré-definidos que associam os diversos OAs. O sistema suporta ainda a inclusão de Tesouros, isto é, um vocabulário controlado de termos em um determinado domínio, para apoiar as consultas dos usuários. Esse sistema utiliza-se de tecnologias recentes, em consonância com a tendência atual dos projetos desenvolvidos no escopo da Web Semântica, tais como RDF (Resource Description Framework)[RDF], padrões de Metadados, tecnologia Java, XML. Esses pontos são relevantes quando se almeja a interoperabilidade e integração de recursos na Web.

---

<sup>1</sup> Homepage ROSA: <http://www.des.ime.eb.br/~Rosa>; ROSA: <http://200.20.120.133:8090/rosa>

O restante do artigo está organizado da seguinte forma: a seção 2 apresenta um estudo de caso que servirá de exemplo ao longo do texto; a seção 3 descreve a arquitetura e o modelo de dados ROSA; a seção 4 apresenta a ferramenta e a seção 5 conclui o artigo.

## 2. Modelagem de OAs

OAs têm sido apontados na literatura como uma solução eficiente para os problemas de padronização e baixo custo de desenvolvimento de conteúdo instrucional. Um OA é uma coleção re-utilizável de material usado para apresentar e dar apoio a um único objetivo de aprendizagem. Pode representar um módulo ou lição que ensina um conceito específico, fato, procedimento, processo ou princípio. Assim, é necessário que conteúdos instrucionais sejam criados de forma modular e padronizada, para o qual têm sido criados padrões de metadados específicos para o cadastramento das propriedades de um OA. O padrão adotado para cadastramento dos metadados de cada OA foi o padrão LOM [IEEE02]. Este padrão é constituído de um conjunto de elementos de dados classificados segundo uma hierarquia (nós intermediários e folhas), que no primeiro nível permitem descrever OAs segundo suas características educacionais, técnicas, ciclo de vida, descrição, autoria, relação com outros OAs, etc.

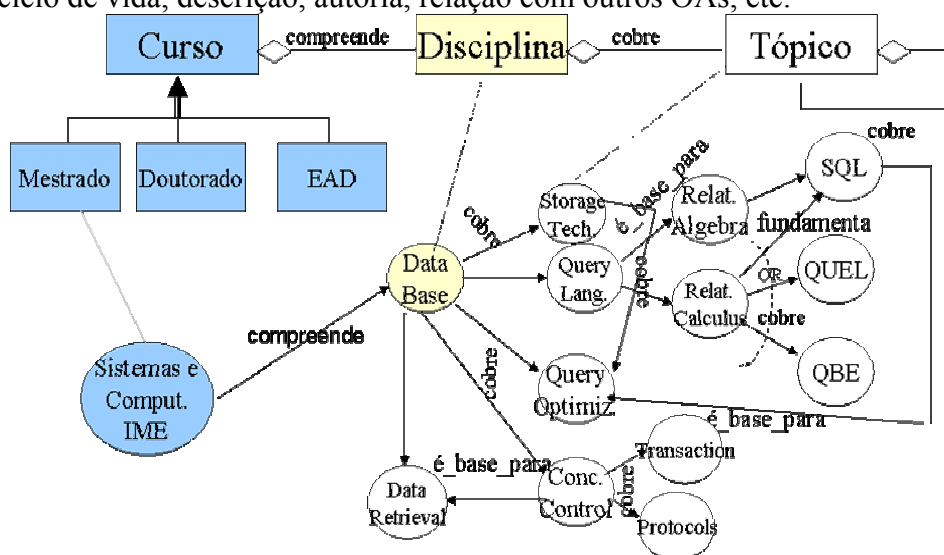
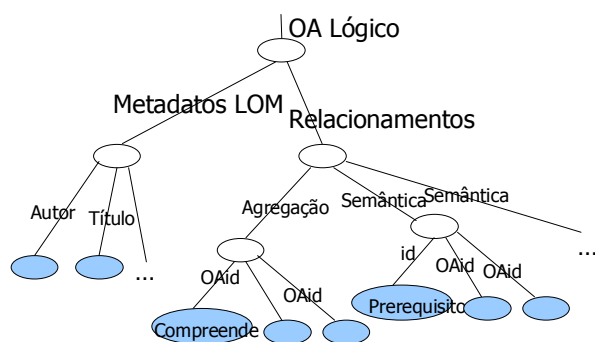


Figura 1: Mapa Conceitual do Curso de Sistemas de Computação

Mapa conceitual é uma forma de representação gráfica utilizada para visualizar os relacionamentos ou associações entre OAs, tratados nesse trabalho através de predicados. Um curso é armazenado no ROSA segundo seu Mapa Conceitual, que organiza OAs de acordo com um elenco de predicados definidos pelo usuário. OAs são classificados em Lógicos e Físicos, onde o primeiro diz respeito aos conceitos a serem apresentados e o segundo representa os conteúdos a serem ministrados. No exemplo que adotaremos ao longo desta seção, é importante definir a seguinte situação, em consonância com os cursos EAD (Ensino a Distância) oferecidos no nosso mundo real: um curso é constituído de várias disciplinas, sendo que estas compreendem vários tópicos, que por sua vez podem abranger ou apenas associar outros tópicos. Essas classes serão utilizadas na representação do mapa conceitual mostrado na Figura 1, que retrata um Mapa Conceitual parcial do Curso de Mestrado em Sistemas e Computação do IME. Nesta figura, elipses representam OAs, arestas rotuladas direcionadas





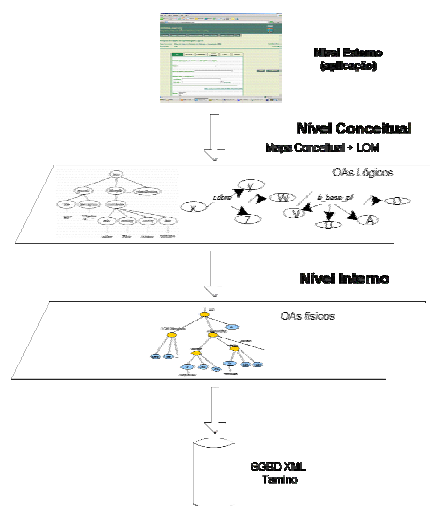
representam Predicados e linhas tracejadas classificam recursos segundo as respectivas classes. As cores das elipses indicam respectivamente o papel de cada OA no mapa, i.e., se os mesmos são cursos, disciplinas ou tópicos. Relacionamentos

**Figura 2: Estrutura de um OA**

são expressos através de predicados que podem ser de dois tipos: predicados de agregação (compreende, é formado por, abrange, possui, etc.) e predicados de associação ou semânticos (pré-requisito, fundamenta, implica, é equivalente\_a, etc.). A Figura 2 apresenta a estrutura gráfica de um OA formado pelos seus metadatos e relacionamentos semânticos e de agregação.

## 2. O Sistema ROSA

O sistema ROSA está especificado em três níveis segundo a arquitetura tradicional de BDs: nível externo ou de aplicação, nível conceitual e nível interno ou físico. No nível externo é fornecida uma interface de alto nível ao usuário que pode navegar por mapas a ele associados ou formular consultas com auxílio de uma interface *à la QBE*. O nível lógico corresponde à representação de mapas conceituais e da especificação de OAs lógicos e físicos [Porto et al. 2003]. Finalmente, no nível físico, são especificadas as informações do modelo lógico, representadas em XML armazenadas no SGBD XML nativo Tamino. A figura 3 ilustra os três níveis mencionados.

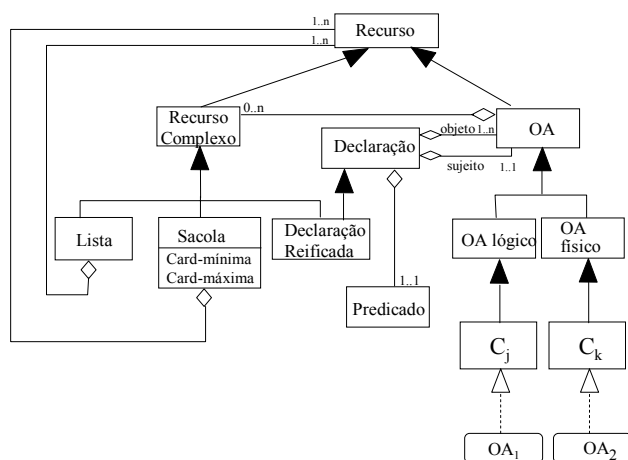


**Figura 3: Arquitetura do ROSA**

gerência de usuários, OAs, Mapas Conceituais e Tesouros. O usuário Convidado pode usufruir apenas dos serviços de navegação e pesquisa de OAs, não podendo incluir ou modificar conteúdos didáticos. Já o usuário Conteudista é responsável pela criação e manutenção dos seus mapas conceituais, conteúdos didáticos e tesouros de domínio, usufruindo também das funcionalidades de navegação e pesquisa.

A navegação e pesquisa a OAs no ROSA são realizadas através de uma ferramenta navegacional, tendo como base um modelo de dados bem definido [Porto et al. 2004]. O objetivo principal é prover um ambiente amigável ao instrutor, permitindo que este possa criar seus próprios conteúdos instrucionais ou pesquisar outros que possam auxiliá-lo a construir seus cursos.

As principais funcionalidades da ferramenta são realizadas por três tipos de usuários distintos: Administrador, Convidado e Conteudista. O primeiro é responsável por toda a gerência do ambiente, tais como serviços de



A representação de OAs no ROSA é feita através de um modelo de dados expressivo, ilustrado na figura 4. *Associações* são expressas através de predicados nos moldes do RDF. Estes relacionam OAs segundo uma semântica particular. A participação de OAs em predicados é capturada no Modelo de Dados através de *declarações* formando triplas

**Figura 4: Modelo de Dados ROSA**

$T(\text{sujeito}, \text{predicado}, \langle \text{objeto} \rangle^2)$ . Nas declarações, *sujeito* e *objetos* são objetos do tipo *Recurso*, sendo este uma unidade básica do Modelo de onde derivam as demais estruturas: *OAs* e *Recursos Complexos*. *OAs* representam os objetos de interesse do Domínio de EAD, enquanto *Recursos Complexos* provêm estruturas de composição para *OAs*. *Recursos complexos* podem ser de três tipos: *não ordenados*, *ordenados* e *declarações reificadas*. *Não ordenados* correspondem aos *recursos* de tipo *Sacola* e *Sacola com restrições de cardinalidade*, encontrados em RDF. *Ordenados* são aqueles compostos por *recursos* em uma dada sequência (*Lista*) e *declarações reificadas* correspondem à reificação do RDF. Apesar de serem contempladas no modelo, tais declarações ainda não foram implementadas na atual versão 1.0 do sistema. Um *OA* é especializado em *OALógico* e *OAFísico*, diferenciando os elementos estruturais (lógicos) dos referentes a arquivos físicos, que contém o conteúdo de um curso, como por exemplo, arquivos do tipo ppt, doc, pdf, texto, imagens, etc.

Esse modelo permite recuperar OAs segundo a semântica expressa nas suas associações tais como: “Que tópicos cobrem a disciplina Database?”; “Que tópicos fundamentam o ensino de SQL?”; “ao ministrar o tópico *Relational Calculus*, que linguagens deverão ser vistas: SQL, QUEL ou QBE?”; “Quais os predicados que *associam* os *cursos* a outros *recursos* quaisquer?”; Que arquivos *criados* por *Fábio* *fundamentam* o tópico *SQL*?”; “Que OAs que são cadastrados como curso?”, etc.

Os metadados e associações de um OA são armazenados no Tamino<sup>3</sup>, um sistema de Banco de Dados XML nativo. O sistema possibilita também realizar a exportação de seus OAs (metadados e declarações) sob forma de um documento XML.

<sup>2</sup>  $\langle \rangle$ : indica lista; no caso significa que pode existir mais de um objeto associado ao sujeito por meio do predicado definido.

<sup>3</sup> <http://www.softwareag.com/tamino>

### 3. Ambiente de Aplicação

Nessa seção será apresentado um panorama geral da ferramenta, com ênfase às suas principais funcionalidades.

A Figura 5 apresenta os mapas conceituais correspondentes aos cursos cadastrados no ROSA. Ao selecionar o mapa conceitual do Curso de Mestrado em Sistemas e Computação (tela à esquerda), é apresentada a lista com opções de disciplinas e seus respectivos tópicos (tela à direita). Propriedades, tipos de relacionamentos com outros OAs e conteúdos físicos (ppt, documentos textos, etc.) podem ser recuperados via navegação ou pesquisa.

