



SBBD 2007

XXII Simpósio Brasileiro de Banco de Dados

15 a 19 de outubro de 2007

João Pessoa • Paraíba • Brasil

IV Sessão de Demos

ANAIS

Coordenadora da Sessão de Demos

Cristina Dutra de Aguiar Ciferri

Coordenadores

Valéria Gonçalves Soares

Glêdson Elias

Realização

DI • Departamento de Informática

UFPB • Universidade Federal da Paraíba

Promoção

SBC • Sociedade Brasileira de Computação

ACM • Association for Computing Machinery

SIGMOD • Special Interest Group on Management of Data

VLBD Endowment • Very Large Data Base Endowment

S612q	<p>Simpósio Brasileiro de Banco de Dados (22 : João Pessoa : 2007)</p> <p>IV Sessão de Demos: Anais / Coordenadora da Sessão de Demos: Cristina Dutra de Aguiar Ciferri; Coordenadores: Valéria Gonçalves Soares, Glêdson Elias – João Pessoa : Sociedade Brasileira de Computação, 2007.</p> <p>70 p. : l.</p> <p>ISBN: 978-85-7669-141-9</p> <p>1. Banco de Dados. I. Ciferri, Cristina Dutra de Aguiar – Coord. da Sessão de Demos. II. Soares, Valéria Gonçalves – Coord. III. Elias, Glêdson – Coord.</p>
UFPB/BC	CDU: 004

Apresentação

A Sessão de Demos, em sua quarta edição, demonstra sua consolidação como um evento satélite ao Simpósio Brasileiro de Banco de Dados (SBBD). O objetivo básico da Sessão de Demos é proporcionar um fórum para intercâmbio de experiências e de soluções automatizadas entre pesquisadores e desenvolvedores da área de Banco de Dados no Brasil. Portanto, a Sessão de Demos apresenta um foco prioritário em produtos e protótipos na área de Banco de Dados.

A Sessão de Demos 2007 recebeu um total de 26 submissões. Deste total, foram selecionadas 11 ferramentas para participar do processo de escolha das três melhores ferramentas. Mantendo as estatísticas passadas, mais de 70% dos artigos enviados estão diretamente relacionados a trabalhos de mestrado e doutorado, bem como a projetos de pesquisa. Vale ressaltar que os artigos submetidos encontram-se, em sua grande maioria, dentro das grandes áreas de Banco de Dados.

O processo de avaliação das ferramentas é realizado em duas etapas. Na primeira etapa, os autores enviam artigos descrevendo as principais características de suas ferramentas, para avaliação por parte dos membros do Comitê Avaliador. Na segunda etapa, os artigos selecionados são apresentados, sob a forma de demonstrações *in loco*, durante o evento, quando então são novamente avaliados por membros do Comitê. Esta segunda avaliação visa garantir que as ferramentas selecionadas sejam operacionais e funcionalmente compatíveis com a descrição contida nos artigos avaliados na primeira etapa. Além disso, os resultados das duas avaliações fornecem subsídios para que o Comitê Avaliador possa escolher as três melhores ferramentas. Em sua quarta edição, a apresentação de ferramentas durante o evento consiste das demonstrações práticas dos artigos selecionados pelo Comitê Avaliador. As demonstrações estão distribuídas nos turnos da manhã e da tarde do dia 16 de outubro de 2007. Nesta data, os pôsteres que descrevem as ferramentas selecionadas também são exibidos ao público.

Gostaria de agradecer ao Comitê Avaliador pelo excelente trabalho realizado e pelo atendimento aos prazos. Agradeço ainda ao Comitê Diretivo do SBBD e, especialmente ao Prof. Altigran Soares da Silva (UFAM), que apoiaram a publicação dos anais impressos para a Sessão de Demos.

Finalmente, desejo a todos os participantes um evento excelente!

João Pessoa, Outubro de 2007

Cristina Dutra de Aguiar Cifferri, USP/São Carlos
Coordenadora da Sessão de Demos do SBBD 2007

Biografia



Cristina Dutra de Aguiar Ciferri. É docente do Departamento de Ciências de Computação da Universidade de São Paulo/Campus de São Carlos desde abril de 2005. Obteve seu título de doutor em Ciência da Computação pela Universidade Federal de Pernambuco em 2002, na área de concentração de banco de dados. Entre 1996 e 2005 foi professora do Departamento de Informática da Universidade Estadual de Maringá, onde participou de diversos projetos de pesquisa financiados pelo CNPq, MCT e Fundação Araucária. Atualmente, Profa. Cristina é responsável pelas disciplinas da área de Banco de Dados, tem vários artigos publicados e participa de diversos projetos de pesquisa dentre os quais destacam-se o projeto FragDW (Fragmentação dos Dados em Ambientes de Data Warehousing), o projeto MIRVisIM (Mineração, Indexação, Recuperação e Visualização de Dados em Sistemas de Arquivamento de Imagens Médicas) financiado pela FAPESP, e o projeto Urano (Integração de Dados XML de Currículos de Docentes em um Banco de Dados Relacional) financiado pela CODAGE/USP. Suas principais áreas de interesse são data warehousing, bancos de dados heterogêneos, bancos de dados distribuídos e sistemas de informação geográfica.

Comitês

Coordenação Geral SBBD-SBES

Valéria Gonçalves Soares, UFPB, BR
Glêdson Elias, UFPB, BR

Comitê Diretivo SBBD

Altigran Silva, UFAM, BR
Carlos Heuser, UFRGS, BR
Mario Nascimento, University of Alberta, CA
Sergio Lifschitz, PUC-Rio, BR
Sandra de Amo, UFU, BR
Caetano Traina Junior, USP, BR

Coordenação do Comitê de Programa SBBD

Altigran Silva, UFAM, BR

Coordenação da Sessão de Demos

Cristina Dutra Aguiar Ciferri, USP, BR

Comitê de Avaliação da Sessão de Demos

Bernadette Faria Lóscio, UFC, BR
Caetano Traina Júnior, USP-São Carlos, BR
Carina Friedrich Dorneles, UPF, BR
Carmem Satie Hara, UFPR, BR
Cristina Dutra Aguiar Ciferri, USP, BR
João Eduardo Ferreira, IME-USP, BR
Marcelo Finger, IME-USP, BR
Marco Antônio Casanova, PUC-Rio, BR
Marilde Terezinha Prado Santos, UFSCar, BR
Nina Edelweiss, UFRGS, BR
Renata de Matos Galante, UFRGS, BR
Renato Fileto, INE-UFSC, BR
Sérgio Lifschitz, PUC-Rio, BR
Valéria Cesário Times, UFPE, BR
Vânia Maria Ponte Vidal, UFC, BR

Avaliadores Externos da Sessão de Demos

Deise de Brum Saccol, UFRGS, BR
Mirella Moura Moro, UFRGS, BR

Comitê de Organização

Coordenação Geral SBBB

Valéria Gonçalves Soares, UFPB, BR

Coordenação Geral SBES

Glêdson Elias, UFPB, BR

Coordenação de Publicações

Gustavo Cavalcanti, UFPB, BR

Coordenação de Infra-Estrutura

Daniel Charles Ferreira Porto, UFPB, BR

Coordenação de Inscrições

Mateus Novaes, UFPB, BR

Coordenação de Recursos Humanos

João Paulo Freitas de Oliveira, UFPB, BR

Membros

Aluísio Bruno Ataíde Lima, UFPB, BR

Amilcar Soares Júnior, UFPB, BR

André Lira Rolim, UFPB, BR

Bruno Neiva Moreno, UFPB, BR

Eudisley Gomes, UFPB, BR

Gregório Enrico Linhares de Melo, UFPB, BR

Herbet Ferreira Rodrigues, UFPB, BR

Josey Wales Diniz Belmont, UFPB, BR

Júlio César Ferreira da Silva, UFPB, BR

Kelly Christinne Fernandes de Souza, UFPB, BR

Manuella Dias Carvalho Silva, UFPB, BR

Marcela Balbino Santos de Moraes, UFPB, BR

Mariana Meirelles de M. Lula, UFPB, BR

Roberta Macêdo Marques Gouveia, UFPB, BR

Sebastião Estefâncio Pinto Rabelo Júnior, UFPB, BR

Talles Brito Viana, UFPB, BR

Vítor Márcio Paiva de Sousa Baptista, UFPB, BR

Walber José Adriano Silva, UFPB, BR

Índice

Ferramentas

<i>Simulador de Rede de Sensores para Distribuição de Consultas Contínuas</i>	03
Diorgens Meira (UNIFOR), Ângelo Brayner (UNIFOR), Aretusa Lopes (UNIFOR), Ronaldo Menezes (Florida Institute of Technology)	
<i>FOQuE: Um Sistema para Expansão Semântica de Consultas baseada em Ontologias Difusas</i>	09
Cristiane A. Yaguinuma (UFSCar), Victor S. Faria (UFSCar), Marilde T. P. Santos (UFSCar), Mauro Biajiz (UFSCar)	
<i>RecS-DL: Uma Plataforma para Serviços de Recomendação em Bibliotecas Digitais</i>	15
Daniel Carlos Guimarães Pedronette (UNICAMP), Ricardo da Silva Torres (UNICAMP)	
<i>Um Mediador para o Processamento de Consultas sobre Bases XML Distribuídas</i>	21
Guilherme Figueiredo (COPPE-UFRJ), Vanessa Braganholo (IM-UFRJ), Marta Mattoso (COPPE-UFRJ)	
<i>NanoBase: Um Processador de Consultas para a Plataforma JME/CLDC</i>	27
Leonardo Eloy (UNIFOR), Vitor Araújo (UNIFOR), José Maria Monteiro (UNIFOR), Ângelo Brayner (UNIFOR)	
<i>SimEval: Uma Ferramenta para Avaliação de Qualidade para Funções de Similaridade</i>	33
Francisco N. A. Krieser (UFRGS), Carlos Alberto Heuser (UFRGS), Viviane Moreira Orenço (UFRGS)	
<i>Litebase: Um Gerenciador de Banco de Dados para PDAs com Índices Baseados em Árvores-B</i>	39
Guilherme C. Hazan (SuperWaba), Renato L. Novais (PUC-Rio), Sérgio Lifschitz (PUC-Rio)	
<i>5SQual: A Quality Assessment Tool for Digital Libraries</i>	45
Bárbara L. Moreira (UFMG), Marcos André Gonçalves (UFMG), Alberto H. F. Laender (UFMG), Edward A. Fox (Virginia Tech)	
<i>WhizKEY: A Wizard-Based Tool for Installing Digital Libraries</i>	51
Rodrygo L. T. Santos (UFMG), Marcos André Gonçalves (UFMG), Alberto H. F. Laender (UFMG)	
<i>XVBA: A Tool for Semi-Automatic Generation of SQL/XML Views</i>	57
Vânia Maria Ponte Vidal (UFC), Fernando Cordeiro Lemos (UFC)	
<i>GeoDWCASE: Uma Ferramenta para Projeto de Data Warehouses Geográficos</i>	63
Rafael Fonseca (UFPE), Robson Fidalgo (UFPE), Joel da Silva (UFPE), Valéria Times (UFPE)	
Índice por Autor	69

SBBB 2007
IV Sessão de Demos



Ferramentas

SBBD 2007
IV Sessão de Demos

Simulador de Rede de Sensores para Distribuição de Consultas Contínuas

Diorgens Meira¹, Ângelo Brayner¹, Aretusa Lopes¹, Ronaldo Menezes²

¹ Mestrado em Informática Aplicada - Universidade de Fortaleza (UNIFOR) – Brasil.