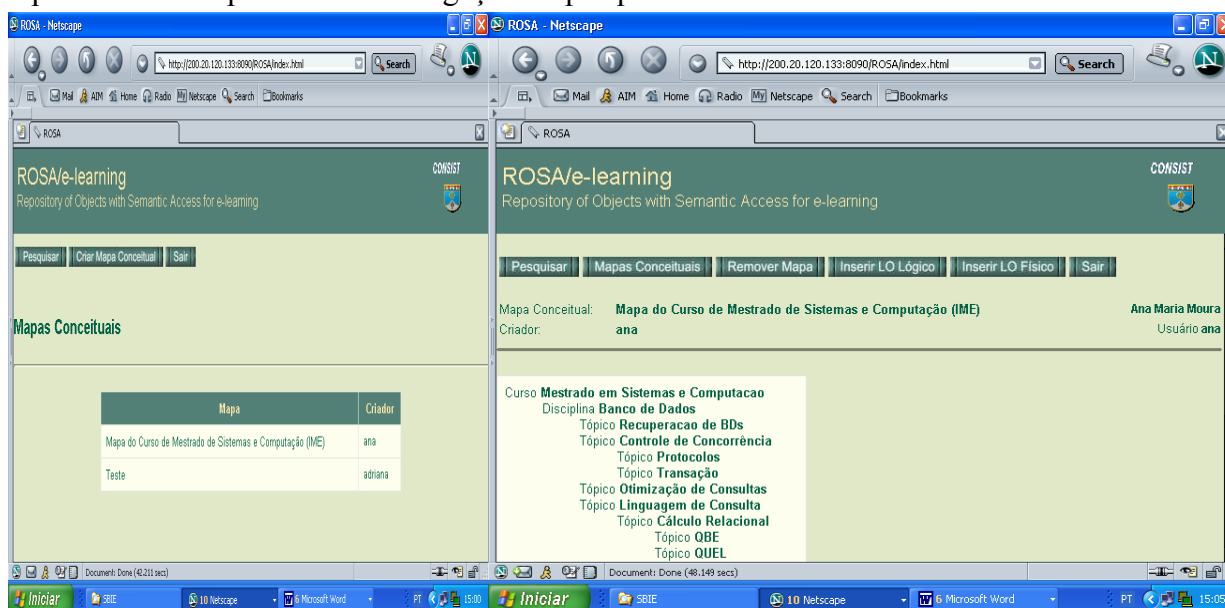


Figura 5: Seleção e Visualização de um mapa conceitual

Consultas aos OAs podem ser realizadas de duas formas: através de navegação pelos relacionamentos em um mapa conceitual; e através de uma interface *à la* QBE. Adicionalmente, consultas podem ser auxiliadas por um tesouro de domínio. Como exemplo, considere que o usuário tenha selecionado a disciplina Banco de Dados numa tela anterior, o conteúdo do OA selecionado é apresentado na Figura 6. De modo a prover uma visão contextual sobre o OA selecionado, todos os relacionamentos de que o OA faz parte, bem como os demais OAs por ele associados, são listados. A partir dos *links* associados a esses OAs, é possível continuar a navegação. Finalmente, as associações podem ser criadas de três tipos diferentes: agregações sem ordem, agregações ordenadas e predicados semânticos. Nas agregações é possível visualizar seus conteúdos clicando na sua própria descrição. Já no predicado semântico, basta clicar no seu próprio nome para a visualização dos objetos associados. O menu à direita da tela (exibir metadados, modificar, remover, exportar e associar OAs lógicos e físicos) permite realizar as operações correspondentes sobre o OA em questão.

O sistema conta, adicionalmente, com uma funcionalidade de exportação que, associada à interface QBE, se transforma em uma boa ferramenta de interface com outros sistemas, sobretudo aqueles de gerência de aprendizado, mais conhecidos como *Learning Management Systems* (LMS).

Figura 6: Conteúdo de um OA realizado a partir de uma pesquisa navegacional

#### 4. Conclusão

ROSA é um sistema concebido para dar suporte à elaboração de cursos voltados a EAD, desenvolvido em parceria entre o IME e a empresa Consist. Sua maior contribuição está voltada à criação e armazenamento de conteúdos didáticos que se relacionam através de predicados, provendo acesso semântico a esses objetos de aprendizagem. Esta característica facilita a localização de recursos, permitindo que usuários da área educacional modelem seus conteúdos instrucionais baseados numa semântica e segundo uma ordem pré-estabelecida. O modelo está sendo atualmente estendido para suportar regras e inferência, bem como seu ambiente ampliado para o uso em ambiente Peer to Peer (P2P).

#### Referências:

- IEEE, Summary of Changes from Working Group Draft v6.1, 6.2, 6.3, & 6.4 to Final Draft\* LOM Standard v1.0, IEEE, julho 2002.
- Fábio Porto, Ana Maria Moura, Gilda H. Campos et al. ROSA: A Data Model and Query Language for e-Learning Objects. I Conferência PGL de Pesquisa em Banco de Dados para E-Learning" (PGLDB'2003), PUC- RJ (2003).
- Fábio Porto, Ana Maria Moura, Fábio Coutinho. ROSA: a Repository of Objects with Semantic Access for e-Learning, 8th International Database Engineering & Applications Symposium - IDEAS '04 – Coimbra - Portugal - julho 2004.
- RDF, Resource Description Framework (RDF), <http://www.w3.org/TR/PR-rdf-syntax/>.

## **XAloc: Uma ferramenta para avaliar algoritmos de alocação de dados**

**Matheus Wildemberg, Melise M. V. Paula,  
Fernanda Baião, Marta Mattoso**

Programa de Engenharia de Sistemas e Computação - COPPE  
Universidade Federal do Rio de Janeiro, Brasil  
{mwild, mel, baiao, marta}@cos.ufrj.br

**Abstract.** *The problem of data allocation directly impacts the execution cost of applications over a distributed database, such as web servers, heterogeneous database integration systems and distributed databases. Allocation algorithms are typically used to define a data distribution among the sites of the network such as to minimize the execution cost of the application. In this work, we propose XAloc, a software tool that provides a fragment allocation scheme for distributed databases. This tool contain three allocation algorithms while comparing the results with the optimal solution found by exhaustive search.*

**Resumo.** *O problema de alocação de dados apresenta influência direta no custo de execução de aplicações sobre uma base de dados distribuída em ambientes como servidores Web, sistemas de integração de bancos de dados heterogêneos e bancos de dados distribuídos. Algoritmos de alocação são utilizados para definir uma distribuição dos dados nos nós da rede de tal forma a minimizar o custo de execução das aplicações. Neste trabalho, é proposta XAloc, uma ferramenta que gera esquemas de alocação de fragmentos para bancos de dados distribuídos. Esta ferramenta contém três algoritmos e permite a comparação dos resultados com a solução ótima encontrada por um algoritmo de busca exaustiva.*

### **1. Introdução**

O problema de alocação é um aspecto crítico em ambientes distribuídos como Web, P2P, clusters e grid. Para reduzir o custo de execução de aplicações sobre bases de dados distribuídas, é necessário que técnicas eficientes de alocação de dados sejam adotadas. Em Sistemas de Bancos de Dados Distribuídos (SBDD), a alocação de fragmentos influencia diretamente o custo de acesso ao SBDD. Em geral, em um projeto de distribuição da base de dados, a fase de alocação é realizada após a fase de fragmentação, a qual determina a distribuição dos dados em fragmentos. Um dos principais objetivos da fase de alocação é atingir a proximidade entre os fragmentos e os nós que os utilizam, minimizando o custo de comunicação entre os nós da rede durante a execução das consultas da aplicação.

O resultado da fase de alocação é a definição de um esquema de alocação para os fragmentos da base de dados. O esquema de alocação determina, para cada fragmento, o número de réplicas e a localização de cada réplica pelos nós da rede.

Em [Sacca and Wiederhold, 1985], os autores afirmam que o problema de

encontrar uma alocação de fragmentos ótima em um SBDD é NP-Difícil. Segundo [Huang and Chen, 2001], dado um problema com  $n$  fragmentos e  $m$  nós,  $(2^m - 1)^n$  é o número de combinações possíveis para este problema. Apesar da existência de um número finito de soluções, a avaliação de cada solução para a escolha da melhor é muito custosa. Assim, é inviável solucionar esse tipo de problema com um algoritmo exato em função da sua complexidade.

A ferramenta apresentada, XAloc, propõe um esquema de alocação de fragmentos de uma base de dados pelos nós de uma rede distribuída. Através de uma interface gráfica bastante simples, ilustrada pela Figura 1, o usuário (projetista da distribuição) da ferramenta XAloc fornece os parâmetros de entrada para o problema de alocação de dados, que são: o conjunto de fragmentos, o conjunto de nós, o conjunto de transações mais frequentes e a frequência de execução das transações e de acesso aos fragmentos.

Para a definição do esquema de alocação, o usuário tem a opção de escolher dentre quatro algoritmos distintos implementados em XAloc: *Huang* [Huang and Chen, 2001], *Aloc* [Wildemberg et al., 2003], *GRADA* [Wildemberg 2004] além de um algoritmo de busca exaustiva que encontra a solução ótima.

Para a avaliação de cada solução considerada no algoritmo escolhido, é utilizada uma função de custo. Na XAloc, esta função de custo pode ser escolhida pelo usuário dentre três possíveis.

A interface gráfica da ferramenta XAloc, intitulada "Algoritmo HuCh (1\_444.bkp)", apresenta uma aba "Principal" e uma aba "Ferramentas". Na aba "Principal", há campos para definir os seguintes parâmetros:

- Quantidade de Nós: 4
- Total de Transações: 4
- Quantidade de Fragmentos: 4
- Custo de transferir um pacote de dados: 0,032
- Tamanho de um pacote de dados: 6250
- Custo de construção do circuito virtual: 65,5676

Abaixo desses campos, há um botão "Dimensiona Matrices". Abaixo dele, há uma seção com seis abas: "Matriz de Frequência", "Matriz de Custo de Comunicação", "Vetor de Tamanho de Fragmento", "Matriz de Acesso", "Matriz de Atualização" e "Matriz de Seletividade". A aba "Matriz de Acesso" está selecionada, mostrando a seguinte matriz:

	F1	F2	F3	F4
T1	1	0	0	3
T2	0	1	0	0
T3	2	0	3	0
T4	0	4	0	5

Na base da interface, há um ícone de ajuda (?) e um botão "Gerar Matriz de Alocação".

**Figura 1: Dados de entrada**

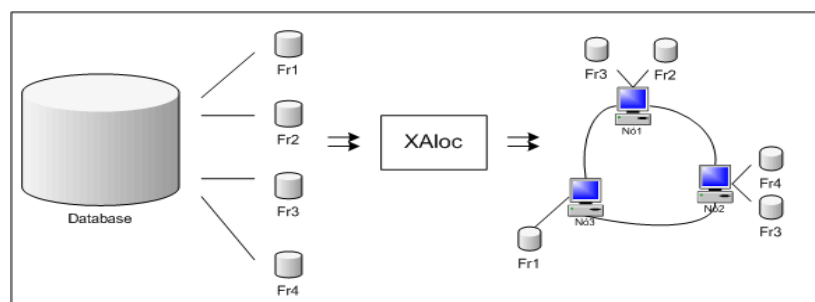
O restante desse artigo descreve na Seção 2 as principais características da ferramenta, assim como o objetivo, os algoritmos e funções de custo implementadas e as funcionalidades que facilitam a validação dos resultados. Para finalizar o artigo, a Seção 3 faz a conclusão do trabalho.

## 2. Componentes da ferramenta XAloc

O objetivo desta ferramenta é permitir a comparação entre os custos dos esquemas de alocação propostos pelos algoritmos implementados. São quatro os algoritmos implementados: *Huang* [Huang and Chen, 2001], Solução Ótima, *Aloc* e *GRADA*

[Wildemberg 2004]. *Aloc* e *GRADA* propõem variações do algoritmo *Huang* buscando encontrar um esquema de alocação onde o custo de execução das aplicações seja menor. A Solução Ótima foi implementada para validar quão próximos os resultados encontrados pelos algoritmos escolhidos se encontram da solução ótima.

A Figura 2 representa a entrada e saída da XAloc. Sobre as informações dos fragmentos são executados os algoritmos de alocação utilizando uma função de custo para encontrar a melhor distribuição dos fragmentos nos nós da rede.



**Figura 2: Características da XAloc**

## 2.1. Preparação dos Dados

Os algoritmos consideram um conjunto de matrizes de entrada [Wildemberg 2004]. A matriz de acesso representa a frequência leitura que uma transação  $t$  realiza em um fragmento  $f$  a cada vez que é executada. A matriz de atualização representa a frequência que uma transação  $t$  atualiza um fragmento  $f$  a cada vez que é executada. A matriz de seletividade representa a porção de dados de um fragmento  $f$  que é acessada por uma transação  $t$ . A matriz de frequência representa a frequência de execução de cada transação em cada nó. A matriz de custo de comunicação representa custo de transferir uma unidade de dado entre dois nós da rede. Finalmente o vetor de tamanho dos fragmentos que representa o tamanho dos fragmentos.

As dimensões destas matrizes variam de acordo com o número de nós, fragmentos e transações também considerados como dados de entrada dos algoritmos.

## 2.1. Algoritmos

A Seções 2.1.1, 2.1.2, 2.1.3 e 2.1.4 descrevem as principais características dos algoritmos *Huang*, *Aloc*, *GRADA* e Solução Ótima, respectivamente, mostrando os pontos que diferem os algoritmos implementados.

### 2.1.1. Algoritmo Huang

O algoritmo *Huang* [Huang and Chen, 2001] propõe uma alocação gulosa dos fragmentos nos nós da rede. Em um passo inicial os fragmentos são distribuídos de acordo com as transações de leitura. Esta alocação inicial tem o objetivo de beneficiar as transações de leitura, uma vez que a alocação definida neste passo apresenta o menor custo de execução das transações de leitura.

Em um segundo passo, o algoritmo reduz o número de cópias considerando as transações de atualização. Como dito anteriormente, a solução encontrada pelo primeiro passo do algoritmo é a solução ótima se considerar as transações de leitura, porém esta

solução apresenta um custo muito alto para a execução das transações de atualização em virtude do custo de manter a consistência das réplicas dos fragmentos.

Finalmente, o terceiro passo do algoritmo *Huang* faz a alocação dos fragmentos que não são acessados por nenhuma transação de leitura, mas são acessados por alguma transação de atualização. Assim, cada fragmento que esteja nestas condições é alocado no nó onde o custo de execução das transações de atualização seja mínimo.

### 2.1.2. Algoritmo *Aloc*

Como o algoritmo *Huang*, o algoritmo *Aloc* [Wildemberg et al., 2003] propõe uma alocação gulosa dos fragmentos, porém *Aloc* propõe uma alteração no primeiro e terceiro passos do algoritmo *Huang*.

No primeiro passo o algoritmo *Aloc* tenta beneficiar não somente as transações de leitura como também as transações de atualização. Assim, os fragmentos são alocados em nós que executem qualquer transação, leitura ou atualização, sobre o fragmento. O objetivo desta alteração é aumentar o espaço de soluções do algoritmo considerando soluções viáveis com um custo de alocação menor.