² Florida Institute of Technology - USA.

dmm@unifor.edu.br, brayner@unifor.br, aretusaal@unifor.edu.br,
rmenezes@cs.fit.edu

Resumo. *Este artigo apresenta a implementação de um simulador de uma Rede de Sensores (SNetSim), visando operacionalizar um cenário virtual para execução de um motor de distribuição de consultas contínuas. O propósito deste projeto é auxiliar pesquisadores da área de Redes de Sensores sem Fio (RSSF) no desenvolvimento de seus trabalhos, ao disponibilizar um pacote de funções básicas relacionadas com o gerenciamento e a distribuição das consultas e permitir extensões customizáveis sobre estas.*

1. Introdução

A simulação é uma importante aliada na busca de conhecimentos, tanto dos problemas abordados durante uma pesquisa científica, quanto de suas soluções. Quanto mais aderente ou específico é o ambiente em relação ao escopo do trabalho, maiores serão as condições para ajudar os desenvolvedores que o utilizam. A viabilidade e o funcionamento sobre diferentes perspectivas e cargas de sistemas podem ser observados através de simulação. Em [3] há um bom exemplo de um arcabouço utilizado na simulação de uma RSSF e voltado especificamente para prova de conceito de uma arquitetura para gerenciamento deste tipo de rede.

Neste trabalho apresentamos o *SNetSim*, que é a implementação de um conjunto de aplicações que compõem um simulador de Rede de Sensores sem Fio (RSSF), desenvolvido para ser uma ferramenta de auxílio em implementações de estratégias de consultas sobre esse tipo de plataforma. Este simulador não pretende englobar todos os aspectos necessários existentes num modelo de RSSF real[1], mas apenas dar o suporte mínimo para o desenvolvimento de estratégias de processamento e distribuição de consultas contínuas, e facilitar a sua implementação, ao ocultar detalhes inerentes a protocolos de comunicação e controle do fluxo entre os nós envolvidos.

O presente documento está estruturado conforme descrito a seguir: a seção 2 apresenta o sumário da arquitetura do simulador de rede de sensores *SNetSim*, a categorização de seus módulos e o resumo das características do mecanismo de processador de consultas implementado. A seção 3 apresenta alguns trabalhos relacionados, e a seção 4 conclui este artigo e referencia futuros desenvolvimentos relacionados com os tópicos aqui tratados.

2. Simulador da Rede de Sensores (*SNetSim*)

A implementação do simulador de redes de sensores *SNetSim* seguiu as premissas do processador de consultas descrito no tópico 2.3 deste artigo, quando foi possível criar um cenário mínimo de testes para provas de conceito para o que esse motor propõe.

O ponto de partida para a concepção deste sistema foi a criação do ambiente que simulasse uma Rede de Sensores sem Fio (RSSF) usando como plataforma nativa estações de uma rede local baseada nos protocolos TCP/IP.

Um exemplo de cenário simulado de monitoramento em uma RSSF consiste de uma rede virtual para sensoriamento de umidade e temperatura numa região geográfica, tal como visto na Figura 1.

A simulação proposta consiste na submissão de uma consulta a partir de um nó central (estação base) que a distribuirá aos nós sensores conforme o grupo sensor (ou região) do qual fazem parte, até que os pacotes da consulta alcancem todos os nós da rede. Esta distribuição deverá considerar as questões relacionadas com o tipo de sensoriamento que cada nó estiver apto a realizar, ou seja, somente processará solicitações de consultas relacionadas com o tipo daquele nó.

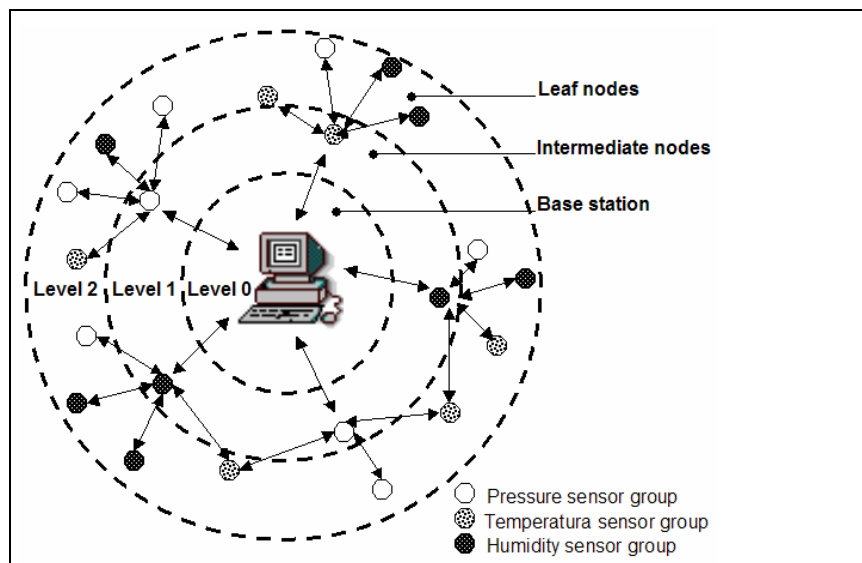


Figura 1 – Diagrama simplificado de uma RSSF para o cenário a ser simulado.

2.1. Categorização das estações de trabalho

O *SNetSim* é composto por 3 (três) tipos de estação de trabalho, cada um com o seu papel bastante específico dentro desta arquitetura: (i) Estação base; (ii) Agente de Rede; (iii) Estação sensores. Cada uma dessas categorias pode ser materializada dentro de um ou mais microcomputadores executando os módulos de serviços (aplicativos executáveis) relativos a cada tipo de estação a ser operacionalizada. Para que uma simulação ganhe em termos de escalabilidade, as estações podem ser instaladas em mais de um computador que participem de uma rede. Assim, é transparente para a aplicação

SBBD 2007

IV Sessão de Demos

que utilizará a *SNetSim* se os nós virtuais estão dispostos localmente (na mesma máquina), ou disseminados através de uma rede real compatível com o protocolo nativo utilizado. O papel de cada tipo de estação *SNetSim* é descrito a seguir:

- **Estação base:** concentrará os módulos relativos ao nó central da rede, tal como a interface de gerenciamento na qual o usuário submete consultas SNQL (módulo executável *Base Station Manager*, Figura 2), e o processamento da consolidação (módulo executável *Base Station Processor*) dos resultados advindos das aquisições nos nós sensores distribuídos. Os módulos da estação base podem ser instalados em máquinas distintas dentro rede real;

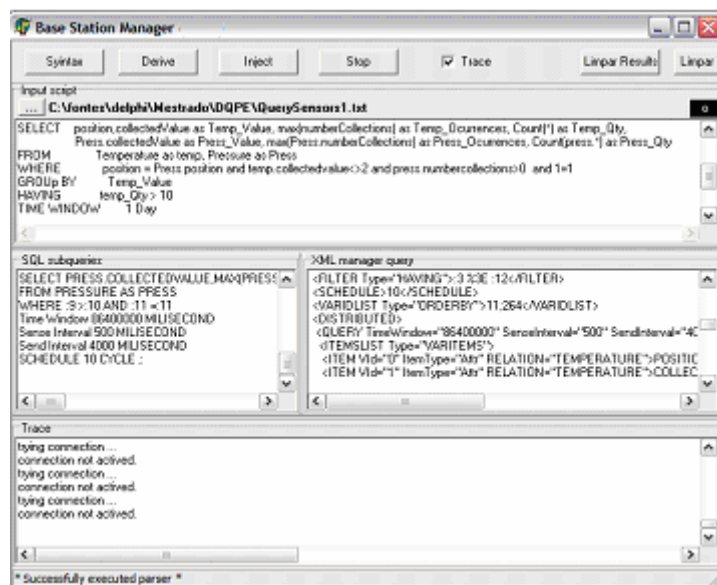


Figura 2 – Tela do módulo *Base Station Manager* em execução.

- **Network Agent:** módulo executável que representa o núcleo da rede (Figura 3), simulando o meio físico pelo qual as comunicações entre os módulos trafegam (nuvem da rede). Esta estação é responsável por manter os canais de comunicação entre os nós virtuais da RSSF, simulando *broadcast* como uma estação de rádio, a partir dos pacotes enviados pelas demais estações;

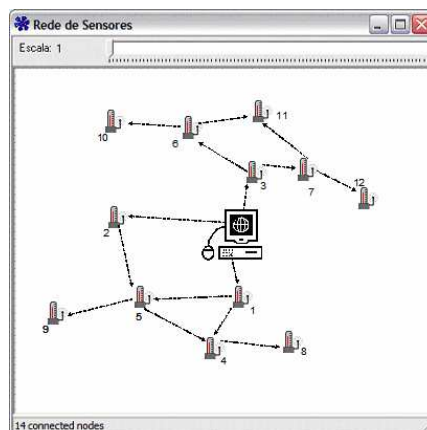
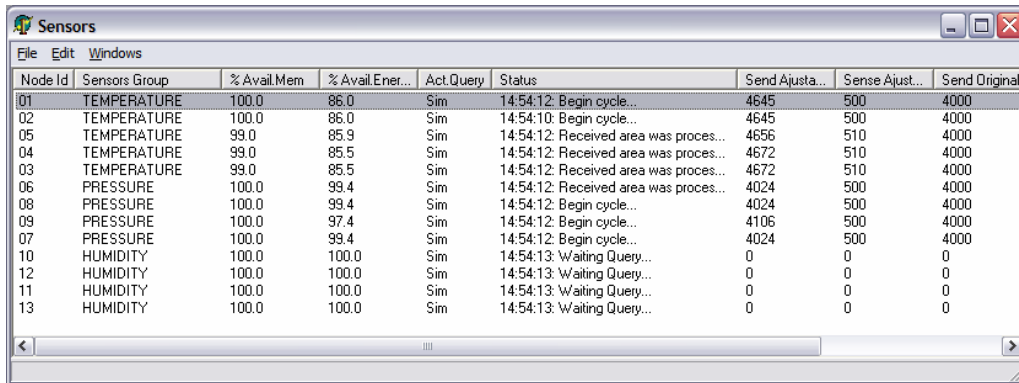


Figura 3 – Tela de rastreamento gráfico do *Network Agent*.

SBBD 2007

IV Sessão de Demos

- **Estação Sensores:** Estação contendo os módulos que implementam os nós sensores distribuídos pela rede (Figura 4). Executará o módulo *Sensor Node Processor*, responsável pela simulação de sensores de diversas categorias. Pode haver mais de uma estação de sensores na rede real, podendo cada uma simular um ou mais nós. Cada sensor simulado, ao ser criado, terá uma identificação única, gerada automaticamente pela *SNetSim*.



Node Id	Sensors Group	% Avail Mem	% Avail Ener...	Act Query	Status	Send Ajusta...	Sense Ajust...	Send Original
01	TEMPERATURE	100.0	86.0	Sim	14:54:12: Begin cycle...	4645	500	4000
02	TEMPERATURE	100.0	86.0	Sim	14:54:10: Begin cycle...	4645	500	4000
05	TEMPERATURE	99.0	85.9	Sim	14:54:12: Received area was proces...	4656	510	4000
04	TEMPERATURE	99.0	85.5	Sim	14:54:12: Received area was proces...	4672	510	4000
03	TEMPERATURE	99.0	85.5	Sim	14:54:12: Received area was proces...	4672	510	4000
06	PRESSURE	100.0	99.4	Sim	14:54:12: Received area was proces...	4024	500	4000
08	PRESSURE	100.0	99.4	Sim	14:54:12: Begin cycle...	4024	500	4000
09	PRESSURE	100.0	97.4	Sim	14:54:12: Begin cycle...	4106	500	4000
07	PRESSURE	100.0	99.4	Sim	14:54:12: Begin cycle...	4024	500	4000
10	HUMIDITY	100.0	100.0	Sim	14:54:13: Waiting Query...	0	0	0
12	HUMIDITY	100.0	100.0	Sim	14:54:13: Waiting Query...	0	0	0
11	HUMIDITY	100.0	100.0	Sim	14:54:13: Waiting Query...	0	0	0
13	HUMIDITY	100.0	100.0	Sim	14:54:13: Waiting Query...	0	0	0

Figura 4 – Lista de sensores simulados em execução.