O terceiro passo do algoritmo *Huang* não é necessário, pois com a alteração do primeiro passo, os fragmentos não lidos são alocados de acordo com as transações de atualização no primeiro passo do *Aloc*. Porém, foi identificada uma falha no algoritmo *Huang* em relação aos fragmentos que não foram referenciados por nenhuma das transações freqüentes consideradas no projeto de distribuição. Assim, um fragmento que não é acessado por nenhuma transação não é alocado pelo algoritmo *Huang*. Para estes fragmentos, o algoritmo *Aloc* propõe uma alocação circular.

Com a alteração no terceiro passo, o algoritmo *Aloc* garante a alocação completa dos fragmentos, desconsiderada pelo algoritmo *Huang*. Apesar destas alterações o algoritmo *Aloc* mantém a ordem de complexidade do algoritmo *Huang* definida como  $O(nm^2q)$ , onde  $n$  é o número de fragmentos distintos,  $m$  é o número de nós e  $q$  é o número de transações.

### 2.1.3. Algoritmo *GRADA*

*GRADA* propõe a utilização da meta-heurística GRASP [Feo and Resende, 1995] para resolução do problema de alocação de fragmentos em SBDD. Como em *Aloc*, *GRADA* também é dividido em três passos. Os três passos do algoritmo *GRADA* foram alterados a fim de incorporar a meta-heurística. No primeiro passo, o algoritmo *GRADA* faz uma alocação completa dos fragmentos, ou seja, cada fragmento é alocado em todos os nós da rede sem considerar transações de leitura ou atualização. No segundo passo, *GRADA* propõe a utilização de uma lista de réplicas candidatas à remoção. Esta lista possui as  $k$  réplicas com maior fator benefício/custo de retirada da réplica [Wildemberg 2004]. Desta lista, uma das réplicas é selecionada aleatoriamente para ser removida e, a cada remoção, esta lista é refeita. Esta alteração define a inclusão de um fator aleatório ao método de solução do *GRADA*, segundo a estratégia proposta pela meta-heurística GRASP. O terceiro passo do algoritmo *GRADA* propõe uma técnica de busca local que visa encontrar uma solução melhor que a solução apresentada no segundo passo.



#### 2.1.4. Algoritmo de Solução Ótima

A solução considerada ótima é definida como a solução viável com menor custo de alocação. A solução ótima é encontrada por um algoritmo de busca exaustiva. Pelo fato do algoritmo considerar todas as soluções viáveis para o problema, torna-se impraticável sua utilização para problemas que envolvem um número alto de soluções devido ao tempo de processamento. O objetivo de sua utilização é validar as soluções encontradas pelos algoritmos em problemas com menor número de soluções.

#### 2.2. Função de custo

A ferramenta permite a comparação dos algoritmos de acordo com três tipos de função de custo. A função de custo denominada *HuCh* implementa a função de custo proposta em [Huang and Chen, 2001]. Nesta função de custo, a atualização de um fragmento envolve o tráfego deste fragmento de um nó da rede para todos os outros nós que alocam réplicas deste fragmento (*transporte de dados* [Ozsu and Valduriez, 1999]). Nesta função, é desconsiderado o custo de envio de uma mensagem do nó que executa a transação de atualização para o nó de onde os dados do fragmento serão transferidos.

A segunda função de custo, denominada *Tráfego duplo de  $F_i$* , faz uma alteração da função de custo proposta por [Huang and Chen, 2001]. Nesta função de custo, o nó que atualiza um determinado fragmento recupera seus dados, realiza sua atualização e transfere de volta ao nó de origem. Assim o custo de atualização é a soma do custo de recuperação dos dados do fragmento de cada nó que o aloca para o nó que executa a atualização e o custo de retornar estes dados atualizados para os nós que alocam este fragmento. Como a função de custo anterior, esta função de custo representa a execução de *transporte de dados*.

Finalmente, a terceira função de custo implementa a atualização de fragmentos definida como *transporte de função* [Ozsu and Valduriez, 1999]. Nesta abordagem, o custo de atualização é definido como o envio de uma mensagem do nó que solicita a transação de atualização para cada nó que aloca uma réplica do fragmento, para que todos executem a atualização. Esta abordagem substitui o custo de transferência de dados pelo custo de processamento local.

#### 2.3. Características gerais da XAloc

Esta seção descreve algumas propriedades da entrada e saída de Xaloc. A ferramenta foi desenvolvida em Delphi 5 e pode ser executada em ambiente Windows.

Para facilitar o preenchimento dos parâmetros de entrada definidos na Seção 2.1, a ferramenta apresenta algumas propriedades. As matrizes de entrada de um determinado problema podem ser salvas em um arquivo. Estes dados podem ser recuperados para re-execuções futuras de algum algoritmo. Estas matrizes podem ser geradas de forma aleatória de acordo com o número de nós, transações e fragmentos. Atualmente, o custo de comunicação entre os nós da rede varia de 0,16 a 2,88 ms.

Além destas propriedades, XAloc apresenta graficamente as matrizes de alocação resultantes de cada algoritmo. Estas matrizes podem ser salvas em arquivos XML e a partir destes resultados é possível alterar, através de uma interface gráfica, alguma alocação de fragmento recalculando o custo da nova alocação. Esta propriedade facilita a análise dos resultados alcançados pelos algoritmos.

### 3. Conclusão

Neste trabalho foi apresentada a ferramenta XAloc que viabiliza a validação dos algoritmos *Aloc* e *GRADA*, propostos em [Wildemberg 2004], comparado aos resultados alcançados pelo algoritmo proposto por [Huang and Chen, 2001]. Além disso, com a utilização da solução ótima, é possível mostrar que os resultados alcançados pelos algoritmos propostos não se distanciam da solução ótima.

O objetivo de *Aloc* e *GRADA* é apresentar uma solução para o problema de alocação de fragmentos em SBDD através de algoritmos simples. Baseados em um trabalho previamente definido na literatura [Huang and Chen, 2001] que atende a estes quesitos, foi possível encontrar uma solução próxima de ótima com uma pequena variação do seu método de solução, mantendo a complexidade do algoritmo na solução heurística. O algoritmo *Aloc* conseguiu reduzir o custo da matriz de alocação para todos os casos onde exista ao menos um fragmento onde sua melhor alocação deva ser em um nó que não execute nenhuma transação de leitura e somente atualização deste fragmento. O algoritmo *GRADA* apresentou resultados satisfatórios em situações com maior número de nós na rede e/ou uma quantidade menor de transações de atualização, nos demais casos, os resultados são semelhantes aos alcançados pelo algoritmo *Aloc*.

Foram realizados experimentos com o esquema definido no *benchmark* TPC-C para representar o comportamento dos algoritmos em situações reais. Segundo a especificação do TPC-C foram modelados os dados de entrada como definido nos algoritmos. Nestes experimentos, os algoritmos *Aloc* e *GRADA* conseguiram reduzir o custo da matriz de alocação em até 12,50%. Outra contribuição deste trabalho é a possibilidade de comparação dos algoritmos utilizando três funções de custo com características diferentes.

A ferramenta XAloc pode ser estendida também para problemas de alocação de dados em ambientes com distribuição e replicação, como servidores web, clusters, grids e sistemas de integração de banco de dados heterogêneos. Isto é possível por tratar os fragmentos de forma genérica, sem considerar o modelo de dados. Para isso, seria necessário modelar o problema segundo os dados de entrada dos algoritmos, e adotar uma função de custo adequada ao problema.

Outras funções de custo da literatura consideram custo de processamento local, que foi desprezado nas funções de custo deste trabalho. Em trabalhos futuros pretendemos realizar experimentos considerando esses custos. Ainda, pretende-se tratar alocação dinâmica, ou seja, a re-alocação de fragmentos em tempo de execução.

### Referências

- Feo, T., Resende, M., 1995 *Greedy randomized adaptive search procedures*. Journal of Global Optimization. 6: 109-133.
- Huang, Y. and Chen, J. (2001). *Fragment allocation in distributed database design*. Information Science and Engineering, 13(3):491-506.
- Ozsu, M. T. and Valduriez, P. (1999). *Principles of distributed database systems*. Prentice Hall
- Sacca, D. e Wiederhold, G., 1985. *Database Partitioning in a Cluster of processors*. ACM Transactions on Database System, 10(1): 29--56.
- Wildemberg, M., Paula, M., Baião, F., and Mattoso, M. (2003). Alocação de dados em bancos de dados distribuídos. *XVIII Simpósio Brasileiro de Banco de Dados*, pp. 215 -228.
- Wildemberg, M., (2004). Técnicas para alocação de fragmentos em projeto de bancos de dados distribuídos. *Dissertação de Mestrado, COPPE-Sistemas, UFRJ*.

## Um Agente de Software para Criação de Índices no PostgreSQL

Marcos Antonio Vaz Salles<sup>1</sup>, Sérgio Lifschitz<sup>1</sup>

<sup>1</sup>Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)  
Rua Marquês de São Vicente, 225, Gávea – 22453-900 Rio de Janeiro, RJ

mvsalles@inf.puc-rio.br, sergio@inf.puc-rio.br

**Abstract.** *This paper briefly describes a prototype developed as part of a Masters thesis focused on autonomic index creation for database systems. The system is composed of a software agent that collects SQL commands submitted to the DBMS, analyzes which indexes are appropriate for those commands and automatically creates them. The analysis process uses server extensions that enable the creation of hypothetical indexes. The agent's implementation was done in C++ and integrated in the open source DBMS PostgreSQL. The server extensions made in PostgreSQL to simulate hypothetical index configurations were coded in C.*

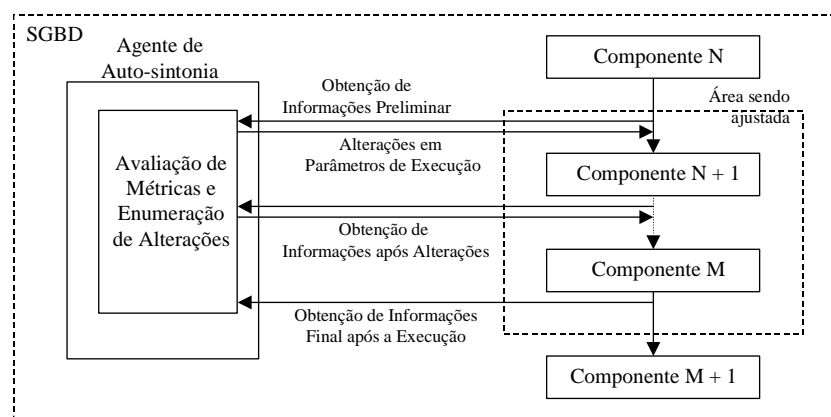
**Resumo.** *Este artigo descreve brevemente um protótipo desenvolvido como parte de uma dissertação de mestrado focada na criação autônoma de índices em sistemas de bancos de dados. O sistema é composto de um agente de software que coleta comandos SQL processados pelo SGBD, analisa quais índices seriam adequados para estes comandos e os cria automaticamente. O processo de análise se utiliza de extensões codificadas no servidor para a criação de índices hipotéticos. A implementação do agente foi realizada em C++ e integrada no SGBD de código fonte aberto PostgreSQL. As extensões feitas no PostgreSQL para simulação de configurações hipotéticas de índices foram escritas em C.*

### 1. Introdução

A sintonia de bancos de dados (*database tuning*) é a atividade de tornar a execução de uma aplicação de bancos de dados mais rápida [Shasha and Bonnet, 2003]. Por mais rápida podemos entender que a aplicação terá maior vazão (*throughput*) em termos das transações que executa ou que algumas de suas transações terão menores tempos de resposta.

Uma atividade comumente realizada por administradores de sistemas de bancos de dados para acelerar o desempenho de consultas submetidas a um SGBD relacional é a seleção de índices sobre as tabelas presentes na base de dados. A automatização da atividade de criação de índices envolve diversas considerações, como, por exemplo, a obtenção da carga de trabalho adequada, a escolha de quais tabelas e colunas devem ser indexadas, e a determinação do momento adequado para criar ou remover índices.

Neste artigo comentamos a implementação de uma arquitetura de agentes para a auto-sintonia de índices em um sistema de bancos de dados. A arquitetura propõe o uso de um agente de *software* embutido no SGBD que obtém os comandos processados pelo SGBD e decide quando criar índices adequados para estes comandos. Esta arquitetura faz parte de uma dissertação de mestrado recentemente aceita no Departamento de Informática da PUC-Rio [Salles, 2004].



**Figura 1: Modelo local para auto-sintonia usando agentes**

O agente, denominado *Agente de Benefícios*, foi integrado com o SGBD de código fonte aberto PostgreSQL [Pos, 2004]. Foi necessário realizar extensões sobre este SGBD para permitir que o agente simulasse configurações hipotéticas de índices interagindo com o otimizador do sistema. Este tipo de técnica foi aplicada em outros trabalhos da literatura [Chaudhuri and Narasayya, 1997, Lohman et al., 2000], porém sempre em sistemas comerciais.

O restante deste artigo está organizado conforme descrito a seguir. A seção 2 descreve a arquitetura de integração do agente utilizado com o SGBD e discute qual a metodologia empregada para a tomada de decisões de auto-sintonia. Na seção 3, abordamos as demonstrações que podem ser realizadas do protótipo desenvolvido para a pesquisa. Por fim, na seção 4, apresentamos algumas conclusões e extensões possíveis do trabalho de implementação.

## 2. Arquitetura do Agente de Benefícios

Podemos classificar o foco das iniciativas de auto-sintonia de sistemas de bancos de dados em local ou global [Lifschitz et al., 2004]. Quando tratamos de uma iniciativa específica de sintonia, como é o nosso caso com a sintonia de índices, estamos preocupados eminentemente com o modelo local de auto-sintonia a ser empregado. A Figura 1 mostra este modelo para o *Agente de Benefícios*.