2.2. Gerenciamento dos esquemas

A estratégia adotada pelo Simulador de Redes de Sensores, implementando a filosofia do mecanismo PDCRS (Processamento e Distribuição de Consultas em Redes de Sensores) descrito na sub-seção 2.3, provê recursos para manutenção dos esquemas das relações de sensores, definidas dentro do contexto adequado para as consultas, a serem demandadas por aplicações usuárias. A raiz do gerenciamento dá-se na definição do esquema global de relações, que é conceitualmente o seu banco de dados.

O gerenciamento do esquema das relações envolvidas (mapeamento dos grupos de sensores que terão suas saídas invocadas nas consultas de usuários) se inicia com a criação (definição e persistência) dessas relações a partir do simulador, tal como se segue: i) criação do esquema global; ii) definição dos grupos de sensores; e iii) definição das relações e atributos associados aos resultados gerados.

2.3. Processamento e Distribuição de Consultas em Redes de Sensores (PDCRS)

O PDCRS é o mecanismo do processador de consultas implementado no *SNetSim*, e é responsável pelo gerenciamento de esquemas, avaliação e decomposição dessas. Este motor, apresentado na forma de uma biblioteca de classes reutilizáveis, controla o fluxo de execução da aplicação a partir da chamada de métodos especializados. O Simulador *SNetSim* é composto por aplicativos que se interagem através das classes do PDCRS.

A Figura 5 mostra um modelo abstrato de processamento de consultas na RSSF implementada no simulador, no qual a consulta é processada por quatro fases distintas, executadas no nó central dessa rede: Interpretação (*parsing* da linguagem declarativa *SQL-Like*), Decomposição, Distribuição dos fragmentos e Pós-processamento da consulta.

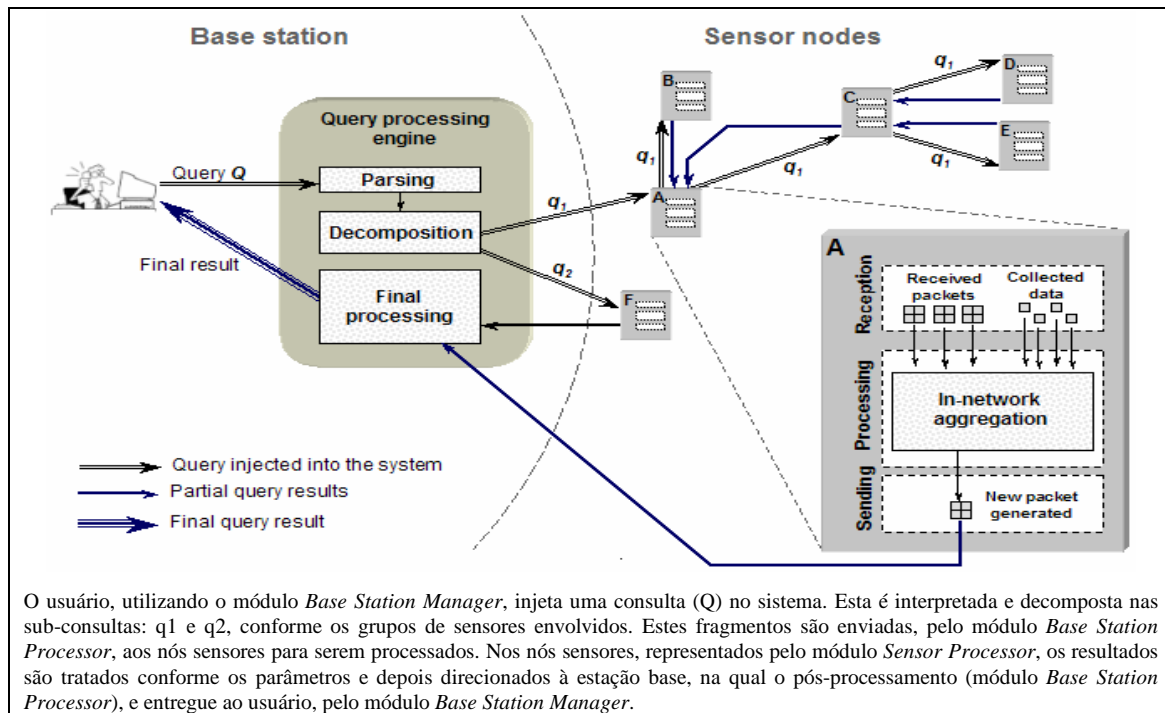


Figura 5 – Modelo de processamento de consultas numa RSSF.

As consultas submetidas ao simulador são escritas numa linguagem *SQL-Like*, denominada SNQL (*Sensor Network Query Language*), conforme sumarizado na Seção 3, a seguir.

2.4. SNQL – Sensor Network Query Language

A SNQL (*Sensor Network Query Language*) é uma linguagem declarativa *SQL-Like* acrescida de cláusulas específicas para processamento de consultas (DML – *Data Manipulation Language*) e definição de dados (DDL – *Data Definition Language*) em redes de sensores. Esta linguagem foi proposta por nosso grupo de pesquisa durante a definição do mecanismo de processamento de consultas PDCRS, e adaptou alguns conceitos utilizados em [7].

Cláusulas comuns ao padrão SQL, como *Select*, *From*, *Where*, *Group By* e *Having*, foram mantidas, enquanto que outras, especialmente adequadas às necessidades do processamento de consultas em RSSF, foram incluídas. Tais cláusulas especiais dão flexibilidade à aplicação ao permitir que sejam definidos intervalos de tempo para captação, empacotamento e envio dos dados, especificação da validade e a frequência de submissão da consulta. Um dessas sumário dessas cláusulas pode ser visto em [4].

3. Trabalhos relacionados

Sobre simulação de RSSF, encontramos em [3] um arcabouço específico para simulação de RSSF, chamado *MANASim*, que permite a simulação de diferentes configurações e características para este tipo de rede e os elementos que a compõe. Esse trabalho é composto por um conjunto de classes base para a simulação de RSSF, as quais podem ser especializadas pelos pesquisadores usuários, estendendo as funcionalidades do simulador *Network Simulator 2* (NS-2) [6], uma ferramenta para simulação de redes

bastante utilizada no meio científico e acadêmico, que ainda não se adaptou à tecnologia das RSSF.

4. Conclusão e trabalhos futuros

Foi apresentado aqui o simulador de redes de sensores para distribuição de consultas contínuas (*SNetSim*). O protótipo implementado considera otimizações relacionadas a agregações em rede [4], gerenciamento de energia, decisões sobre o descarte ou re-encaminhamento de pacotes verificados e implementação de hierarquia de nós dentro da rede, conforme conceitos estudados em [2][8]. Os resultados dos testes realizados demonstraram a eficiência do motor proposto.

Estimamos para versões posteriores da ferramenta aqui apresentada a implementação de medidas relacionadas com a escalabilidade da rede, tal como controles nativos para níveis hierárquicos de nós com gerenciamento distribuído. A aplicação de simulação da arquitetura da rede será acrescida de módulos controle de potência do sinal de transmissão, permitindo assim o desenvolvimento de cenários mais próximos do mundo real.

Referências

1. Akyildiz, I., Su, W., Sankarasubramaniam, Y. And Cyirci, E. (2002) “Wireless sensor networks: A survey”. *Computer Networks*, vol. 38, no. 4, pp. 393-422.
2. Babu, S., Motwani, R., Babcock, B., Datar, M. (2002) “Models and issues in data stream systems”. *ACM PODS*. June.
3. Braga, T., Silva, F., Beatrys, L., Nogueira, J. (2004) “MannaSim: Um Arcabouço para Simulação de Redes de Sensores Sem Fio”. *REIC - Revista eletrônica de iniciação científica*, v. 4, n. 3.
4. Brayner, A., Lopes, A., Meira, D., Vasconcelos, R., Menezes, R. (2006) “ADAGA: ADaptive AGgregation Algorithm for Sensor Networks”. *XXI SBBD 2006 – Simpósio Brasileiro de Banco de Dados*.
5. Chatterjea, S., Havinga, P. (2004) “A Framework for a Distributed and Adaptive Query Processing Engine for Wireless Sensor Networks”. *INSS*.
6. Fall, K., Varadhan, K. (2006) “The NS Manual”. *The VINT Project*, June, 5.
7. Madden, Samuel R., Franklin, Michael J. and Hellerstein, Joseph M. (2005) “TinyDB: An Acquisitional Query Processing System for Sensor Networks”. *ACM Transactions on Database Systems*, Vol. 30, No. 1, Pages 122-173. March.
8. Rentala, P., Musunuri, R., Gandham, S. and Saxena, U. (2001) “Survey on Sensor Networks”. *Technical Report, UTDCS-33-02, University of Texas at Dallas*.

FOQuE: um sistema para expansão semântica de consultas baseada em ontologias difusas

Cristiane A. Yaguinuma, Victor S. Faria, Marilde T. P. Santos, Mauro Biajiz

Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
Caixa Postal 676 – 13565-905 – São Carlos – SP – Brasil

{cristiane_yaguinuma, marilde, mauro}@dc.ufscar.br
victor_faria@comp.ufscar.br

Resumo. *Muitas vezes o conhecimento do usuário sobre um determinado domínio de aplicação não corresponde exatamente à forma como os dados estão representados e, portanto, pode não existir uma resposta precisa para as consultas. Dentro deste contexto, este artigo apresenta o sistema FOQuE, que utiliza ontologias difusas para recuperar resultados aproximados semanticamente relevantes de acordo com parâmetros de expansão definidos pelo usuário. As respostas adicionais são classificadas conforme o tipo de expansão realizada e a relevância para a consulta, permitindo, assim, que o usuário identifique quais os resultados aproximados mais apropriados para seus requisitos.*

1. Introdução

Diante da oferta abundante de dados pertencentes a diversas áreas do conhecimento, cada vez mais são necessárias estratégias para busca e recuperação de informação que reduzam as respostas irrelevantes e não desprezem resultados relevantes. Em muitos casos, o usuário especifica termos de consulta que não correspondem exatamente aos termos que representam os dados a serem recuperados, sendo necessário obter aproximações que satisfaçam às consultas. Neste sentido, a expansão de consultas visa modificar a consulta do usuário (consulta original), seja pela alteração de suas condições ou pela adição de termos relacionados, com o intuito de tornar as consultas mais eficazes e abrangentes.

Algumas abordagens de expansão de consultas utilizam ontologias e mecanismos de inferência para derivar novas informações e recuperar respostas semanticamente relevantes. Entretanto, como utilizam ontologias tradicionais (*crisp*), essas abordagens não são capazes de representar nem raciocinar sobre informações imprecisas como a intensidade de relacionamentos entre conceitos ou o grau de pertinência de conceitos a classes. Assim, ontologias difusas têm sido utilizadas com o objetivo de aumentar a expressividade da representação de domínio e possibilitar a obtenção de respostas aproximadas com base em conceitos e relacionamentos difusos.

Diante deste contexto, este artigo apresenta o sistema FOQuE (*Fuzzy Ontology-based Query Expansion*), desenvolvido para realizar diversos tipos de expansão semântica com base em ontologias difusas, proporcionando resultados aproximados tanto relevantes quanto abrangentes para a consulta do usuário. O restante deste artigo está organizado da seguinte forma: a seção 2 apresenta uma discussão sobre trabalhos relacionados; a seção 3 descreve uma visão geral sobre a arquitetura e os tipos de expansão de consulta do sistema FOQuE; a seção 4 ilustra o protótipo desenvolvido e a seção 5 finaliza o artigo com as conclusões e os trabalhos futuros.

2. Trabalhos relacionados

Em geral, as estratégias para expansão de consultas baseada em ontologias difusas consideram principalmente taxonomias difusas e similaridade entre conceitos. Parry [Parry 2004] e o sistema PASS [Widyantoro and Yen 2001] reformulam consultas com base em taxonomias contendo graus de pertinência que representam o quanto um conceito é mais específico que outro. Os sistemas MIEL [Buche et al. 2005] e *Corese* [Corby et al. 2006] consideram tanto taxonomias difusas quanto relacionamentos de similaridade para expandir consultas conforme as preferências do usuário. Enquanto Parry [Parry 2004] se dedica exclusivamente a tratar conceitos homônimos, existem trabalhos que sequer mencionam como é feito esse tratamento como [Widyantoro and Yen 2001] [Buche et al. 2005] [Corby et al. 2006], fato que pode comprometer tanto a revocação quanto a precisão dos resultados. Deste modo, é interessante um sistema de expansão de consultas abrangente, capaz de efetuar diversos tipos de expansão, considerando também construções semânticas como conceitos homônimos, transitividade difusa e relacionamentos todo-parte, para permitir a recuperação de respostas relevantes que não seriam obtidas pelas consultas originais. Identificados esses aspectos, a próxima seção apresenta o sistema FOQuE, que expande consultas considerando não somente taxonomias difusas e similaridade mas também outros tipos de expansão, como proximidade todo-parte e transitividade difusa, além de tratar consultas que contenham conceitos homônimos.

3. Sistema FOQuE

O objetivo do sistema FOQuE é expandir consultas com base na análise de ontologias difusas para obter resultados aproximados que sejam semanticamente relacionados aos termos especificados pelo usuário em consultas. A Figura 1 apresenta a arquitetura geral do sistema, ilustrando seus principais módulos, a ontologia difusa e o mapeamento entre a ontologia e o banco de dados.

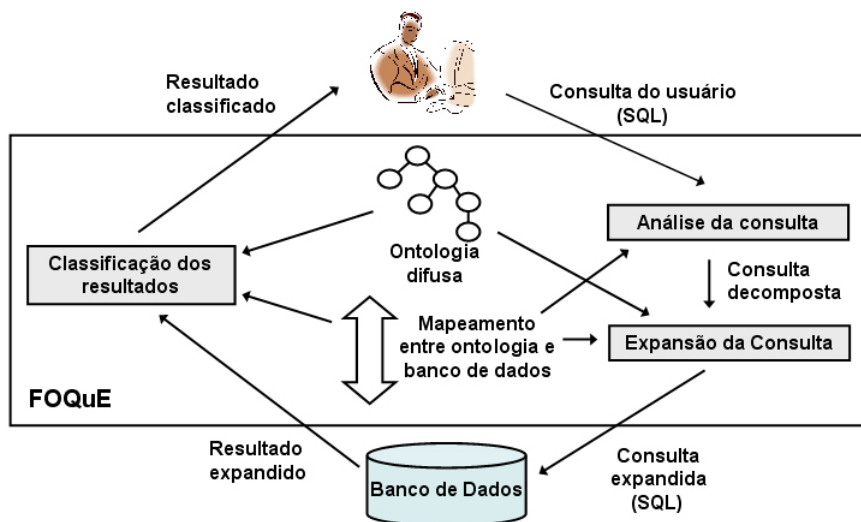


Figura 1. Arquitetura do sistema FOQuE.

O sistema FOQuE recebe a consulta especificada na linguagem SQL que, por ser a linguagem padrão para consultas a Sistemas Gerenciadores de Bancos de Dados Relacionais (SGBDR), evita que consultas tenham que ser adaptadas para a expansão. O sistema também recebe parâmetros para controlar as expansões: similaridade mínima (*minSimilarity*), pertinência mínima (*minMembership*) para expansões que envolvem graus de per-

tinência, e proximidade mínima (*minCloseness*) para a expansão por proximidade toda-parce. A consulta é então decomposta pelo módulo *Análise da consulta* para que o módulo *Expansão da consulta* seja capaz de identificar quais expansões podem ser aplicadas com base na ontologia difusa e no mapeamento entre a ontologia e o banco de dados. A consulta expandida é especificada em SQL, podendo ser executada por qualquer SGBDR (abordagem não-intrusiva). Por fim, as respostas são classificadas (módulo *Classificação dos resultados*) para exibir os resultados por ordem de relevância de acordo com o tipo de expansão realizada. As próximas sub-seções descrevem características da ontologia difusa e do mapeamento utilizados e os tipos de expansão realizados por FOQuE.

3.1. Ontologia Difusa e Mapeamento

A ontologia difusa utilizada pelo sistema FOQuE representa a semântica dos dados armazenados no banco de dados, ou seja, especifica conceitos, relacionamentos e axiomas do domínio que estão associados às tabelas, atributos e dados pertencentes ao banco. Além disso, a ontologia representa a intensidade de relacionamentos e axiomas, possibilitando o raciocínio sobre conceitos e relacionamentos difusos. Para tanto, são considerados mecanismos de inferência com base em definições da lógica difusa [Dubois and Prade 1980] ($\mu_A(x)$ representa o grau de pertinência da instância x para a classe A e $\mu_R(x, y)$ a intensidade do relacionamento R entre as instâncias x e y):

- Generalização/especialização de conceitos difusos: $A \subseteq B \Leftrightarrow \mu_A(x) \leq \mu_B(x)$;
- Propriedade reflexiva de relacionamentos difusos: $\mu_R(x, x) = 1$;
- Propriedade simétrica de relacionamentos difusos: $\mu_R(x, y) = \mu_R(y, x)$;
- Transitividade max-min: $\mu_R(x, z) = \max_{y \in X} \min[\mu_R(x, y), \mu_R(y, z)]$.

Para permitir a expansão de consultas, o mapeamento define como os elementos do banco de dados estão associados aos elementos da ontologia difusa. No sistema FOQuE, o mapeamento é composto por regras *antecedente* \Rightarrow *consequente*, onde *antecedente* contém elementos da ontologia difusa e *consequente* elementos do banco de dados. Um exemplo de mapeamento é a regra $Produto(x), label(x, y) \Rightarrow Produto(x), nome(x, y)$, que descreve que instâncias da classe *Produto* com um determinado rótulo (*label*) correspondem a dados da tabela *Produto* cujo atributo *nome* possui o mesmo valor que *label*. A partir dessas regras, é possível especificar quais tabelas e atributos correspondem a classes ou relacionamentos da ontologia e quais dados estão associados a conceitos homônimos. Vale ressaltar que nem todos os elementos do banco são associados aos elementos da ontologia, pois são considerados somente aqueles que contribuem para a expansão semântica das consultas. Por exemplo, atributos que representam componentes (partes) de produtos são mapeados porque podem ser expandidos por transitividade, enquanto atributos como código ou senha não são mapeados por não serem relevantes ao processo de expansão semântica.

3.2. Expansão de consultas

A partir da ontologia difusa e do mapeamento, o sistema FOQuE é capaz de realizar os seguintes tipos de expansão de consultas:

Homônimos: identifica e apresenta ao usuário os diferentes contextos presentes em consultas contendo conceitos homônimos. Para cada conceito identificado, são mostrados os respectivos caminhos na hierarquia de classes da ontologia, permitindo que o usuário escolha qual o mais apropriado. O mapeamento é então utilizado para gerar a consulta correspondente ao contexto escolhido.

Classes: expande consultas contendo termos que correspondem a classes da ontologia difusa, através da análise de graus de pertinência que representam o quanto um conceito pertence a uma determinada classe. As instâncias da classe especificada na consulta são adicionadas à consulta expandida, desde que possuam grau de pertinência maior que *minMembership*, para permitir a recuperação de respostas aproximadas pela especialização de conceitos difusos.

Similaridade: modifica a consulta ao adicionar conceitos similares ao conceito especificado pelo usuário, desde que satisfaçam ao parâmetro *minSimilarity*. Para tanto, FOQuE analisa os relacionamentos de similaridade da ontologia difusa, sendo capaz de inferir graus de similaridade segundo a transitividade max-min.

Proximidade todo-parte: considera que conceitos constituídos por um conjunto aproximado de partes em comum podem ser semanticamente próximos entre si. Para tanto, FOQuE verifica relacionamentos todo-parte difusos da ontologia, cujos graus de pertinência representam a intensidade com a qual uma parte é relevante para o todo. Assim, é possível expandir a consulta considerando conceitos cujos graus de proximidade satisfaçam ao parâmetro *minCloseness*. O cálculo dos graus de proximidade todo-parte é descrito com mais detalhes em [Yaguinuma et al. 2007].

Transitividade difusa: analisa relacionamentos transitivos da ontologia que não são explícitos no banco de dados. O conceito especificado na consulta é verificado na ontologia para identificar quantos níveis de transitividade ele possui para o relacionamento transitivo referenciado na consulta. FOQuE também analisa se os conceitos relacionados por transitividade satisfazem ao parâmetro *minMembership*. A consulta original é então modificada pela adição de novas condições que possibilitam recuperar os dados envolvidos nos diversos níveis de transitividade identificados.

4. Protótipo desenvolvido

Para avaliar a aplicabilidade do sistema FOQuE, implementou-se um protótipo do sistema como uma API (*Application Programming Interface*) usando tecnologia Java e o *framework* Jena [Jena 2006], que serviu de base para construir o mapeamento e os mecanismos de inferência sobre ontologias difusas codificadas na linguagem OWL DL. Com relação ao banco de dados, utilizou-se o SGBDR PostgreSQL, porém qualquer outro SGBDR poderia ser utilizado, desde que possuía um *driver* JDBC. A partir da API construída, foram desenvolvidas uma aplicação *desktop* e uma aplicação *web*, sendo esta última apresentada neste artigo. Testes do protótipo consideraram dois domínios, um sobre produtos de supermercado e outro que descreve disciplinas de computação, suas áreas, requisitos e ementas (domínio abordado neste artigo). Para cada domínio, o banco de dados, a ontologia e os mapeamentos foram especificados como parâmetros do sistema FOQuE, pois este foi construído de forma independente do domínio de aplicação.

Para o domínio de disciplinas da computação, foram utilizados dados reais de disciplinas dos cursos de computação da UFSCar e da USP São Carlos, além de informações sobre alunos de computação da UFSCar. A ontologia difusa e os mapeamentos foram construídos manualmente com base nas Diretrizes Curriculares de Cursos da Área de Computação e Informática [MEC 1999], no relatório de avaliação do MEC para o curso de Bacharelado em Ciência da Computação da UFSCar [UFSCar 2000] e nas ementas das disciplinas consideradas, com o auxílio dos professores de computação da UFSCar. Para ilustrar exemplos de expansões nesse domínio, considere uma consulta pelas notas de um

SBBD 2007 IV Sessão de Demos

aluno em todos pré-requisitos da disciplina *Construção de compiladores*, com parâmetros $minMembership = 0.25$, $minSimilarity = 1.0$ e $minCloseness = 0.2$. Nesta consulta, a disciplina *Laboratório de compiladores* foi recuperada pela expansão por proximidade todo-parte (Figura 2B), a partir da análise dos tópicos (partes) em comum que constituem as ementas das disciplinas. Como os pré-requisitos das disciplinas podem ser transitivos, FOQuE realiza a expansão por transitividade para recuperar os pré-requisitos transitivos de *Construção de compiladores* (Figura 2B), enquanto a consulta original retorna somente as notas dos pré-requisitos diretos (Figura 2A). Também são recuperados todos os requisitos transitivos de *Laboratório de compiladores*, pois a expansão por transitividade considera os conceitos previamente expandidos. Vale ressaltar que os pré-requisitos são ordenados quanto ao grau do relacionamento difuso *ehPreRequisito* da ontologia, em que requisitos recomendados possuem grau de pertinência (*Membership*) dentro do intervalo $[0, 1]$ enquanto requisitos obrigatórios possuem grau máximo (*Membership* = 1). Mais detalhes sobre como são realizadas as expansões por proximidade todo-parte e transitividade difusa podem ser vistos em [Yaguinuma et al. 2007].