No modelo local, o agente interage com os componentes do SGBD referentes à sua área de ajuste. Esta interação é regida por um processo de auto-sintonia com as seguintes etapas:

- **Obtenção de Informações:** o agente se utiliza de sensores para receber informações e métricas dos componentes do SGBD.
- **Avaliação de Métricas:** a partir das informações coletadas, o agente estabelece crenças que dão suporte aos algoritmos utilizados para decidir se alguma ação de sintonia deve ser realizada.
- **Enumeração de Alterações Possíveis:** o agente aplica algoritmos ou heurísticas para enumerar alternativas de ajustes que podem levar a um melhor desempenho do sistema. Durante esta enumeração, o agente pode se utilizar de efetadores para simular cenários utilizando mecanismos previamente implementados nos componentes do SGBD.
- **Realização de Alterações:** os ajustes escolhidos pelos algoritmos ou heurísticas empregados pelo agente são aplicados aos componentes do sistema através do uso de efetadores.

No caso do *Agente de Benefícios*, a etapa de *Obtenção de Informações* é realizada através da coleta dos comandos SQL processados pelo otimizador. Em seguida, ocorre a *Avaliação de Métricas* com a atualização de crenças necessárias para a aplicação de heurísticas para seleção de índices. Em nosso trabalho, utilizamos uma heurística de [Lohman et al., 2000] para escolha de índices candidatos e propusemos uma nova heurística para determinação de quais índices deveriam ser materializados. Informações detalhadas sobre as heurísticas utilizadas para a tomada de decisões do agente estão fora do escopo deste artigo.

O agente aplica as heurísticas durante a etapa de *Enumeração de Alternativas Possíveis*. Neste momento, são criados índices hipotéticos<sup>1</sup> no sistema e o agente utiliza o otimizador para obter planos e custos estimados para a execução de comandos com estas configurações físicas simuladas. Por fim, é conduzida a *Realização de Alterações*, que é implementada pela criação ou remoção de índices reais do sistema de bancos de dados.

O processo de auto-sintonia apresentado se baseia em um *ciclo de controle de realimentação* para refinar progressivamente as decisões de sintonia tomadas com o conhecimento das métricas locais obtidas a partir dos componentes do SGBD. O uso deste tipo de abordagem para processos de auto-sintonia é comum na literatura, conforme é detalhado em [Lifschitz et al., 2004].

### 3. Características da Demonstração

O protótipo que foi desenvolvido para apoiar nossa pesquisa permite que realizemos duas demonstrações de funcionalidade. Na primeira, mostramos como o servidor PostgreSQL foi estendido para permitir a simulação de índices hipotéticos. Este mecanismo foi implementado de forma a permitir que a simulação seja executada tanto de forma automática como por um DBA que interaja com o sistema. As extensões do servidor foram implementadas em C.

Na segunda demonstração, ilustramos como o agente adapta os índices do sistema aos comandos SQL que são submetidos. Fazemos a execução de uma carga transacional com múltiplos usuários e verificamos como o uso do agente influi na vazão do sistema. O agente foi escrito em C++ e integrado com o SGBD PostgreSQL.

#### 3.1. Demonstração das Extensões do Servidor

Em nossa implementação, estendemos a linguagem de definição de dados do SGBD PostgreSQL para conter os seguintes novos comandos:

1. *create hypothetical index*;
2. *drop hypothetical index*;
3. *explain hypothetical*;

Os primeiros dois comandos têm como finalidade possibilitar o registro e a remoção de índices hipotéticos. A exemplo de [Lohman et al., 2000], estas definições são armazenadas no próprio catálogo, que foi estendido para permitir diferenciar índices hipotéticos de índices reais. Já o terceiro comando permite que sejam obtidos planos de execução e custos de consultas levando em consideração os índices hipotéticos definidos.

Para ilustrar o comportamento dos novos comandos adicionados ao sistema, apresentamos um exemplo. Suponha que estejamos escrevendo uma consulta sobre uma base de dados simples de vendas. Podemos ver o plano que o SGBD escolheria para a consulta utilizando o comando *explain*:

---

<sup>1</sup>Índices hipotéticos são utilizados para avaliar a qualidade de configurações simuladas de índices. Estes tipos de índices existem apenas no catálogo do sistema e não possuem extensão física.

```
simple=# explain
simple=# select prodNum, data, sum(valor) as total
simple=# from   venda
simple=# where  valor > 1500000 and
simple=#        data between '20040101' and '20040131'
simple=# group by prodNum, data;
                        QUERY PLAN
```

```
-----
HashAggregate  (cost=25388.79..25388.83 rows=12 width=21)
->  Seq Scan on venda  (cost=0.00..25367.00 rows=2906
                        width=21)
    Filter: ((valor > 1500000::numeric) AND
              (data >= '2004-01-01'::date) AND
              (data <= '2004-01-31'::date))
(3 rows)
```

No plano, podemos perceber que o SGBD fará uma varredura seqüencial sobre a tabela de vendas e aplicará os predicados sobre as colunas de valor e de data. A quantidade de linhas esperada como resultado é de 2906 e o custo de processamento da varredura será de 25367.00. Após a varredura, um operador de agregação será aplicado para processar a cláusula *group by* do comando. O custo total esperado é de 25388.83.

Gostaríamos de verificar a utilidade de um determinado índice sobre a tabela de vendas para nossa consulta. Podemos criar um índice hipotético conforme mostrado abaixo:

```
simple=# create hypothetical index hi_venda_valor_data
simple=# on venda (valor, data);
```

Reparar que, ao fazermos isto, o índice não é materializado e, assim, o processamento de outras transações no sistema continua praticamente inafetado. Podemos, agora, verificar que plano seria escolhido pelo SGBD se este índice fosse materializado na base. Utilizamos o comando *explain hypothetical*:

```
simple=# explain hypothetical
simple=# select prodNum, data, sum(valor) as total
simple=# from   venda
simple=# where  valor > 1500000 and
simple=#        data between '20040101' and '20040131'
simple=# group by prodNum, data;
                        QUERY PLAN
```

```
-----
HashAggregate  (cost=10514.63..10514.66 rows=12 width=21)
->  Index Scan using hi_venda_valor_data on venda
    (cost=0.00..10492.83 rows= 2906 width=21)
    Index Cond: ((valor > 1500000::numeric) AND
                  (data >= '2004-01-01'::date) AND
                  (data <= '2004-01-31'::date))
(3 rows)
```

Podemos perceber que, caso existisse, o índice sobre as colunas de valor e data seria escolhido para o processamento da consulta. A quantidade esperada de linhas a

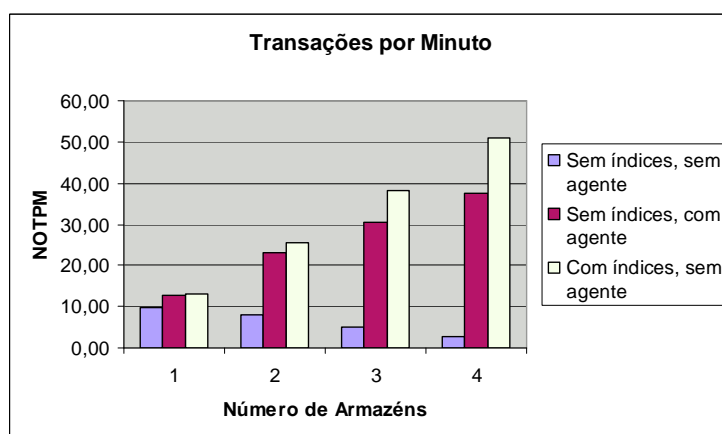


Figura 2: Vazão observada para o agente em testes de 90 min

ser retornada pela varredura indexada seria de 2906 a um custo de 10492.83. Após a varredura indexada, seria aplicado o operador de agregação e o plano como um todo teria um custo de 10514.66. Assim, o novo índice traria benefícios à consulta que pretendemos executar.

### 3.2. Demonstração do Agente

Os testes de desempenho da implementação do agente foram realizados com o *toolkit* OSDL DBT-2 [OSDL-DBT2, 2004]. Este simula a execução de uma carga transacional, inspirada no *benchmark* TPC-C, com múltiplos usuários no SGBD. Ocorre a simulação das operações de um varejista que trabalha com diversos armazéns para estocagem de seus produtos.

O *toolkit* já vem com um conjunto de índices sugeridos para o melhor processamento da carga de trabalho. Alguns destes índices são criados como consequência da criação de chaves primárias. Para avaliarmos a contribuição trazida pelo nosso agente à vazão observada nos testes, estabelecemos três cenários:

1. Execução da carga de trabalho em uma base sem índices e com o agente desligado.
2. Execução da carga de trabalho em uma base sem índices e com o agente ligado.
3. Execução da carga de trabalho em uma base com os índices sugeridos pelo *toolkit* e com o agente desligado.

Existem dois parâmetros que modificam as características de cada teste: a quantidade de armazéns configurados para o *toolkit* e o tempo de duração. Quanto mais armazéns, maior é a carga efetivamente submetida ao SGBD, uma vez que temos bases de dados maiores e também mais terminais acessando o sistema de forma concorrente.

O tempo de duração afeta a vazão observada uma vez que o agente precisa de um período de aprendizado para determinar os índices adequados para a carga de trabalho. Na Figura 2, mostramos as vazões observadas em um teste com duração de 90 minutos para os três cenários descritos anteriormente.

Temos uma vazão decrescente para a execução da carga de trabalho em uma base sem índices e sem o agente ligado. Neste cenário, os tempos de resposta aumentam, uma vez que processamos as consultas através de varreduras sequenciais em bases de dados de crescente tamanho. Já no terceiro cenário, em que utilizamos os índices do *toolkit*, temos o comportamento inverso. A vazão do sistema cresce à medida que os tamanhos da base de dados aumentam. Isto se deve ao fato de os tempos de resposta permanecerem praticamente constantes e de o sistema possuir mais terminais simultâneos submetendo transações quando configuramos mais armazéns para o *toolkit*.

A vazão observada para o cenário com o agente se situa entre as vazões dos outros dois cenários. O agente passa por um período de aprendizado, em que ocorre a criação de diversos índices que podem trazer benefícios para os comandos SQL da carga de trabalho. Uma vez que os índices indicados são criados, o agente permanece monitorando o sistema para verificar se alguma nova intervenção é necessária. Em [Salles, 2004], diversos outros tipos de testes são explorados com a implementação realizada.

Na demonstração, pretendemos executar o agente com a carga de trabalho do *toolkit* OSDL DBT-2 durante um curto período. Observaremos a vazão e os índices que são escolhidos pelo agente para o sistema.

#### 4. Conclusões

Implementamos um protótipo para validar uma das arquiteturas para auto-sintonia de índices propostas em [Salles, 2004]. Este protótipo integra um agente de *software* ao SGBD PostgreSQL. Possui como característica, ainda, um conjunto de extensões realizadas sobre o servidor de bancos de dados que permitem a simulação de configurações hipotéticas de índices.

O protótipo pode ser estendido em alguns pontos. Seria interessante permitir que o agente realizasse a remoção de índices previamente existentes na base de dados de forma automática. Outro ponto interessante é a investigação de diferentes heurísticas de decisão para o agente.

#### Referências

- Chaudhuri, S. and Narasayya, V. (1997). An efficient, cost-driven index selection tool for microsoft sqlserver. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 146–155.
- Lifschitz, S., Milanés, A. Y., and Salles, M. A. V. (2004). Estado da arte em auto-sintonia de sistemas de bancos de dados relacionais. Technical report, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio).
- Lohman, G., Valentin, G., Zilio, D., Zuliani, M., and Skelley, A. (2000). DB2 advisor: An optimizer smart enough to recommend its own indexes. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 101–110.
- OSDL-DBT2 (2004). Open source development labs database test 2 (osdl-dbt-2). [http://www.osdl.org/lab\\_activities/kernel\\_testing/osdl\\_database\\_test\\_suite/osdl\\_dbt-2/](http://www.osdl.org/lab_activities/kernel_testing/osdl_database_test_suite/osdl_dbt-2/).
- Pos (2004). PostgreSQL. <http://www.postgresql.org>.
- Salles, M. A. V. (2004). Criação autônoma de Índices em bancos de dados. Master's thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). orientada por Sérgio Lifschitz e aceita pela Comissão Julgadora em 15/07/2004.
- Shasha, D. and Bonnet, P. (2003). *Database Tuning: Principles, Experiments and Troubleshooting Techniques*. Morgan Kaufmann.



## PDM: Palm Database Manager

**Dorotéa Karine Dias Pitombeira<sup>1</sup>, Ricardo Wagner Cavalcante Brito<sup>1</sup>, Wendel Bezerra Silva<sup>2</sup>, Danielle Christina Costa Amorim<sup>2</sup>, Magno Prudêncio de Almeida Filho<sup>3</sup>, Marília Soares Mendes<sup>2</sup>, Thiago Leite e Carvalho<sup>2</sup>, Wandré Araújo de Oliveira<sup>2</sup>, Ângelo Brayner<sup>1</sup>, Nabor das Chagas Mendonça<sup>1</sup>**

<sup>1</sup>Mestrado em Informática Aplicada – Universidade de Fortaleza (UNIFOR)  
Caixa Postal 1258 – 60.811-341 – Fortaleza – CE – Brasil

<sup>2</sup>Curso de Informática – Universidade de Fortaleza (UNIFOR)  
Caixa Postal 1258 – 60.811-341 – Fortaleza – CE – Brasil

<sup>3</sup>Curso de Engenharia de Telecomunicações – Universidade de Fortaleza (UNIFOR)  
Caixa Postal 1258 – 60.811-341 – Fortaleza – CE – Brasil  
doroteakarine@hotmail.com, ricardowcb@yahoo.com.br

**Resumo.** *A popularização dos dispositivos móveis tem estimulado o desenvolvimento de novas aplicações para esse segmento. Porém, estas aplicações não oferecem funcionalidades capazes de controlar estruturas de bancos de dados, como acontece nos bancos de dados relacionais do ambiente desktop. Para atender essa demanda, foi desenvolvida a arquitetura PDM. Ela tem o propósito de fornecer ao usuário as principais características de um sistema gerenciador de banco de dados para dispositivos Palm. Além disso, essa estrutura oferece transparência de acesso a banco de dados remoto através de uma rede sem fio. Para dar suporte a estas funcionalidades, foram desenvolvidas a ferramenta PDM e as APIs DBLib, ConnLib e DBConnLib.*

### 1. Introdução

Diante da evolução computacional, as pessoas têm, atualmente, a possibilidade de acessar informações em qualquer lugar e a qualquer momento. Com um crescimento estimado entre 30% e 50% por ano, o público de usuários de aplicações executadas em *handheld* vem se tornando cada vez mais importante e visado pelas empresas. Dentro de alguns anos, a integração entre as estruturas sem fio e com fio permitirá a conectividade entre os mais diversos dispositivos. Dessa forma, muitas aplicações estão sendo desenvolvidas para suprir essas exigências.

Em dispositivos móveis com ambiente operacional Palm OS, os dados são armazenados em arquivos e manipulados em forma de registros. Uma das principais desvantagens dessa estrutura é a dependência na organização física entre os programas e os dados, dificultando o acesso e o gerenciamento desses dados.

Com o objetivo de facilitar este acesso e visando atender as demandas de mobilidade, foram definidas APIs que permitem criar bancos de dados, tabelas, relacionamentos entre tabelas e restrições de integridade em dispositivos móveis, possibilitando o acesso aos dados armazenados nestas estruturas de forma local ou remota.

Este artigo está organizado nas seguintes seções: a seção 2 descreve uma visão geral da forma de armazenamento de dados em dispositivos Palm. A seção 3 apresenta a arquitetura proposta, descrevendo os componentes que formam essa estrutura. Também nesta seção são abordadas as principais características e funcionalidades da ferramenta PDM e das APIs DBLib, ConnLib e DBConnLib. Ainda nesta seção são mencionados alguns aplicativos que realizam o armazenamento de dados para ambiente Palm OS. A seção 4 apresenta as conclusões do artigo bem como as idéias que serão implementadas em trabalhos futuros. Por fim, a seção 5 descreve as fontes bibliográficas utilizadas.

## 2. Armazenamento de Dados em Dispositivos Palm

O armazenamento de dados em dispositivos Palm acontece em forma de registros gravados em memória e não possui a idéia de arquivos com extensões. Todas as informações armazenadas no ambiente operacional Palm OS são consideradas banco de dados. Existem os bancos de dados chamados *Palm Database* (PDB) e *Palm Resource* (PRC). Aqueles do tipo PDB armazenam os registros com dados que são utilizados por outras aplicações. Os bancos do tipo PRC são os programas e têm a finalidade de armazenar código fonte e recursos de *interface* para usuário. De acordo com o formato específico para cada tipo de dado armazenado em memória, o sistema operacional Palm OS tem condições de gerenciar como esses dados serão disponibilizados para o usuário e para as aplicações existentes no dispositivo.

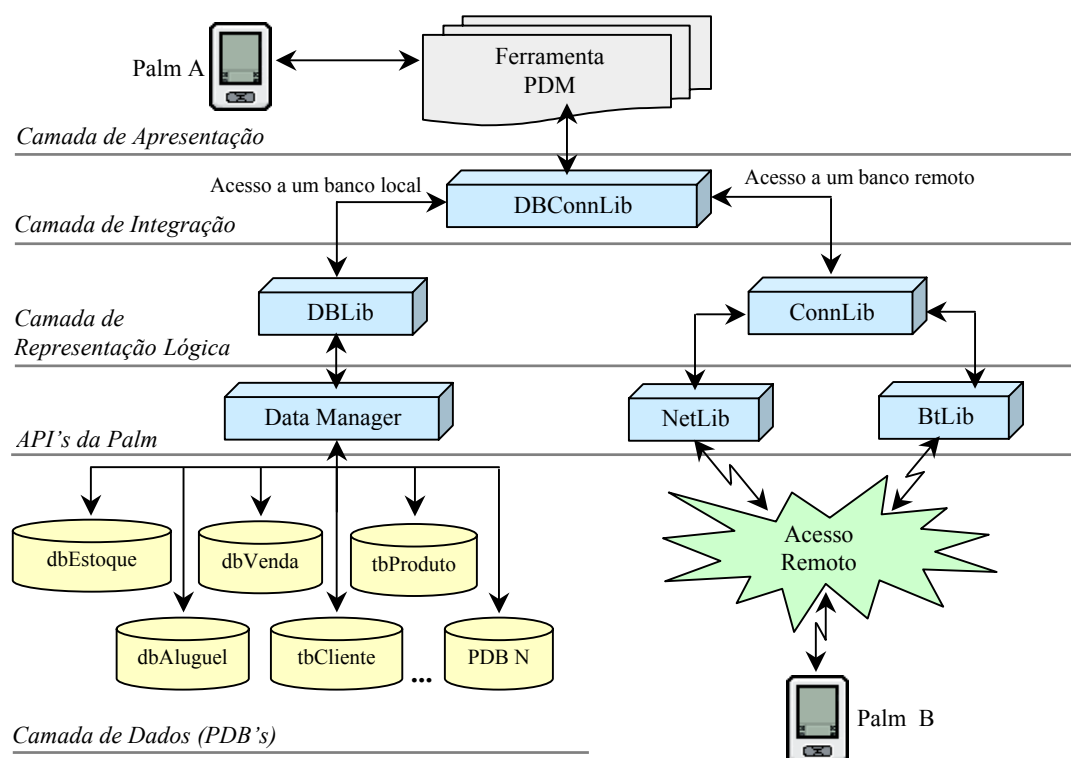
Os dados são armazenados no Palm seguindo uma estrutura sequencial de registros, não possuindo o conceito de campos, tipo de dados, índices ou chave primária. O desenvolvedor é inteiramente responsável por determinar como os dados são gravados nos registros, bem como por determinar o tamanho de cada registro. Dessa forma, a persistência dos dados, no sistema de arquivos do Palm OS, não acontece de forma transparente para a aplicação cliente. Para existir restrições de integridade entre os dados, a aplicação deve garantir esse tipo de propriedade.

## 3. A Ferramenta PDM

A ferramenta *Palm Database Manager* (PDM), desenvolvida em C++, tem como objetivo fornecer propriedades de um sistema gerenciador de banco de dados para ambientes Palm OS. Essa idéia foi motivada pela necessidade de disponibilizar funcionalidades que permitissem ao usuário criar, manipular e visualizar estruturas como bancos de dados, tabelas e relacionamentos entre tabelas nos dispositivos Palm. Além disso, o desenvolvimento de tal aplicação permite validar a utilização das APIs definidas.

A Figura 1 apresenta a estrutura geral da arquitetura em que se baseia a ferramenta PDM. O componente DBLib representa uma biblioteca de APIs cuja principal funcionalidade é prover uma camada de abstração entre aplicação e arquivos do ambiente Palm OS, com o objetivo de abstrair o conceito de arquivo. Para tanto ela utiliza as funções da API da Palm: *Data Manager*. Com isto, qualquer aplicação pode trabalhar com a idéia de banco de dados relacionais e não mais com arquivos Palm (PDB, por exemplo). Na realidade, através da biblioteca DBLib, um desenvolvedor de aplicação para o ambiente operacional Palm OS pode manipular bancos de dados com suas tabelas, restrições de integridade, como chave primária (integridade de chave) ou chave estrangeira (integridade referencial).

As APIs disponibilizadas pela DBLib permitem manipular tais estruturas oferecendo funcionalidades como a inclusão ou exclusão de uma coluna em uma tabela existente através das funções DBLibAddColumn e DBLibDropColumn respectivamente; a inserção, exclusão, alteração ou busca dos dados armazenados nas tabelas através das funções DBLibInsertRecord, DBLibDeleteRecord, DBLibAlterRecord e DBLibFindRecord respectivamente, e; a criação ou exclusão de tabelas (DBLibCreateTable e DBLibDropTable), bancos de dados (DBLibCreateDatabase e DBLibDropDatabase) e restrições de integridade (DBLibCreateRelationship e DBLibDropRelationship).



**Figura 1. Estrutura Geral da Arquitetura PDM**

A aplicação PDM pode realizar o acesso aos dados de forma local ou remota. Se for local, a aplicação chama a DBConnLib que executa as funções da biblioteca DBLib (ver Figura 1). Se a conexão for remota, o usuário deve determinar em qual dispositivo e de que maneira (por *Bluetooth* ou *Network*) a conexão deve ser estabelecida. Para isso, o dispositivo a ser acessado precisa estar executando a aplicação PDM para que possa receber a solicitação remota e executá-la localmente. A cada operação realizada no banco de dados remoto, a aplicação chama a DBConnLib que executa a função de envio de mensagem da ConnLib. Quando essa mensagem for recebida no dispositivo remoto, a aplicação chama a DBConnLib que, por sua vez, interpreta a mensagem recebida formatando-a no padrão esperado pela DBLib. Esta executa a operação localmente e envia para a DBConnLib a resposta apropriada que, por sua vez, a retorna ao Palm solicitante.

Assim, o usuário pode realizar, em um banco de dados localizado em outro dispositivo Palm, todas as operações de definição e manipulação de bancos de dados

disponibilizadas pela biblioteca DBLib remotamente. A DBLib sempre executa suas funções localmente, mesmo que a solicitação seja de um cliente remoto. A DBConnLib é a responsável por gerenciar o acesso local ou remoto realizado pela aplicação.

Uma característica importante da arquitetura PDM é a independência existente entre as APIs DBLib e ConnLib. Isso permite que ambas possam ser utilizadas isoladamente por outras aplicações.

A biblioteca ConnLib disponibiliza um conjunto de APIs que permitem o estabelecimento de conexões com dispositivos remotos e a troca de mensagens entre eles, de forma que nela estão todas as funcionalidades referentes ao acesso remoto. Essa biblioteca utiliza as funcionalidades das APIs de conexão do ambiente Palm OS: NetLib e BtLib. A primeira oferece serviços básicos de rede usando TCP e UDP através de *sockets*. De acordo com as funcionalidades suportadas pelo aparelho utilizado, o usuário pode determinar se deseja se conectar através de Wi-Fi ou utilizando o serviço GPRS. A BtLib também oferece a possibilidade de acesso a outros dispositivos. Inicialmente, o usuário deve utilizar a função de descobrir outros dispositivos ativos ao alcance do sinal e, em seguida, pode escolher o Palm que deseja se conectar.

Após conectar-se ao dispositivo escolhido, o usuário poderá abstrair-se de qual tipo de conexão está utilizando. Todas as funcionalidades disponibilizadas pela ferramenta serão oferecidas de forma idêntica para os diversos tipos de serviços.

Depois de definido o tipo de acesso, o usuário pode manipular as estruturas do banco de dados remoto normalmente. A aplicação permite que possam existir várias tabelas em um banco de dados. Uma tabela pode ser formada por uma ou várias colunas e registros. Para se criar uma tabela é necessário definir pelo menos uma coluna, e para cada coluna é necessário definir atributos como o nome, o tipo, o tamanho e se a coluna é chave primária. A estrutura de uma tabela pode ser alterada através da inclusão ou exclusão de colunas. As colunas também podem ser definidas como chave estrangeira a partir da criação de uma integridade referencial entre duas tabelas. A criação de um relacionamento define uma restrição de integridade entre duas tabelas oferecendo condições para melhor controlar a persistência dos dados armazenados nas tabelas.

Além da manipulação dos esquemas dos bancos de dados, a ferramenta permite visualizar os dados existentes nas tabelas. Para realizar a manipulação dos dados das tabelas, a DBLib possui funções que realizam operações de inclusão, exclusão, alteração e busca dos dados armazenados em uma determinada estrutura.

Dessa forma, a ferramenta PDM apresenta funcionalidades que oferecem um melhor gerenciamento dos dados armazenados em um dispositivo Palm e disponibiliza a conectividade a bancos de dados remotos a partir da arquitetura definida.

As Figuras 2 e 3 mostram algumas telas reais da ferramenta PDM. A Figura 2a mostra a tela inicial com todos os bancos de dados criados. Ao selecionar um determinado banco, será mostrada uma tela, conforme a Figura 2b, que apresenta vários aspectos relacionados ao banco selecionado. A Figura 2c mostra as tabelas criadas neste banco. Ao selecionar uma determinada tabela, será mostrada uma tela, conforme a Figura 3a, que permite explorar vários aspectos relacionados à tabela selecionada. A Figura 3b lista as colunas da tabela mostrando seus atributos. A Figura 3c mostra a tela que permite criar uma integridade referencial entre duas tabelas.



Figura 2. Telas da ferramenta PDM: (2a) Tela inicial que lista todos os banco do Palm; (2b) Aspectos relacionados a um banco de dados; (2c) Listagem das tabelas de um banco.

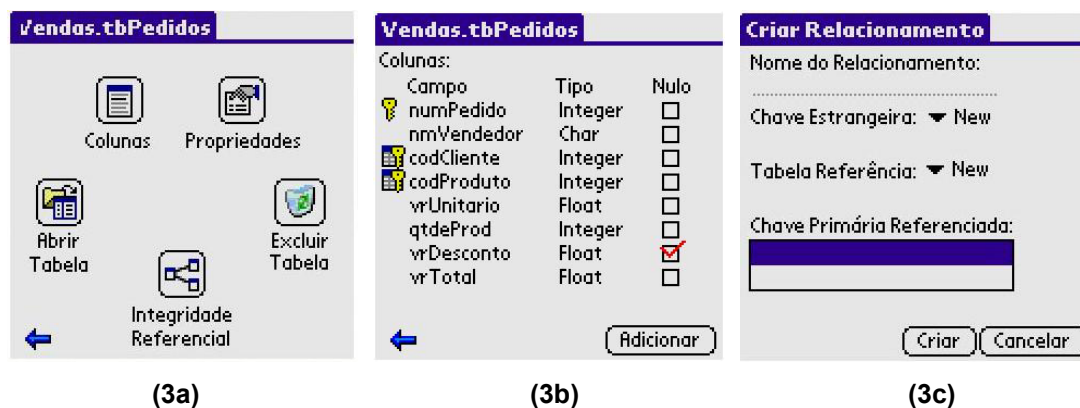


Figura 3. Telas relacionadas às tabelas: (3a) Aspectos relacionados a uma tabela; (3b) Lista com as colunas de uma tabela; (3c) Atributos de uma integridade referencial.