Original Result **A**

discip.nome_discip	discipreq.nome_discip	insc.nota_insc
CONSTRUCAO DE COMPILADORES	ESTRUTURAS DE DADOS	6.0
CONSTRUCAO DE COMPILADORES	LINGUAGENS DE PROGRAMACAO	2.0
CONSTRUCAO DE COMPILADORES	LINGUAGENS DE PROGRAMACAO	2.1
CONSTRUCAO DE COMPILADORES	LINGUAGENS DE PROGRAMACAO	6.9
CONSTRUCAO DE COMPILADORES	LINGUAGENS FORMAIS E AUTOMATOS	1.8
CONSTRUCAO DE COMPILADORES	LINGUAGENS FORMAIS E AUTOMATOS	6.9

Result augmented by transitivity to CONSTRUCAO DE COMPILADORES

discip.nome_discip	discipreq.nome_discip	insc.nota_insc	Membership
CONSTRUCAO DE COMPILADORES	PROGRAMACAO DE COMPUTADORES	7.3	1.0
CONSTRUCAO DE COMPILADORES	CONSTRUCAO DE ALGORITMOS	8.6	1.0

Result augmented by transitivity to LABORATORIO DE COMPILADORES **B**

discip.nome_discip	discipreq.nome_discip	insc.nota_insc	Membership
LABORATORIO DE COMPILADORES	ESTRUTURAS DE DADOS	6.0	1.0
LABORATORIO DE COMPILADORES	PROGRAMACAO DE COMPUTADORES	7.3	1.0
LABORATORIO DE COMPILADORES	CONSTRUCAO DE ALGORITMOS	8.6	1.0
LABORATORIO DE COMPILADORES	LINGUAGENS FORMAIS E AUTOMATOS	1.8	0.3
LABORATORIO DE COMPILADORES	LINGUAGENS FORMAIS E AUTOMATOS	6.9	0.3
LABORATORIO DE COMPILADORES	LINGUAGENS DE PROGRAMACAO	6.9	0.25
LABORATORIO DE COMPILADORES	LINGUAGENS DE PROGRAMACAO	2.0	0.25
LABORATORIO DE COMPILADORES	LINGUAGENS DE PROGRAMACAO	2.1	0.25

Figura 2. Resultados expandidos para obter as notas de todos os pré-requisitos da disciplina *Construção de compiladores* considerando disciplinas próximas.

Os resultados da Figura 2 mostram que a expansão por transitividade melhorou principalmente a revocação, pois a consulta original recuperou apenas seis dos oito pré-requisitos (75%) da disciplina *Construção de compiladores*, enquanto a consulta expandida obteve 100% dos pré-requisitos porque considerou também os pré-requisitos transitivos. Além disso, a expansão por proximidade todo-parte permitiu recomendar a disciplina *Laboratório de compiladores* e seus respectivos pré-requisitos, informação que pode ser

interessante para o usuário pois essa disciplina satisfaz aos parâmetros de expansão especificados. Deste modo, além de melhorar a revocação em comparação com a consulta original, FOQuE é capaz de recomendar respostas aproximadas que podem ser relevantes aos requisitos dos usuários.

5. Conclusões e trabalhos futuros

Este artigo apresenta o sistema FOQuE, que recupera resultados aproximados relevantes ao efetuar diversos tipos de expansão semântica de consultas com base em ontologias difusas. Além de construções semânticas como taxonomias difusas e relacionamentos de similaridade, o sistema FOQuE também considera conceitos homônimos, relacionamentos todo-parte e transitividade difusa, contribuindo, assim, para melhorar a revocação das respostas com relação às consultas originais. O sistema FOQuE também recomenda e classifica as respostas aproximadas de acordo com o tipo de expansão e os graus difusos considerados. Desta forma, é possível identificar as expansões realizadas e as respostas adicionais mais relevantes para as consultas.

Quanto aos trabalhos futuros, pretende-se realizar baterias de testes envolvendo outros domínios de aplicação mais complexos. Além disso, como a ontologia e os mapeamentos utilizados pelo sistema FOQuE são especificados manualmente, pretende-se considerar técnicas automáticas ou semi-automáticas para obter ontologias difusas e definir as regras de mapeamento com o banco de dados.

Referências

- Buche, P., Dervin, C., Haemmerle, O., and Thomopoulos, R. (2005). Fuzzy querying of incomplete, imprecise, and heterogeneously structured data in the relational model using ontologies and rules. *IEEE Transactions on Fuzzy Systems*, 13(3):373–383.
- Corby, O., Dieng-Kuntz, R., Faron-Zucker, C., and Gandon, F. (2006). Searching the Semantic Web: Approximate Query Processing Based on Ontologies. *IEEE Intelligent Systems*, 21(1):20–27.
- Dubois, D. and Prade, H. (1980). *Fuzzy Sets and Systems: Theory and Applications*. Academic Press.
- Jena (2006). Jena - A Semantic Web Framework for Java. <<http://jena.sourceforge.net/>>.
- MEC (1999). Diretrizes Curriculares de Cursos da Área de Computação e Informática. <<http://www.inf.ufrgs.br/mec/ceeinf/diretrizes.html>>.
- Parry, D. (2004). Fuzzification of a standard ontology to encourage reuse. In *IEEE International Conference on Information Reuse and Integration (IRI)*, pages 582–587, Las Vegas, NV, USA.
- UFSCar (2000). Relatório de Avaliação do Curso de Bacharelado em Ciência da Computação. Departamento de Computação, Universidade Federal de São Carlos.
- Widiantoro, D. and Yen, J. (2001). A fuzzy ontology-based abstract search engine and its user studies. In *IEEE International Conference on Fuzzy Systems*, pages 1291–1294, Melbourne, Vic., Australia.
- Yaguinuma, C. A., Biajiz, M., and Santos, M. T. P. (2007). Sistema FOQuE para Expansão Semântica de Consultas Baseada em Ontologias Difusas. In *Simpósio Brasileiro de Banco de Dados (a ser publicado)*.

RecS-DL: Uma plataforma para Serviços de Recomendação em Bibliotecas Digitais

Daniel Carlos Guimarães Pedronette¹, Ricardo da Silva Torres¹

¹Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
CEP 13081-970 – Campinas – SP – Brasil

daniel.pedronette@students.ic.unicamp.br, rtorres@ic.unicamp.br

Resumo. *Esta demonstração visa ilustrar o funcionamento de uma plataforma de serviços de recomendação recentemente criada no Instituto de Computação da Unicamp. Baseada em Serviços Web e interfaces bem definidas, a ferramenta possibilita que engines de recomendação, baseados em diferentes técnicas, possam ser desenvolvidos, instalados e testados no arcabouço definido. A modelagem de dados realizada e a utilização de Serviços Web tornam a plataforma independente de domínio de aplicação e portátil sob o ponto de vista tecnológico.*

1. Introdução

Os sistemas de bibliotecas digitais são uma das formas mais avançadas e complexas de sistemas de informação, dado que oferecem um amplo conjunto de serviços, como busca, navegação, recomendação, entre outros, voltados para comunidades de usuários específicas. Há anos, as aplicações de Bibliotecas Digitais apresentam um crescimento vertiginoso, seja quanto ao volume de dados gerenciados, seja quanto a abrangência de domínios de utilização.

Neste cenário, a localização e escolha do conteúdo desejado, assim como a tarefa de manter-se atualizado diante de grandes volumes de informações tornam-se um grande desafio. Dessa forma, o processo de recomendação, já bastante difundido em relações sociais humanas, passa a ter um importante papel nos sistemas de Bibliotecas Digitais [Reategui and Cazella 2005]. Um Sistema de Recomendação é um software que, a partir de uma grande coleção de itens disponíveis, sugere um conjunto desses itens para um usuário de acordo com seus interesses. Dessa forma, os sistemas de recomendação procuram reduzir a sobrecarga de informações e auxiliar o usuário nos processos de escolha de conteúdo.

Dada a amplitude de aplicação e a difusão dos algoritmos de recomendação, as ferramentas existentes apresentaram limitações em relação a diversos aspectos, como:

- **Domínio de aplicação:** as ferramentas são projetadas tendo em vista uma Biblioteca Digital ou domínio específico, impossibilitando ou limitando sua reutilização;
- **Técnicas de recomendação:** os pacotes disponíveis são, em sua maioria, sistemas monolíticos. Os serviços oferecidos não são extensíveis nem configuráveis e geralmente baseiam-se em uma única técnica de recomendação;
- **Tecnologia:** a implementação das ferramentas é realizada em linguagem, plataforma ou interface específica, limitando o acesso de aplicações clientes.

Esta demonstração visa ilustrar o funcionamento de uma plataforma de serviços de recomendação, recentemente criada no Instituto de Computação da Unicamp, que tem como objetivo apresentar soluções para os problemas abordados. Assim a plataforma deve ser flexível sob o aspecto de algoritmos, domínios de aplicação ou tecnologias.

2. Trabalhos Relacionados

Os sistemas de recomendação vêm sendo bastante explorados na literatura desde o início da década de 90. As técnicas existentes podem ser divididas em duas grandes categorias: filtragem colaborativa e filtragem baseada em conteúdo. O termo filtragem colaborativa foi inicialmente cunhado visando designar um tipo de sistema específico no qual a filtragem de informação é realizada com o auxílio humano. As técnicas baseadas em conteúdo geram recomendações baseadas em informações textuais e técnicas de recuperação de informações. Mais recentemente, estudos têm sido realizados sobre metodologias para combinar as duas abordagens, dando origem a técnicas híbridas [Burke 2002].

Alguns pacotes e ferramentas têm sido propostos visando prover acesso a algoritmos de recomendação. As ferramentas CoFE - Collaborative Filtering Engine [CoFE 2004] e Multilens [Miller 2003], por exemplo, podem ser acopladas às aplicações, são independentes de domínio de aplicação e suportam armazenamento e configuração de banco de dados relacionais. Todavia não permitem a utilização de outras técnicas e têm limitações quanto à *interface* de acesso, baseada no uso de linguagem específica. O pacote Duine [van Setten et al. 2002] suporta várias técnicas de recomendação e configuração de parâmetros, é independente de domínio e possibilita o armazenamento das informações em banco de dados. Todavia, a ferramenta apresenta limitações relacionadas ao uso de linguagem específica por as aplicações clientes. O projeto EasyUtil [EasyUtil 2006] oferece um serviço de recomendação colaborativo operando sobre o protocolo HTTP e os resultados são codificados em formato XML. A ferramenta suporta aplicações em linguagens e domínios diversos. Entretanto, apresenta desvantagens, dado que se limita às técnicas colaborativas e não oferecem interfaces formalmente definidas, baseando-se apenas na URL do serviço. O pacote Taste [Owen 2005] pode ser acoplado a aplicações clientes ou pode ser acessada através de interface de Serviços Web. Embora portátil tecnologicamente, a ferramenta limita-se às técnicas colaborativas de recomendação. O projeto *The MobLife Recommender* [Petteri Nurmi 2006] também oferece um serviço de recomendação baseado em Serviços Web e interfaces formalmente definidas em WSDL. A utilização de Serviços Web possibilita boa portabilidade, todavia, a ferramenta também não permite a utilização de outras técnicas de recomendação.