Atualmente, existem no mercado algumas ferramentas que permitem manipular os dados armazenados em arquivos PDB. Entre elas, pode-se citar *MobileDB* da *HandMark*, *JFile* da *Land-J Technologies* e *HandBase* da *DDH Software*. Porém, não se tem conhecimento da existência de uma ferramenta capaz de manipular os dados abstraindo a visão de arquivos, oferecendo propriedades semelhantes às existentes em banco de dados relacionais, bem como funcionalidades de acesso a dados remoto. Tais características, entretanto, podem ser encontradas na ferramenta PDM, conforme mostram as Figuras 2 e 3.

#### 4. Conclusão

A arquitetura apresentada tem a finalidade de gerenciar e armazenar os dados no ambiente operacional Palm OS de acordo com uma estrutura conceitual de banco de dados relacional. Para tanto, foram definidas e implementadas as APIs *DBConnLib*, *DBLib* e *ConnLib*. A *DBConnLib* executa o papel de uma camada intermediária entre a aplicação PDM e as APIs *ConnLib* e *DBLib*. Esta última utiliza o conceito de esquema dos bancos de dados relacionais e a *ConnLib* oferece ao usuário a possibilidade de conexão remota. Essas camadas facilitam o acesso ao banco de dados, além de possibilitar o gerenciamento e manipulação das informações de forma local e remota.

Como consequência destes benefícios, a utilização da arquitetura PDM permite uma maior consistência dos dados armazenados no sistema de arquivos do Palm. Esta estrutura dispõe de diversas funcionalidades, como a manutenção de integridade referencial, objetivando, portanto, um maior grau de usabilidade e uma maior persistência dos dados gerados a partir das bibliotecas definidas.

Como trabalhos futuros, serão incorporadas as seguintes funcionalidades à ferramenta PDM:

- (i) permitir a execução de expressões da DML (*select, insert, update e delete*) da linguagem SQL;
- (ii) possibilitar a manipulação de dados armazenados em formato de compressão. Dessa forma, os dados estarão sempre em um formato de compressão. O processo de descompressão será realizado automaticamente no acesso aos dados;
- (iii) tornar transparente o processo de *handoff* vertical. Dessa forma, ficará transparente para a aplicação se, em função do deslocamento do dispositivo móvel no espaço, há a necessidade de troca de tecnologia de comunicação. Por exemplo, dois dispositivos móveis podem estar conectados via *Bluetooth*, mas em função do deslocamento de um deles, houve a perda da conexão, pois ficaram fora da área de alcance do padrão *Bluetooth*. Contudo, os dois dispositivos podem estar participando de uma rede Wi-Fi (que tem alcance maior). Portanto, através de classes do *framework*, será garantida uma troca de tecnologia de conexão de *Bluetooth* para Wi-Fi;
- (iv) fornecer aos dispositivos móveis e às aplicações a capacidade de obter informação do ambiente computacional em que estão inseridos e utilizar esta informação para, dinamicamente, construir modelos computacionais;
- (v) realizar uma análise da arquitetura com o objetivo de avaliar sua escalabilidade e desempenho visando promover possíveis otimizações.

## 5. Bibliografia

- Davis, Gordon B. *Anytime/Anyplace Computing and the Future of Knowledge Work*. *Communications of the ACM*. Dezembro 2002, p. 67-73.
- Palm, Inc. *Palm OS 68K SDK Documentation: Palm OS File Format Specification*. Maio 2001. Disponível em [http://www.palmos.com/dev/support/docs/palmos/68k\\_books.html](http://www.palmos.com/dev/support/docs/palmos/68k_books.html). Último acesso em 15/07/2004.
- Palm, Inc. *Palm OS 68K SDK Documentation: Palm OS Programmer's API Reference*. Setembro 2003. Disponível em [http://www.palmos.com/dev/support/docs/palmos/68k\\_books.html](http://www.palmos.com/dev/support/docs/palmos/68k_books.html). Último acesso em 15/07/2004.
- Palm, Inc. *Palm OS 68K SDK Documentation - Palm OS Programmer's Companion - Volume I*. Setembro 2003. Disponível em [http://www.palmos.com/dev/support/docs/palmos/68k\\_books.html](http://www.palmos.com/dev/support/docs/palmos/68k_books.html). Último acesso em 15/07/2004.
- Palm, Inc. *Palm OS Protein SDK Documentation - Exploring Palm OS: Memory, Databases, and Files*. Janeiro 2004. Disponível em [http://www.palmos.com/dev/support/docs/protein\\_books.html](http://www.palmos.com/dev/support/docs/protein_books.html). Último acesso em 15/07/2004.

## ***ClockTool: Uma Ferramenta Web para Desenvolvimento de Bancos de Dados Temporais***

**Eugênio O. Simonetto<sup>1</sup>, Duncan D.A. Ruiz<sup>2</sup>, Marcio T. Nazari<sup>1</sup>**

<sup>1</sup>Curso de Sistemas de Informação – Centro Universitário Franciscano (UNIFRA)  
Andradas, 1614 – 97015-032 – Santa Maria – RS – Brasil

<sup>2</sup>Faculdade de Informática – Pontifícia Universidade Católica do Rio Grande do Sul  
(PUCRS) Av. Ipiranga, 6681 - Partenon - 90619-900 – Porto Alegre-RS-Brasil

eosimonetto@unifra.br , duncan@inf.pucrs.br , mtamiosso@terra.com.br

**Abstract.** *The work introduces a web tool called ClockTool that allows a friendly incorporation of temporal properties on tables and attributes of SQL data-base models, exploiting data types proper to represent time and the use of database triggers. The designer, or the user in charge of temporal information of an information system, can obtain all needed structures for representing the time aspect in his/her database, as well as the maintenance of his/her temporal data. It is accomplished in accessing the database schema produced by the implementation design phase via ClockTool, before loading such schema in the DBMS.*

**Resumo.** *O trabalho apresenta uma ferramenta web, chamada ClockTool, que permite agregar propriedades temporais a tabelas e atributos de modelos de bancos de dados SQL, de maneira amigável, explorando tipos de dados para representação de tempo e com o uso de gatilhos (triggers). O projetista, ou o usuário responsável pelas informações temporais de um sistema de informação, pode, a partir da definição do esquema de um banco de dados (após o projeto de implementação e antes da carga do modelo no SGBD), obter todas as estruturas necessárias para a representação do aspecto tempo em seu banco de dados, bem como a manutenção de seus dados temporais.*

### **1. Introdução**

Sistemas de Gerência de Banco de Dados (SGBDs) são sistemas que representam e gerenciam os dados das aplicações. Para que estes sistemas possam representar situações do mundo real com a maior fidelidade possível, é necessário introduzir a dimensão tempo a eles, visto que os SGBDs pouco oferecem, em termos de funcionalidades, para o tratamento adequado de informações com características temporais (Edelweiss, 1998). Isto não significa que os sistemas de informação não tratem este tipo de informação.

Vista esta deficiência por parte dos SGBDs de primeira linha, foi proposto um modelo (Simonetto, 2000) que tem por objetivo incorporar aspectos da representação temporal aos bancos de dados do tipo relacional, utilizando-se somente das funcionalidades apresentadas por estes, padronizadas ou não, conforme (SQL, 1999). O modelo utiliza os

conceitos de estados do banco de dados, intervalo temporal, tempo de transação e tempo de validade (Snodgrass, 2000).

Em bancos de dados temporais, as operações de atualização devem garantir que nenhum dado, que se torne antigo, venha a ser perdido. Portanto, todas as atualizações devem ser seguidas de operações para manutenção do valor antigo. No modelo proposto, para tais atualizações, foram desenvolvidas rotinas (utilizando-se gatilhos – *triggers* – dos SGBDs) que atualizam automaticamente os rótulos temporais nas tabelas componentes do modelo. A construção manual desses componentes é exaustiva e difícil, pois é muito grande a quantidade de linhas de código, entre gatilhos e tabelas, a serem produzidos. Ciente dessa dificuldade, desenvolvemos *ClockTool*, uma ferramenta *web* para apoiar esta etapa do desenvolvimento de uma aplicação com características temporais.

A utilização da *ClockTool* se dá após a conclusão do projeto de implementação do banco de dados via uma ferramenta CASE, e antes da carga de especificações no SGBD. Por exemplo, DB-MAIN (DBMain, 2004) é uma ferramenta CASE que opera desta maneira: ao final do projeto, a geração do esquema do banco de dados é feita em um arquivo texto, chamado de *script* do BD. Posteriormente, o administrador de banco de dados (DBA) *carrega* o *script* no SGBD. Modificar tal *script*, portanto, é algo que não implica em alterar as rotinas de trabalho das equipes de desenvolvimento de aplicações; apenas acrescenta mais um passo nessas rotinas, o que é inevitável caso se queira a incorporação de aspectos temporais nos BDs das aplicações.

## 2. *ClockTool* - A Ferramenta Proposta

Para a especificação da ferramenta de apoio ao desenvolvimento do modelo de dados, procuramos identificar as diferentes etapas que constituem o processo de desenvolvimento do modelo e, que poderiam vir a ser apoiadas por algum tipo de ferramenta computacional. Foram identificadas as seguintes etapas:

1. *Análise do Esquema Inicial;*
2. *Classificação dos Atributos;*
3. *Criação das Estruturas do Modelo.*

Após a execução destas etapas é obtido o *script* final, conforme o tipo de banco de dados temporal (tempo de transação e/ou tempo de validade), para ser submetido a um determinado SGBD. Este *script* é composto por todos os gatilhos e tabelas derivados do esquema inicial do banco de dados. Das três etapas, a única que há a interação com o usuário é a de classificação dos atributos, onde estes são classificados como temporais ou estáticos, conforme o ponto de vista (ou a necessidade) do projetista do sistema. Tempo de transação foi implementado em conformidade com o modelo proposto por Simonetto (2000), e tempo de validade foi implementado conforme Snodgrass (2000).

### 2.1 - Análise do Esquema Inicial

A primeira etapa envolve o reconhecimento do banco de dados a ser submetido à ferramenta, bem como para identificar para qual SGBD deve ser gerado o *script* do banco de dados. Na *ClockTool*, a interface para seleção do SGBD e da localização do arquivo que contém o *script* original é apresentada na figura 2.1. Também nesta etapa, são identificadas todas as tabelas componentes do banco de dados e, dentro de cada tabela todos os seus atributos, bem como sua chave primária (*primary key*). Esta etapa tem por finalidade preparar os atributos para sua posterior classificação.



O esquema do banco de dados a ser reconhecido pela ferramenta, deve ser fornecido através de um arquivo *script*, gerado por uma ferramenta CASE ou pelo projetista do sistema. É dada, na interface inicial (figura 2.1), a opção de o usuário criar seu *script* original a partir da própria *ClockTool*.

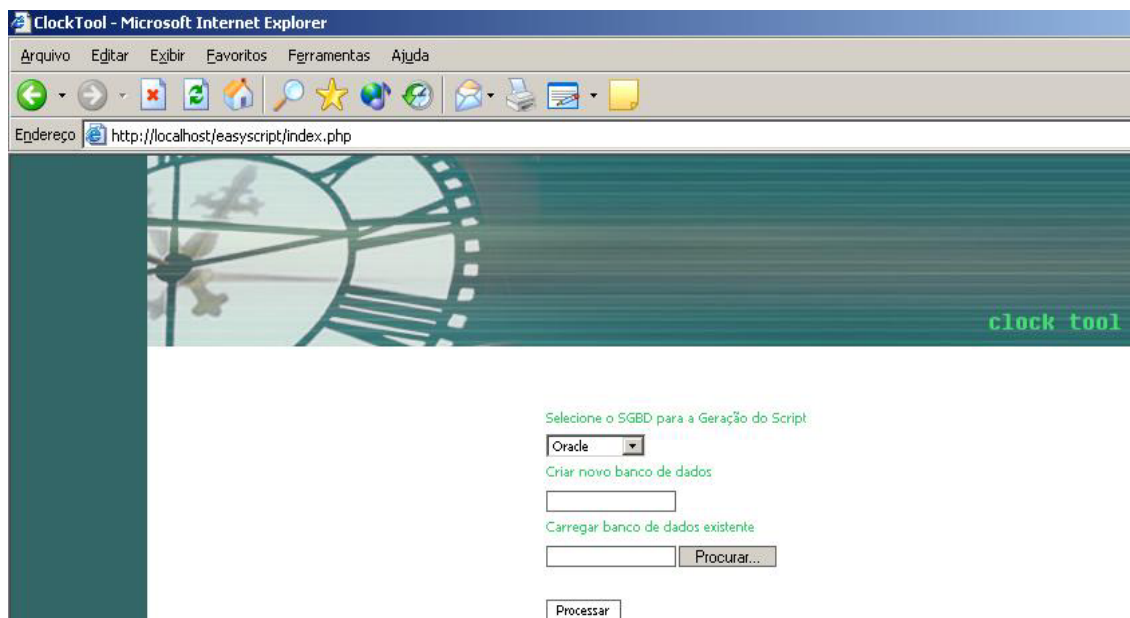


Figura 2.1 – Interface da *ClockTool* para a seleção do SGBD e do *script* original.

## 2.2 - Classificação dos Atributos

Nesta etapa, o projetista deve classificar cada atributo dentro de cada tabela. Tal classificação consiste na identificação do tipo de cada atributo como sendo: estático (que não varia com o correr do tempo) ou dinâmico (que podem variar com o correr do tempo). Em uma tabela, além dos atributos componentes da chave primária, poderão ter nenhum ou vários atributos temporais (dinâmicos) e, nenhum ou vários atributos estáticos. Nesta etapa, também, o projetista define o tipo de banco de dados temporal a ser gerado pela ferramenta: tempo de transação e/ou tempo de validade (Snodgrass, 2000). Quando o tipo de banco de dados a ser gerado for bi-temporal, ambos os *checkbox* deverão ser selecionados. A interface de classificação dos atributos e do tipo de banco de dados temporal pode ser visualizada nas figuras 2.2 e 2.3.