3. Descrição da Ferramenta

3.1. Arquitetura Geral da Plataforma

A arquitetura da plataforma foi definida tendo em vista os objetivos de flexibilidade propostos. A solução da abordagem proposta para a independência de domínio de aplicação consistiu na utilização de termos genéricos e abrangentes para designar as informações armazenadas. Em relação ao requisito de extensibilidade da plataforma, a solução proposta consiste em uma estrutura de *engines*, semelhante ao conceito de *plugins*. Sob o ponto de vista tecnológico foram selecionadas tecnologias abrangentes e padrões já bem aceitos e

difundidos, como Serviços Web. A Figura 1 ilustra a arquitetura geral da plataforma sob diversos aspectos, discutidos a seguir.

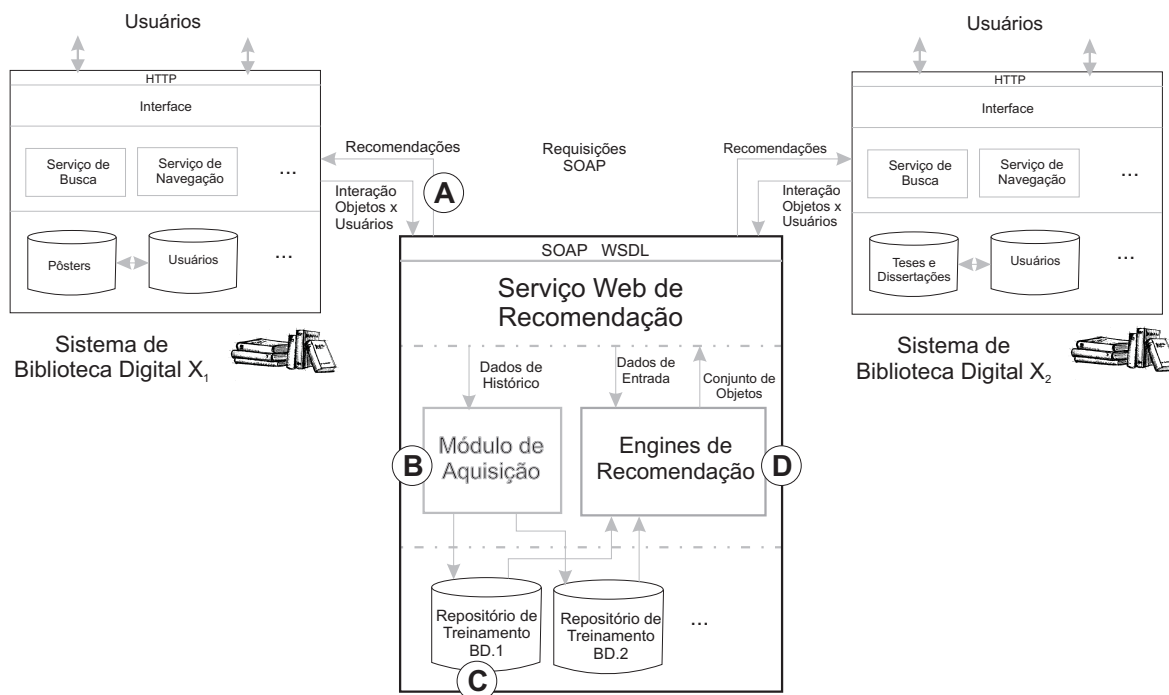


Figura 1. Arquitetura Geral da Plataforma.

Utilizando Serviços Web, os métodos de acesso aos serviços da plataforma são definidos em WSDL [Christensen et al. 2004] e a chamada dos serviços opera através de requisições SOAP [Gudgin et al. 2004], sobre o protocolo HTTP. A parte (A) da Figura 1 ilustra essa interação. Definida a interface, o modelo arquitetural proposto divide a plataforma internamente em dois grandes módulos: Módulo de Aquisição e Engines de Recomendação, ilustrados respectivamente nas partes (B) e (D) da Figura 1.

Um cenário de requisição de recomendação é representado no Diagrama de Sequência apresentado na Figura 2. Uma Biblioteca Digital faz uma requisição à plataforma, por meio da classe *RecommenderWS*. Essa classe faz uma requisição à *EngineFactory*, que por sua vez retorna uma instância de um *engine* de recomendação. Por fim, é realizada a chamada ao método do *engine* que retorna uma lista de itens recomendados.

3.2. Módulo de Aquisição

Dado que as técnicas de recomendação baseiam-se principalmente em dados de histórico de comportamento dos usuário, a plataforma dispõe de um módulo de aquisição de dados de treinamento. As interações desse módulo na plataforma são ilustradas na parte (B) da Figura 1. As principais classes responsáveis pela implementação do Módulo de Aquisição são basicamente classes que implementam o *design pattern* DAO (*Data Access Object*), para acesso às entidades da plataforma: *Usuários*, *Itens* e *Avaliações* para esses itens.

As informações gerenciadas pelo Módulo de Aquisição são armazenadas em um SGBD, para posterior utilização pelos algoritmos de recomendação. Este módulo está preparado para operar sobre SGBDs diversos, atendendo também aos requisitos de portabilidade.

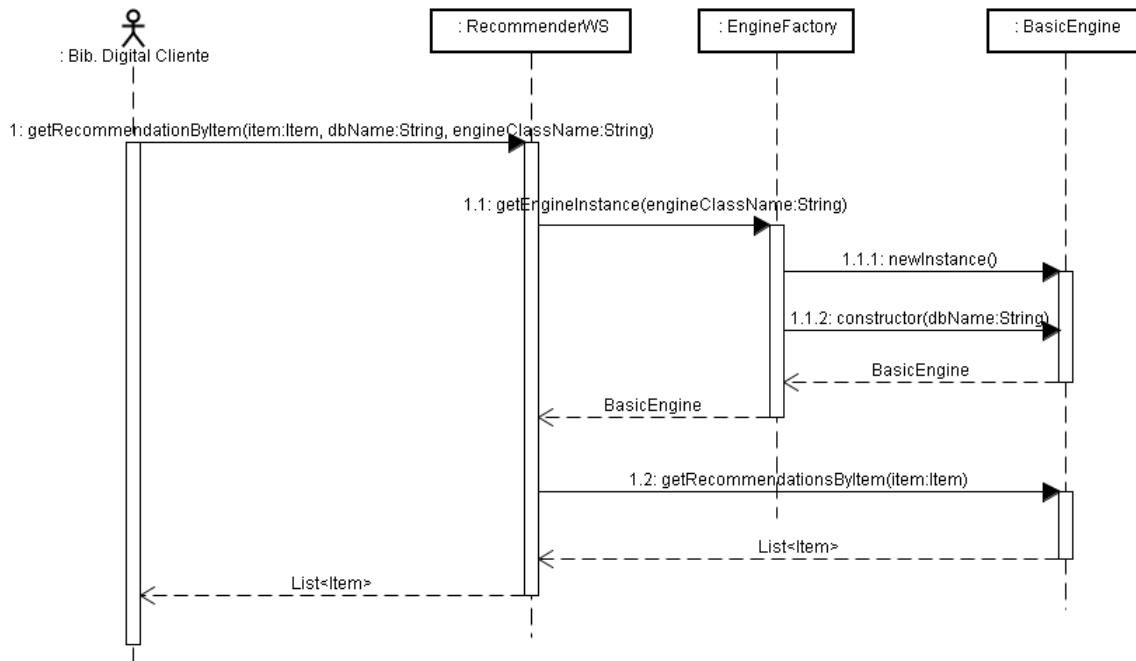


Figura 2. Diagrama de Sequência de uma requisição aos Engines de Recomendação.

3.3. Engines de Recomendação

Os *engines* consistem em mecanismos de recomendação, desenvolvidos segundo padrões e *interfaces* bem definidas pela plataforma. O funcionamento desses elementos assemelha-se ao conceito de *plugin*, já bastante difundido em computação, nos quais módulos de software podem ser desenvolvidos e facilmente instalados sobre uma aplicação principal.

Tal conceito torna a plataforma extensível sem, no entanto, afetar a forma de comunicação e requisição de serviços junto às Bibliotecas Digitais clientes. Outra vantagem importante consiste na independência em relação à técnica de recomendação utilizada, dado que as implementações encontradas na literatura são em geral atreladas a uma técnica específica.

Os *engines* de recomendação são compostos basicamente por dois elementos: um pacote de implementação dada por uma biblioteca Java e estendendo classes da plataforma; e um conjunto de metadados, contendo as principais informações e parâmetros do *engine*.

4. Demonstração da Ferramenta

4.1. Protótipo

Para o desenvolvimento do protótipo foram testadas diversas ferramentas baseadas em software livre. O pacote Apache Axis foi utilizado para prover a interface de Serviço Web, descrições WSDL e requisições SOAP.

Os testes foram realizados selecionando-se um conjunto de serviços da plataforma

e executando-os sobre os mesmos dados, mas em ferramentas distintas. Os testes selecionados foram requisições de recomendação para um *Usuário*, para um *Item* e a consulta a um *Item*.

Quanto aos servidores de aplicação, foram selecionados para os testes os servidores JBoss e Apache Tomcat. Em todos os testes realizados a plataforma comportou-se da mesma forma em ambas as ferramentas. Em relação aos SGBDs, foram selecionados o MySQL e PostgreSQL, nos quais a plataforma apresentou idêntico comportamento.

Dado que a plataforma foi desenvolvida utilizando tecnologias Java, o protótipo procurou validar também a independência de linguagem do aplicativo de Biblioteca Digital cliente. Assim, o cliente do protótipo foi desenvolvido utilizando linguagem PHP sobre servidor Apache.

4.2. Detalhes da Demonstração

A validação em relação à independência de domínio de aplicação foi abordada realizando experimentos com Bibliotecas Digitais distintas. Foram selecionados um *dataset* de uso livre e frequentemente citado na literatura e a Biblioteca Digital da Unicamp como cenários de uso para a plataforma. Os detalhes são apresentados a seguir:

Movielens: Em [Good et al. 1999] é proposto um mecanismo de recomendação para filmes, baseado em técnicas colaborativas. Como base de experimentos foi utilizado um conjunto de dados de 100.000 avaliações (*ratings*) para 1.682 filmes, avaliados por 943 usuários. Foram realizados testes de requisições de recomendação para um determinado *Usuário* e *Item* para cada uma das técnicas implementadas.

Biblioteca Digital da Unicamp: Dada a grande quantidade de informações e a demanda por novas funcionalidades, a Biblioteca Digital da Unicamp apresentou-se como um excelente cenário de uso para a plataforma de recomendação proposta. Foi selecionado para os experimentos um subconjunto contendo 6.760 itens (entre teses, dissertações, relatórios técnicos e outros) e 115.568 registros de *downloads* efetuados por 56.524 usuários distintos. A plataforma de recomendação foi configurada em uma base de dados que continha este subconjunto da biblioteca. A aplicação web do protótipo foi utilizada para realizar os experimentos. Foram realizados testes de consulta e requisições de recomendação utilizando dados reais da Biblioteca Digital da Unicamp. A Figura 3 ilustra esses testes: a parte (A) mostra a tela inicial do protótipo durante uma requisição de recomendação ao *engine* ICLuceneRecommender; a parte (B) mostra os detalhes de uma consulta realizada; e por fim a parte (C) exibe os itens recomendados para esse item.

5. Agradecimentos

Os autores agradecem a FAPESP, CAPES, CNPQ (Proc. 477039/2006-5, 311309/2006-2) FAEPEX, Microsoft eScience and Tablet PC Technology and Higher Education Projects pelo apoio financeiro. Os autores também são gratos aos colaboradores da Biblioteca Central da Unicamp, do CCUEC e da DGA.