Na interface para classificação dos atributos, a ferramenta disponibiliza ao projetista do sistema todos os atributos de cada tabela componente do sistema, para estes serem submetidos à classificação. Após a classificação dos atributos, é dada ao usuário a opção **Gerar Script**, a qual disponibilizará ao usuário o novo *script* (produto final da ferramenta), para ser submetido ao SGBD. Após a classificação dos atributos por parte do projetista do sistema se dá à parte de criação das estruturas. A interface da ferramenta para todo o processo de classificação dos atributos, tipo de banco de dados temporal a ser gerado e geração de *script*, é apresentada na figura 2.4.

TABELA: funcionario

ESTÁTICO	TEMPORAL	NOME DO CAMPO	TIPO	NOT NULL
<input type="checkbox"/>	<input type="checkbox"/>	cod_funcionario	integer	SIM
<input type="checkbox"/>	<input checked="" type="checkbox"/>	cod_empresa	integer	SIM
<input type="checkbox"/>	<input checked="" type="checkbox"/>	cod_setor	integer	SIM
<input type="checkbox"/>	<input checked="" type="checkbox"/>	nome	varchar(80)	SIM
<input type="checkbox"/>	<input checked="" type="checkbox"/>	endereco	varchar(60)	SIM
<input checked="" type="checkbox"/>	<input type="checkbox"/>	nascimento	date	
<input type="checkbox"/>	<input checked="" type="checkbox"/>	fone	varchar(15)	
<input type="checkbox"/>	<input checked="" type="checkbox"/>	email	varchar(60)	

Figura 2.2 – Interface para a classificação dos atributos

:: Selecione o tempo do banco de dados temporal

☒ TEMPO DE TRANSAÇÃO ☐ TEMPO DE VALIDADE

Figura 2.3 – Interface para a seleção do tipo de banco de dados a ser gerado

ClockTool - Microsoft Internet Explorer

Arquivo Editar Exibir Favoritos Ferramentas Ajuda

Endereço http://localhost/easyscript/index.php

clock tool

:: Selecione o tempo do banco de dados temporal

☒ TEMPO DE TRANSAÇÃO ☐ TEMPO DE VALIDADE

TABELA: funcionario

ESTÁTICO	TEMPORAL	NOME DO CAMPO	TIPO	NOT NULL
<input type="checkbox"/>	<input type="checkbox"/>	cod_funcionario	integer	SIM
<input type="checkbox"/>	<input checked="" type="checkbox"/>	cod_empresa	integer	SIM
<input type="checkbox"/>	<input checked="" type="checkbox"/>	cod_setor	integer	SIM
<input type="checkbox"/>	<input checked="" type="checkbox"/>	nome	varchar(80)	SIM
<input type="checkbox"/>	<input checked="" type="checkbox"/>	endereco	varchar(60)	SIM
<input checked="" type="checkbox"/>	<input type="checkbox"/>	nascimento	date	
<input type="checkbox"/>	<input checked="" type="checkbox"/>	fone	varchar(15)	
<input type="checkbox"/>	<input checked="" type="checkbox"/>	email	varchar(60)	

Figura 2.4 – Interface da *ClockTool* para as classificações e geração do *script*

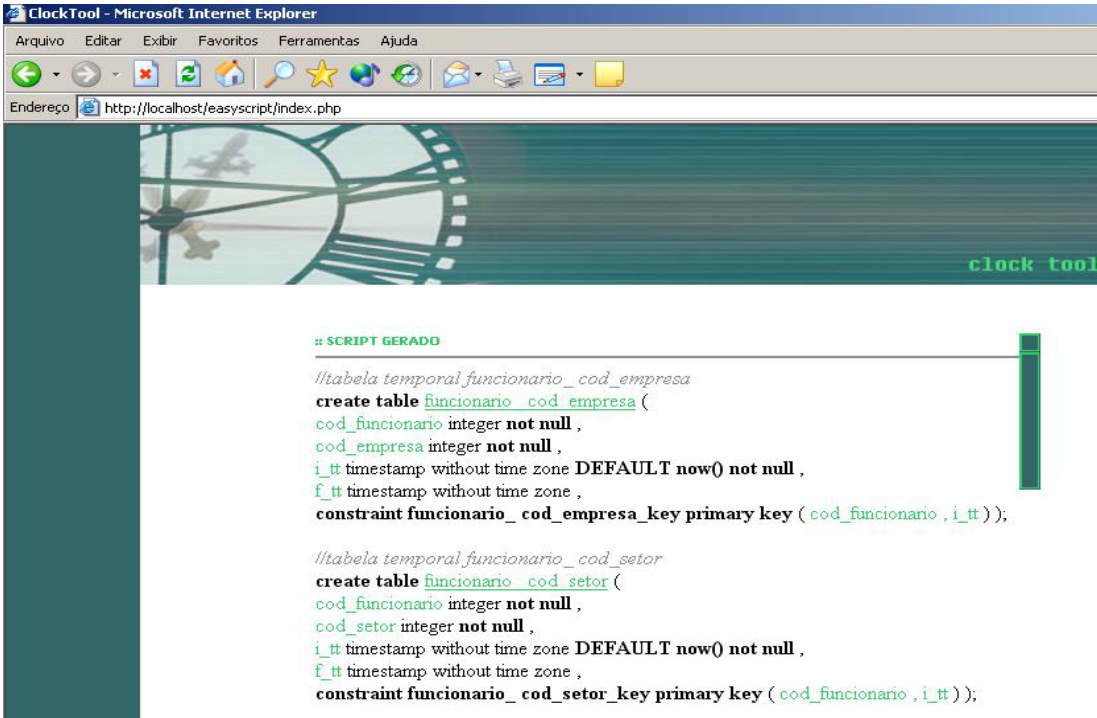
## 2.3 - Criação das Estruturas do Modelo

Nesta etapa de funcionamento da ferramenta, são construídas as estruturas que garantem a manutenção e o armazenamento automático do dado temporal. Conforme a classificação para os atributos de cada tabela, executada na etapa anterior, são

determinadas as estruturas correspondentes. As novas estruturas implicam na criação de *triggers*, procedimentos e novas tabelas, na forma de um arquivo *script*.

Para cada tabela do modelo, a primeira estrutura a ser gerada pela ferramenta é a própria relação definida pelo usuário que, no modelo proposto por Simonetto (2000), chama-se **tabela instantânea**. Para cada atributo classificado como **temporal**, é gerada uma **nova tabela** e um conjunto de rotinas de manutenção do dado temporal que manipule este tipo de dado a cada inserção, atualização e remoção.

Já para os atributos, de uma tabela, classificados como **estáticos** é definida **somente uma tabela**, bem como suas respectivas **rotinas de manutenção do dado temporal**, no caso de inserção e remoção (pois o atributo estático não varia com o correr do tempo) de algum valor de atributo. O modo como o novo *script* é apresentado pela ferramenta pode ser visualizado na figura 3.1.



```

-- SCRIPT GERADO

//tabela temporal funcionario_cod_empresa
create table funcionario_cod_empresa (
cod_funcionario integer not null ,
cod_empresa integer not null ,
i_tt timestamp without time zone DEFAULT now() not null ,
f_tt timestamp without time zone ,
constraint funcionario_cod_empresa_key primary key ( cod_funcionario , i_tt ));

//tabela temporal funcionario_cod_setor
create table funcionario_cod_setor (
cod_funcionario integer not null ,
cod_setor integer not null ,
i_tt timestamp without time zone DEFAULT now() not null ,
f_tt timestamp without time zone ,
constraint funcionario_cod_setor_key primary key ( cod_funcionario , i_tt ));
  
```

Figura 3.1 - Interface para Apresentação do *script* final pela *ClockTool*

### 3. Considerações Finais e Próximos Passos

Neste trabalho foi apresentada uma ferramenta *web* para o desenvolvimento de bases de dados temporais, denominada *ClockTool*. A *ClockTool* permite ao projetista de sistemas de informação, após a concepção do projeto de implementação do banco de dados e, antes da carga do mesmo em algum SGBD como Oracle e PostgreSQL, gerar um novo modelo de dados com as estruturas necessárias ao armazenamento e manutenção dos dados para os quais deseja-se registrar suas variações com o correr do tempo.

Os resultados obtidos utilizando-se a ferramenta foram considerados satisfatórios, visto que transformaram fielmente os *scripts* originais dos modelos de dados dos usuários, em modelos de dados temporais, de acordo com as classificações executadas.

Na primeira etapa do desenvolvimento da ferramenta, para fins de validação, implementamos o reconhecimento e a geração de *scripts* para os SGBDs PostgreSQL (Momjian, 2001) e Oracle (Morelli, 2000). Isso se deve ao fato que mesmo havendo um grau de conformidade considerável com os padrões SQL por parte desses, sempre existem variações sintáticas, próprias de cada ferramenta. Em especial, os tipos de dados para representação de tempo variam consideravelmente (no Oracle, tipo *date*). Na parte de especificação dos gatilhos, esses SGBDs apresentam conformidade na especificação dos cabeçalhos dos mesmos; a lógica de realização desses gatilhos obedece a sintaxes próprias de cada produto. Como próximo passo no desenvolvimento da ferramenta, iremos desenvolver a geração de *scripts* para os SGBDs MS SQL Server e DB2. Por fim, cabe ressaltar que a ferramenta foi desenvolvida utilizando-se a linguagem PHP (*Hipertext Preprocessor*). Também está sendo examinado o uso da ferramenta na geração de *scripts* do modelo de dados de um sistema de gerência de *workflows* (Ruiz, 2003). Espera-se diminuir a complexidade da programação necessária para implementar tal sistema.

### Agradecimentos

Os autores gostariam de agradecer o CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico, pelo suporte parcial aos autores. Em especial, pelo projeto CWf-Flex (Ruiz, 2003). E à Unifra, pela concessão da bolsa de iniciação científica.

### Referências Bibliográficas

- DB-MAIN Case. **Db-Main Tutorial**. <http://www.info.fundp.ac.be/~dbm>. Último acesso: junho/2004.
- EDELWEISS, N. **Bancos de Dados Temporais: Teoria e Prática**. JAI'98 – UFMG, 1998.
- MORELLI, E.T. **Oracle 8 – SQL, PL/SQL e Administração**. Érica, 2000.
- MOMJIAN, B. **PostgreSQL – Introduction and Concepts**. Addison-Wesley, 2001.
- RUIZ, D.D.; COPSTEIN, B.; OLIVEIRA, F. **Um ambiente para gestão flexível de processos de testes de software, baseado na tecnologia de workflow e no uso de ferramentas de software livre, com alocação de recursos**. Projeto de Pesquisa CNPq, CT-INFO, PDPG-TI. (Início 02/2003, término previsto 12/2004).
- SIMONETTO, E.O.; RUIZ,D.D. A Proposal Model to Incorporation of Temporal Aspects to the Relational DBMS Using Active Databases Concept. In: IEEE Fourth International Workshop on Databases and Information Systems, **Proceedings...** v1, p.52-59, Vilnius-Lithuania, 2000.
- SNODGRASS, R.T. **Developing Time-Oriented Database Applications in SQL**. Morgan Kaufmann Publishers, 2000.
- AMERICAN NATIONAL STANDARDS INSTITUTE. **ANSI/ISO/IEC 9075-1:1999- Information Systems-Database Language - SQL**. September,1999.

## TVM WeB - Uma Interface Visual para o Modelo Temporal de Versões

Vincent Nelson Kellers da Silveira, Nina Edelweiss e Renata de Matos Galante

Instituto de Informática - Universidade Federal do Rio Grande do Sul  
Caixa Postal 15.064 - 91.501-970 – Porto Alegre - RS - Brazil

{vincent, nina, galante}@inf.ufrgs.br

**Abstract.** *This paper describes TVM WeB: Temporal Versions Model Web Browser, a graphical browsing and querying tool, developed for the Temporal Versions Model (TVM). TVM WeB aims to supply the user with a friendly and easy learning interface that explores the temporal and versioned information stored in a TVM database, as well as TVM databases own properties.*

**Resumo.** *Este trabalho apresenta o TVM WeB: Temporal Versions Model Web Browser, uma ferramenta gráfica de navegação e consulta desenvolvida para o Modelo Temporal de Versões (TVM), que visa fornecer ao usuário uma interface amigável, de fácil aprendizado que explora as informações temporais e versionadas contidas em uma base TVM, bem como propriedades de bases de dados deste modelo.*

### 1. Introdução

As mais diversas aplicações, tais como das áreas científica, financeira, administrativa e comercial necessitam armazenar em bases de dados a evolução temporal dos seus dados. A evolução dos dados destas aplicações é implementada através do uso de modelagem temporal e de bases de dados que suportem esta modelagem.

Igualmente importante para diversas aplicações é o conceito de versão, que modela diferentes estágios da evolução de uma mesma entidade em tempos distintos ou sob diferentes pontos de vista. A técnica de versionamento possui aplicação em áreas como CAD (*Computer Aided Design*), CASE (*Computer Aided Software Engineering*) e SCM (*Software Configuration Management*), permitindo o trabalho cooperativo e a configuração de alternativas para projetos de software [Moro 2002], por exemplo.

A combinação da modelagem temporal e versionada permite acesso aos valores alterados ao longo do tempo nas diversas versões de uma entidade, inclusive as modificações descartadas, sendo que este acesso pode ser restrito a um determinado período no tempo. Tal resultado não pode ser alcançado com uso exclusivo da modelagem de versões, posto que as dimensões temporais, fundamentais para que esta flexibilidade de acesso seja atingida, não são tratadas pelo modelo de versões.

O TVM (*Temporal Versions Model*) [Moro 2002] incorpora ao conceito de versão o conceito de tempo, a fim de permitir o acompanhamento da evolução temporal dos objetos, versionados ou não, armazenados em uma base de dados.