Referências

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction Journal*, 12(4):331–370.

SBBD 2007 IV Sessão de Demos

The image shows a screenshot of the 'Recommender WS' platform interface. It is divided into three main panels labeled A, B, and C, each with a 'Serviço Web de Recomendação' header. Panel A shows 'Recomendação por Item' with a search bar and buttons for 'Recomendações' and 'Learn!'. Panel B shows 'Recomendação por Usuário' with a search bar and buttons for 'Recomendações' and 'Learn!'. Panel C shows 'Consultar Item' with a search bar and a detailed view of item 15753, including its title, author, and a summary. Below the panels is a table of recommendations.

Item	Título	Autor
16303	Um algoritmo para comparação sintática de genomas baseado na complexidade condicional de Kolmogorov	Marcelo Cezar Pinto
11524	Identificação e caracterização de genes potencialmente transferidos horizontalmente no genoma do <i>Biopatógeno C. perniciosa</i> , causador da doença vasculosa de bruxa no cacauero	Jose Pedro Fonseca
15195	Rearranjo de genomas : uma coletânea de artigos	Zanoni Dias
11149	Estudo do perfil da expressão genica global em leucemias linfóides agudas de linhagens de células B e T	Diana Azevedo Queiroz
16478	Bioinformática de projetos genoma de bactérias	Vagner Katsumi Okura
13145	Expressão e detecção de genes envolvidos com patogenicidade de <i>Crispella perniciosa</i>	Maricene Sabba

Figura 3. Plataforma de Recomendação com dados da Biblioteca Digital da Unicamp.

- Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2004). WSDL: Web services definition language. W3C Technical Reports on WSDL, published online at <http://www.w3.org/TR/wsdl/>.
- CoFE (2004). CoFE - Collaborative Filtering Engine. Disponível em <http://eecs.oregonstate.edu/iis/cofe/>. acessado em 10 de abril de 2007.
- EasyUtil (2006). EasyUtil Recommendation Service. Disponível em <http://easyutil.com/>. acessado em 10 de abril de 2007.
- Good, N., Schafer, B., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J., and Riedl, J. (1999). Combining collaborative filtering with personal agents for better recommendations. In *AAAI/IAAI*, pages 439–446.
- Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., and Nielsen, H. F. (2004). SOAP: Simple object access protocol. W3C Technical Reports on SOAP, published online at <http://www.w3.org/TR/soap/>.
- Miller, B. N. (2003). *Toward a Personal Recommender System*. PhD thesis. Adviser-John Riedl, University of Minnesota.
- Owen, S. (2005). Taste Documentation. Disponível online em <http://sourceforge.net/projects/taste/>. Acessado em 10 de abril de 2007.
- Petteri Nurmi, Jukka Suomela, E. L. (2006). The mobilife recommender. Disponível em <http://www.cs.helsinki.fi/group/acs/mobilife/>. Acessado em 10 de abril de 2007.
- Reategui, E. B. and Cazella, S. C. (2005). Sistemas de recomendação. In *ENIA 2005: Anais do Encontro Nacional de Inteligência Artificial*, pages 306–349.
- van Setten, M., Veenstra, M., and Nijholt, A. (2002). Prediction strategies: Combining prediction techniques to optimize personalization. In *Personalization in Future TV'02 at the Adaptive Hypermedia 2002 conference*, pages 78–91, Malaga, Spain.

Um Mediador para o Processamento de Consultas sobre Bases XML Distribuídas

Guilherme Figueiredo¹, Vanessa Braganholo², Marta Mattoso¹

¹Programa de Engenharia de Sistemas e Computação – COPPE/UFRJ

²Departamento de Ciência da Computação – IM/UFRJ

{g.coelho, marta}@cos.ufrj.br, braganholo@dcc.ufrj.br

Abstract. *This paper describes a tool that implements an architecture to the query processing of XQueries over distributed and fragmented XML databases. This architecture, based on a Mediator with Adapters attached to the remote databases, implements a query processing methodology where the Mediator publishes a global XML view of the distributed data which can be queried transparently.*

Resumo. *Este artigo descreve uma ferramenta que implementa uma arquitetura para o processamento de consultas XQuery sobre bases de dados XML distribuídas e fragmentadas. Esta arquitetura, baseada em um Mediador com Adaptadores acoplados aos bancos de dados remotos, implementa uma metodologia de processamento de consultas distribuídas, onde o Mediador fornece uma visão XML global dos dados distribuídos que pode ser consultada de forma transparente.*

1. Introdução

O processamento distribuído de consultas XML ganhou importância com a popularidade do uso de documentos XML pela *Web*. Neste contexto, surgiram uma série de sistemas para processamento de consultas sobre bases XML distribuídas na *Web*, como [Suciu 2002, Gertz, Bremer 2003, Re et al. 2004]. Outros trabalhos enfocam o processamento de consultas XML sobre bases heterogêneas distribuídas, como [Baru et al. 1999, Gardarin et al. 2002, Lee et al. 2002]. Entretanto, nenhuma dessas propostas utiliza técnicas de fragmentação de bases XML.

Técnicas de fragmentação de bases XML nativas começaram a ser exploradas com o aumento crescente do volume de dados XML armazenado. Algumas técnicas já foram elaboradas [Bremer, Gertz 2003, Andrade et al. 2006]. Em especial, as definições de fragmentos propostas por Andrade et al. fazem uso de uma álgebra XML (TLC [Paparizos et al. 2004]). Neste trabalho, são também definidas regras formais de reconstrução de documentos XML a partir de seus fragmentos. Experimentos que mostram o potencial de ganho em desempenho para consultas realizadas sobre bases XML fragmentadas também foram explorados [Andrade et al. 2006].

Para que o processamento de uma consulta sobre uma base XML nativa distribuída seja automático e genérico, se faz necessário o uso de um formalismo XML. Através de uma álgebra XML, uma consulta XQuery pode ser reescrita através de expressões algébricas. Ao usar fragmentos definidos com operações dessa mesma álgebra, regras formais de equivalência entre um documento XML e seus fragmentos

podem ser usadas. Assim, torna-se possível substituir a referência a um documento XML, numa expressão algébrica de uma consulta, por referências aos fragmentos que compõem esse documento a partir da sua regra algébrica de reconstrução. Essa substituição pode ser realizada de forma correta e automática, quando os fragmentos são definidos por operações da mesma álgebra usada na reescrita da consulta XQuery. Na nossa proposta utilizamos a álgebra TLC [Paparizos et al. 2004].

Entretanto, esse formalismo não é suficiente. Um processamento automático distribuído necessita de uma metodologia que defina etapas que possibilitem o processamento de consultas XQuery distribuídas, de forma análoga à metodologia existente para o modelo relacional [Özsu, Valduriez 1999]. Esta metodologia foi definida em nosso trabalho anterior [Figueiredo et al. 2007] e implementada na forma de uma arquitetura baseada em um Mediador com Adaptadores acoplados às bases remotas, que será descrita neste artigo. O uso desta arquitetura torna o processamento de consultas XQuery distribuídas totalmente transparente para o usuário final. Através da ferramenta construída, é possível visualizar graficamente todas as etapas de processamento da consulta distribuída: decomposição, localização dos dados, e reconstrução do resultado final. Deste modo, a ferramenta pode também ser utilizada para fins didáticos.

A arquitetura proposta para o processamento de consultas distribuídas é apresentada na seção 2. A seção 3 apresenta a interface para testes do Mediador, enquanto que a seção 4 contém as considerações finais.

2. Arquitetura do Sistema

Em nossa arquitetura, gostaríamos que uma base de dados pudesse ser vista de duas maneiras distintas: (i) quando uma base centralizada é fragmentada, deve-se ter uma visão centralizada dos fragmentos, para que a fragmentação fique transparente para o usuário; (ii) quando várias bases locais pré-existentes precisam ser acessadas de forma integrada, deve-se poder definir uma visão XML sobre elas, de modo que cada base local possa ser considerado um fragmento desta visão global (exemplo: uma rede de livrarias, onde cada base de filial possa ser vista como um fragmento de uma visão global (virtual)). Deste modo, definimos uma arquitetura que suportasse ambas estas visões, e chamamos a base central (em ambos os casos), de visão global.

Para compor uma visão global e servir de ponto único de acesso, consideraremos o uso de um mediador [Wiederhold 1992] como mostrado na Figura 1, que seria responsável pelo processamento das consultas distribuídas, ocultando dos usuários os detalhes da localização e da fragmentação da base de dados. As consultas submetidas sobre uma visão global seriam decompostas em um conjunto de sub-consultas, que seriam então executadas pelos nodos remotos sobre os fragmentos. Os resultados de cada sub-consulta retornariam ao mediador para construção do resultado final. No contexto de bases XML distribuídas, consultas XQuery seriam submetidas sobre visões globais de fragmentos XML distribuídos.

A arquitetura apresentada na Figura 1, sobre a qual nossa implementação foi totalmente baseada, contempla os seguintes componentes: Mediador, Adaptadores e Catálogo, que serão detalhados nas seções que seguem.

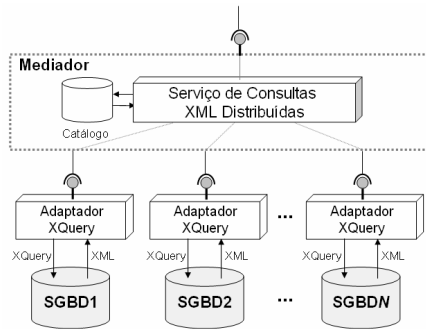


Figura 1: Arquitetura Mediator – Adaptadores

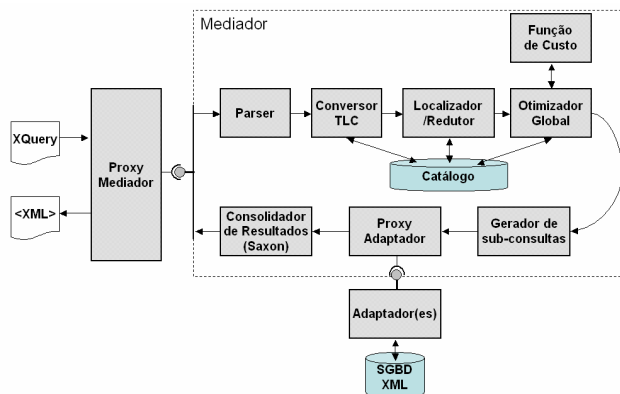


Figura 2: Diagrama de blocos dos componentes do Mediator.

2.1. Mediator

O Mediator é o principal componente da arquitetura, pois ele é responsável pelo processamento da consulta distribuída, realizando as etapas de decomposição, localização e otimização global da consulta, de acordo com a metodologia de processamento de consultas sobre bases XML distribuídas [Figueiredo et al. 2007].

O diagrama da Figura 2 segue a arquitetura básica para processamento de consultas apresentada em [Kossman 2000], estendida para o Mediator. Esta arquitetura possui uma série de módulos, responsáveis por partes isoladas do processamento da consulta distribuída, descritos a seguir.

Parser: responsável pela validação sintática da consulta XQuery.

Conversor TLC: módulo responsável pela representação algébrica TLC da consulta XQuery. Implementa o algoritmo de conversão da XQuery para a TLC descrito em [Paparizos et al. 2004]. Esta representação algébrica em TLC será utilizada pelas próximas fases do processamento da consulta.

Localizador/Redutor: responsável pela localização do plano algébrico da consulta. Essa etapa compreende duas atividades: (i) substituição das referências a coleções globais por referências a fragmentos destas coleções, e (ii) eliminação dos fragmentos irrelevantes ao resultado da consulta, utilizando a abordagem de redução do plano algébrico definida na nossa metodologia [Figueiredo et al. 2007].