Ferramentas gráficas unem as capacidades de grandes bases de dados e linguagens de consulta e disponibilizam ao usuário a possibilidade de construção de buscas e navegação em bases de dados através de interfaces que abstraem o conhecimento técnico requerido por interfaces textuais. A fim de aumentar o

rendimento de usuários com pouca familiaridade com a ferramenta, usualmente são oferecidos recursos que simulam graficamente a construção de consultas e a exibição de esquemas. Por outro lado, tais ferramentas gráficas oferecem a possibilidade de utilização de linguagens de consulta, permitindo a usuários já experimentados aumento de desempenho na execução de suas tarefas.

Este trabalho descreve o navegador visual do Modelo Temporal de Versões (TVM – *Temporal Versions Model*). Esse navegador foi desenvolvido com base nas características específicas do TVM, em estudos sobre as possíveis formas de representação de informações armazenadas em bancos de dados e na análise das funcionalidades de ferramentas visuais para navegação e consulta em bases de dados.

O objetivo deste trabalho foi produzir uma interface visual desenvolvida em ambiente web para navegação e consulta a uma base de dados TVM. A opção pelo ambiente web ocorreu devido à larga utilização de aplicações instaladas neste ambiente, que oferece independência de plataforma por parte do cliente e facilidade na alteração do formato de apresentação dos dados ao usuário. A aplicação possui formato visual, pois esta representação facilita a compreensão dos dados por parte do usuário.

Este trabalho encontra-se dividido em 5 seções. A seção 2 apresenta o Modelo Temporal de Versões. A seção 3 apresenta o TVM WeB e sua arquitetura. Na seção 4 é realizada uma comparação da ferramenta desenvolvida com outros trabalhos. Na seção 5 são apresentadas as conclusões do artigo e trabalhos futuros.

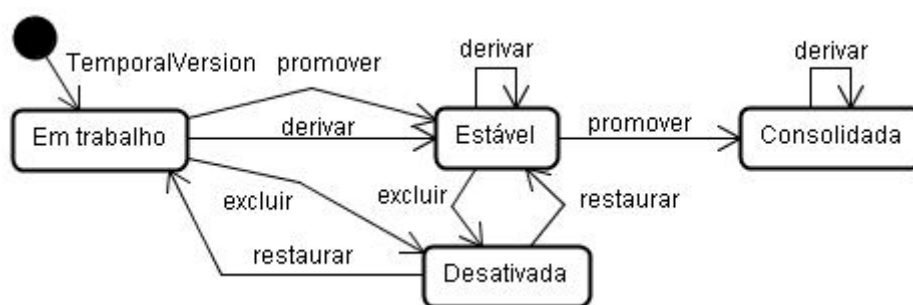
## 2. Modelo Temporal de Versões

O Modelo Temporal de Versões - TVM (*Temporal Versions Model*) [Moro 2002] permite armazenar o histórico das modificações realizadas nas diferentes versões dos dados, além dos tempos de vida de cada versão.

Os elementos que podem ser temporalizados no TVM são objetos, versões, atributos e relacionamentos. Como um objeto possui uma linha de tempo para cada uma de suas versões, e um objeto pode possuir diversas versões, geram-se duas ordens de tempo: o tempo ramificado e o tempo linear. O tempo ramificado refere-se às diferentes linhas de tempo das versões de um objeto, e o tempo linear corresponde à evolução temporal de cada versão. A temporalidade é representada através de um rótulo de tempo que possui variação discreta e modela duas dimensões temporais: o tempo de validade e o tempo de transação [Jensen 1998].

A hierarquia do TVM permite ao usuário definir dois tipos de classes: classe de aplicação não temporal e não versionável, uma classe que não possui definição de dados versionados nem temporalizados e classe de aplicação temporal e versionada, uma classe onde atributos e relacionamentos podem ser definidos como estáticos ou temporalizados. Instâncias desta classe, que podem ser objetos não versionados, objetos versionados e versões, possuem um rótulo de tempo associado.

O TVM prevê regras utilizadas nas operações de atualização da base de dados, que visam garantir que o tempo de vida de uma versão estará contido no tempo de vida do objeto versionado, ao passo que o tempo de vida de um objeto, seja este objeto um atributo ou relacionamento, estará contido no tempo de vida da versão.



**Figura 1. Diagrama de estados de uma versão**

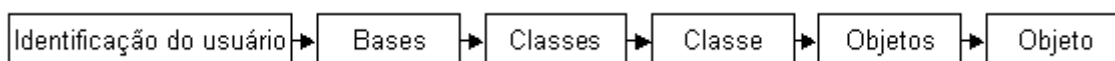
Toda versão possui um *status* associado, sendo que este *status* pode ser modificado ao longo da vida de uma versão devido a eventos especiais exibidos na Figura 1. Estes *status* são:

- em trabalho - estado que uma versão recebe ao ser criada na base de dados. Neste *status*, além de ser considerada temporária, uma versão pode ser apagada e alterada;
- estável - versões nesse *status* não podem ser mais alteradas, mas podem ser logicamente apagadas. Versões em trabalho atingem o *status* estável quando são promovidas pelo usuário ou quando usadas na derivação de uma nova versão;
- consolidada - neste *status* as versões não podem ser apagadas nem modificadas. Uma versão é consolidada através de uma ação explícita do usuário;
- desativada - a versão pode ser apenas consultada e restaurada. Representa uma versão excluída logicamente.

### 3. Arquitetura e funcionalidades do TVM WeB

O modelo de arquitetura utilizado na construção do TVM WeB é o MVC - *Model View Controller* [Gamma 1995]. Neste modelo, os *beans* são classes que representam objetos de uma realidade modelada. Os *controllers* são responsáveis pelo fluxo de navegação do usuário da aplicação, identificando que ações ele pode tomar, sendo os *modellers* responsáveis por implementar as ações disponibilizadas ao usuário.

A ferramenta desenvolvida neste trabalho utiliza a base de dados IBM DB2, escolhida por ser um banco de dados que opera de acordo com o modelo objeto relacional e por ser o mais próximo ao SQL-92 no suporte aos tipos de dados temporais. A ferramenta tem suas funcionalidades orientadas para a web, sendo o fluxo de ativação dos diferentes navegadores do TVM WeB apresentado na Figura 2.



**Figura 2. Fluxo de utilização dos navegadores**

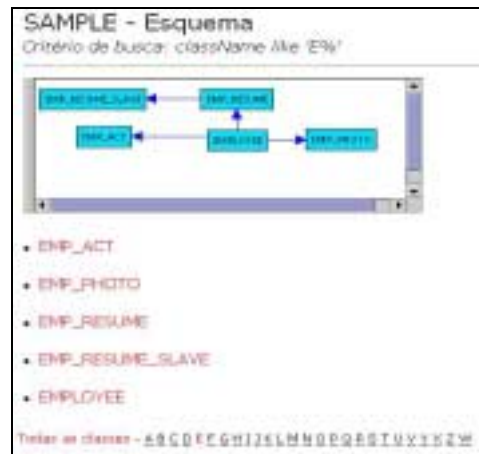
O TVM WeB possui um mecanismo de identificação do usuário que está iniciando a sessão de trabalho. Este dispositivo visa restringir apenas a determinadas bases de dados o acesso de usuários habilitados a utilizar a ferramenta.

Ao iniciar uma sessão de trabalho na ferramenta, o usuário deve selecionar no navegador de bases de dados a base que deseja visualizar, de acordo com as restrições



de acesso aplicadas a cada usuário cadastrado na base de dados. Após a seleção de uma determinada base de dados, o navegador de classes é invocado.

O navegador de classes, ilustrado na Figura 3, permite que o esquema de uma base de dados seja visualizado graficamente, em formato semelhante àquele utilizado em diagramas de classe. Este navegador ainda oferece ao usuário a possibilidade de selecionar uma determinada classe e visualizar os seus detalhes.



**Figura 3. O navegador de classes**

O navegador de classe, ilustrado na Figura 4, permite ao usuário a identificação dos atributos de uma classe, de classes relacionadas, das operações implementadas pela classe e permite a exibição dos objetos da classe selecionada, com destaque para atributos e relacionamento temporalizados. A exibição de objetos pode ser restringida através da especificação de um critério de busca SQL.

**Figura 4. Navegador de classe**

O navegador de objetos exibe todos os objetos de uma classe, ou ainda, objetos que atendem a um critério de busca, com destaque para os objetos versionados. Os atributos temporalizados possuem o seu valor corrente exibido tanto neste quanto no navegador de objeto. Por utilizar representação textual dos objetos, o navegador permite ao usuário definir a ordem de exibição dos atributos e a ordenação em que os próprios



objetos são exibidos. As configurações de interface realizadas por cada usuário são persistidas, para que em uma próxima sessão de trabalho sejam reutilizadas.

O navegador de objeto, ilustrado na Figura 5, permite ao usuário visualizar a evolução temporal dos atributos de um objeto, a sua posição dentro da hierarquia de versões, realizar navegação sincronizada nos objetos relacionados ao objeto visualizado, além de permitir seleção de outros objetos do grafo de versões, a fim de que estes sejam analisados em igual nível de detalhe.



Figura 5. O navegador de objeto

No navegador de classe e de objeto é permitido ao usuário definir quais seções serão visíveis bem como em que ordem as diferentes seções do navegador serão exibidas. As configurações definidas por cada usuário da ferramenta são persistidas a fim de serem reutilizadas nas sessões de trabalho seguintes.

#### 4. Comparação com outros trabalhos

Apresentado em [Agrawal 1990], o OdeView é um navegador que permite a visualização gráfica de uma base de dados Ode, com suporte a navegação sincronizada e a consultas definidas através do preenchimento de formulários ou através de um modo semelhante àquele implementado pelo QBE.

PESTO [Carey 1996] é um navegador independente de bases de dados, com funcionalidades semelhantes às do OdeView. O PESTO diferencia-se nas consultas, pois estas utilizam um paradigma de buscas integradas à visualização de dados. PESTO permite que critérios de busca como seleção, negação e disjunção sejam utilizados sem que o usuário escreva código explícito de consulta.

A ferramenta definida em [Silva 1998] opera com bancos de dados versionados, e utiliza o conceito de navegadores com interfaces híbridas especializados em diferentes porções de uma base de dados.

O TVM WeB utiliza conceitos introduzidos por estas ferramentas, como a navegação sincronizada, a especificação de critérios de busca integrada à consulta de dados e navegadores especializados em diferentes porções de uma base de dados.

Comparado a estas ferramentas, o TVM WeB diferencia-se por ser voltado para bancos de dados TVM explorando as características de temporalidade e versionamento destas bases e por utilizar o ambiente web, que além de possuir aplicações muito conhecidas atualmente, garante independência de plataforma por parte do usuário.

## **5. Conclusões e trabalhos futuros**

Composto por diferentes navegadores especializados e com funcionalidades extraídas de ferramentas semelhantes, o TVM WeB permite ao usuário não familiarizado com navegadores de bases de dados o acesso às informações armazenadas em bases TVM, como a visualização da evolução temporal e das versões de objetos, sem que consultas específicas para tanto sejam escritas, bem como a visualização de objetos não versionados e não temporalizados. Visando facilidade de uso, a ferramenta é desenvolvida em ambiente web, uma vez que aplicações baseadas neste ambiente estão amplamente difundidas. Devido à limitação de espaço, apenas as principais características da ferramenta estão detalhadas neste trabalho, onde é apresentado um resumo do estudo que compara o TVM WeB com outras ferramentas existentes.

Após a implementação inicial do TVM WeB, algumas questões persistem, como a utilização de interfaces gráficas para consultas, o suporte à linguagem TVQL (linguagem de consulta do TVM, definida em Moro, 2002) nos navegadores de objeto e de objetos e a persistência do posicionamento do grafo de classes.

## **Referências**

- AGRAWAL, R.; GEHANI, N. H.; SRINIVASAN, J. OdeView: The Graphical Interface to Ode. SIGMOD Record, New York, v.9, n.2, June 1990. ACM SIGMOD International Conference on Management of Data, 1990.
- CAREY, M. et al. PESTO: An Integrated Query/Browser for Object Databases. In: VLDB CONFERENCE, 22., 1996, Mubai, India. Proceedings.
- GAMMA, E. et al. Design Patterns. Addison-Wesley Pub Co., 1995.
- JENSEN, C.S. et al. The Consensus Glossary of Temporal Database Concepts February 1998 Version. In: ETZION, O. et al. Temporal Databases Research and Practice. Berlin-Heidelberg: Springer-Verlag, 1998. p. 367-405.
- MORO, M. M.; EDELWEISS, N.; SANTOS, C. S. Modelo Temporal de Versões. IX Concurso de Tesis de Maestría Clei-UNESCO, Primeiro lugar, XXVIII Conferencia Latinoamericana de Informatica, November 2002, Montevideo, Uruguay, pp.116.
- SILVA, J. T. Uma Interface Visual para Modelos de Bancos de Dados Orientados a Objeto com Suporte para Versões. Porto Alegre: PPGC da UFRGS, 1998. Dissertação de Mestrado.

## **Índice por Autor** ***Author Index***

Abílio Fernandez	25
Adriana Fernandez	25
Ana Maria de C. Moura	25
Ana Paula Appel	19
Andrei Eneas Nunes Lima	13
Angelo Brayner	43
Caetano Traina Júnior	19
Carina F. Dorneles	13
Carlos A. Heuser	13
Denis Lima	7
Dorotéia Pitombeira	43
Duncan D.A.Ruiz	49
Eduardo Kroth	1
Eugênio O. Simonetto	49
Fábio Coutinho	25
Fábio Porto	25
Fernanda Baião	31
Fernando Cordeiro	7
Lineu Santos	7
Márcio T. Nazari	49
Marcos Antonio Vaz Salles	37
Marta Mattoso	31
Matheus Wildemberg	31
Melise M. V. Paula	31
Nabor das Chagas Mendonça	43
Nina Edelweiss	55
Renata de Matos Galante	55
Sérgio Lifschitz	37
Sérgio Mergen	1
Vânia Vidal	7
Vincent Nelson Kellers da Silveira	55
Wainberg Oliveira	7