Otimizador Global: responsável pela otimização global do plano algébrico. Na implementação do protótipo, desenvolvemos um otimizador que produz um conjunto de planos algébricos equivalentes a partir das réplicas dos fragmentos existentes no ambiente distribuído para descobrir, com o uso de uma função de custo, o plano de menor custo total dentre os planos gerados.

Função de Custo: função para a estimativa do custo de execução de um plano algébrico a partir dos dados estatísticos de cada fragmento, como o número de nodos, tamanho médio em *bytes* de cada nodo, parâmetros de seletividade, peso de leitura em disco, peso de comunicação, etc. Na implementação do protótipo utilizamos apenas o

peso da comunicação e a estimativa do número total de nodos do fragmento para o cálculo do custo do plano algébrico.

Gerador de Sub-consultas: responsável pela extração e composição das sub-expressões (sub-consultas) do plano algébrico otimizado. Cada sub-consulta é transformada em uma representação em XQuery e enviada para o seu Adaptador correspondente. A composição do resultado final será feita também por uma sub-consulta, que é executada pelo próprio Mediador sobre os resultados das sub-consultas remotas. As sub-consultas são criadas a partir de operações da TLC, através de um algoritmo inverso ao algoritmo de conversão da XQuery, produzindo uma consulta textual em XQuery a partir de expressões de operações algébricas.

Este componente merece uma atenção especial, já que ele é o grande responsável pela característica não intrusiva de nossa arquitetura. O SGBD XML nativo não precisa ser modificado para uso de nossa solução. Nossa ferramenta é sempre executada como uma camada acima do SGBD, interceptando as consultas e gerenciando os resultados. Deste modo, como não temos acesso ao processador de consultas interno do SGBD, precisamos transformar as sub-consultas TLC novamente em consultas XQuery, para que elas possam ser enviadas aos adaptadores e executadas por eles.

Proxy do Adaptador: permite a comunicação entre o Mediador e os Adaptadores, através da execução dos protocolos para chamada de serviços *Web*. O *proxy* permite a definição do endereço do Adaptador que será invocado, tornando todo o processo de comunicação com o serviço *Web* transparente para o resto do Mediador.

Consolidador dos Resultados: responsável pela composição do resultado final. Em nossa implementação, a composição do resultado final é realizada através da execução de uma consulta XQuery local sobre os resultados das sub-consultas retornadas pelos Adaptadores. Utilizamos o processador de XQuery Saxon para execução da consulta em memória, sem necessidade de armazenamento dos resultados dos Adaptadores em disco. Se houver apenas uma sub-consulta, não será necessário compor resultados e o resultado final será o próprio resultado desta sub-consulta.

Proxy do Mediador: permite a comunicação de um cliente com o Mediador, através da implementação dos protocolos de comunicação e da configuração de atributos específicos do Mediador.

2.2. Catálogo

O Catálogo armazena todas as informações necessárias para o processamento da consulta distribuída, em especial para a etapa de localização, como o nome e o *schema* das visões globais; os fragmentos que formam a visão global da coleção distribuída; a definição de cada fragmento; o endereço de cada Adaptador remoto que possui uma cópia do fragmento; estatísticas dos fragmentos, como número total de nodos, características de seletividade, etc. O Catálogo foi implementado como um conjunto de objetos em Java que pode ser serializado e desserializado em um documento XML para edição manual.

2.3. Adaptadores

Os Adaptadores são responsáveis pela execução das sub-consultas nos SGBDs XML a eles acoplados. O resultado de cada sub-consulta é retornado ao Mediador para

composição do resultado final. A única responsabilidade do Adaptador, além de implementar a interface de comunicação com o Mediador, é atualizar a localização do documento ou coleção XML sendo consultada, para o seu endereço na base de dados local. Para isso, o Adaptador possui um arquivo de configuração que contém o mapeamento entre o nome do fragmento e seu endereço completo no servidor local. Após a realização deste mapeamento, o Adaptador pode executar a consulta utilizando a interface ou API do banco de dados por ele acoplado ao ambiente. Na nossa implementação utilizamos o banco de dados XML nativo eXist. O diagrama de blocos dos componentes de um Adaptador é apresentado na Figura 3.

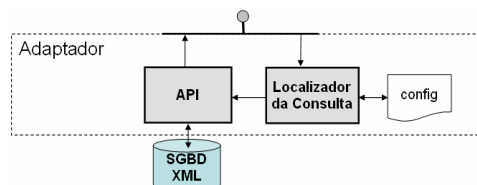


Figura 3: Diagrama de blocos dos componentes do Adaptador

3. Execução de Consultas sobre Bases Distribuídas

Consultas podem ser submetidas sobre as bases distribuídas de duas formas: através do *proxy* do Mediador; ou através de uma interface desenvolvida para testes do sistema e que permite a visualização gráfica da representação algébrica da consulta nas diferentes etapas de processamento da consulta.

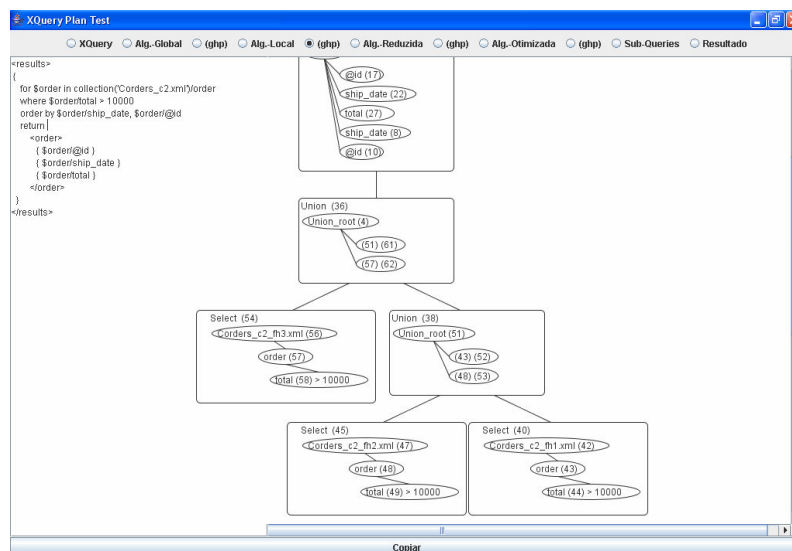


Figura 4: Interface gráfica para execução de consultas no Mediador.

O *proxy* do Mediador é uma classe que contém métodos que simplificam a comunicação de um cliente com o Mediador, através da implementação do protocolo de comunicação. Já a interface gráfica (Figura 4) para execução de consultas sobre o Mediador foi desenvolvida para que pudéssemos acompanhar a evolução da consulta submetida durante as etapas da metodologia de processamento de consultas. Através da interface, tem-se uma noção muito clara de todas as etapas, o que facilita o entendimento da metodologia. Nossa demonstração será focada nesta interface.

4. Conclusão

Este artigo apresentou um mediador que implementa uma metodologia para o processamento de consultas XQuery sobre bases de dados XML nativas distribuídas e fragmentadas [Figueiredo et al. 2007]. O mediador decompõe uma consulta XQuery em sub-consultas que são destinadas a adaptadores acoplados aos bancos de dados remotos. Estes adaptadores executam a sub-consulta através do SGBD XML a ele acoplado e retornam o resultado ao Mediador, que realiza a composição do resultado final. Consultas podem ser submetidas programaticamente através de uma classe *proxy* que se comunica com o Mediador ou manualmente através de uma interface gráfica que executa a consulta no Mediador e ainda exibe a representação algébrica da consulta submetida nas diferentes etapas de processamento.

Nossa arquitetura torna o processamento de consultas XQuery distribuídas totalmente transparente para o usuário final, além de ser não intrusiva e poder ser utilizada com qualquer SGBD. O fato de utilizarmos adaptadores nas bases locais faz com que seja possível inclusive utilizar bases não XML (relacionais, por exemplo). Basta, para isso, que o adaptador daquela base publique os dados no formato XML esperado por aquele fragmento.

Referências

- Andrade, A., Ruberg, G., Baião, F., Braganholo, V., Mattoso, M. (2006) "Efficiently processing XML queries over fragmented repositories with PartiX", In: DATA, p. 150-163, Munich, Germany.
- Baru, C., Gupta, A., Ludaesher, B., Marciano, R., Papakonstantinou, Y., Pavel, V., Chu, V. (1999) "XML-Based Information Mediation with MIX", In: SIGMOD, p. 597-599. ACM Press.
- Bremer, J.-M., Gertz, M. (2003) "On Distributing XML Repositories", In: WebDB, p. 73-78, San Diego, California.
- Figueiredo, G. C., Braganholo, V., Mattoso, M. (2007) "A Methodology for Query Processing over Distributed XML Databases", In: Technical Report ES-710/07 (<http://www.dcc.ufrj.br/~braganholo/artigos/RT-guilherme.pdf>), p. 1-24, Rio de Janeiro, Brazil.
- Gardarin, G., Mensch, A., Dang-Ngoc, T.-T., Smit, L. (2002) "Integrating Heterogeneous Data Sources with XML and XQuery", In: DEXA, p. 839-846. IEEE Computer Society.
- Gertz, M., Bremer, J.-M. (2003) "Distributed XML Repositories: Top-down Design and Transparent Query Processing". Department of Computer Science.
- Kossman, D. (2000) "The State of the Art in Distributed Query Processing", In: ACM Computing Surveys, v. 32, p. 422-469.
- Lee, K., Min, J., Park, K., Lee, K. (2002) "A Design and Implementation of XML-Based Mediation Framework (XMF) for Integration of Internet Information Resources", In: Hawaii International Conference on System Sciences, v. 7, p. 202-211. IEEE Computer Society.
- Özsu, M. T., Valduriez, P. (1999) "Principles of Distributed Database Systems". 2 ed., Prentice Hall
- Paparizos, S., Wu, Y., Lakshmanan, L. V. S., Jagadish, H. V. (2004) "Tree Logical Classes for Efficient Evaluation of XQuery", In: SIGMOD, p. 71-82. ACM.
- Re, C., Brinkley, J., Hinshaw, K. P., Suciu, D. (2004) "Distributed XQuery", In: IIWeb, p. 116-121, Toronto, Canada.
- Suciu, D. (2002) "Distributed Query Evaluation on Semistructured Data", ACM TODS, v. 27, 1, p. 1-62.
- Wiederhold, G. (1992) "Mediators in the Architecture of Future Information Systems", In Michael N.Huhns and Munindar P.Singh, Readings in AgentsMorgan Kaufmann

NanoBase: Um Processador de Consultas para a Plataforma JME/CLDC

Leonardo Eloy¹, Vitor Araújo¹, José Maria Monteiro¹, Ângelo Brayner¹

¹Centro de Ciências Tecnológicas – Universidade de Fortaleza (Unifor)
{eloy,vitor}@unifor.edu.br, {monteiro,brayner}@unifor.br

Resumo. Os recentes avanços obtidos na tecnologia de computação móvel possibilitam o desenvolvimento de novas e sofisticadas aplicações, as quais podem agora armazenar dados no próprio dispositivo móvel. A plataforma JME vem se consolidando como um padrão para o desenvolvimento de aplicações em dispositivos móveis, também denominadas de aplicações embarcadas. Nesta plataforma, a API RMS é responsável pela persistência dos dados. Esta API possibilita a recuperação, inserção, alteração e exclusão de registros em arquivos representados por vetores (arrays) de bytes. Tal característica torna a programação bastante complexa e de baixa produtividade, principalmente no tocante à persistência de dados. Além disso, como nenhuma estrutura de índice é disponibilizada, o acesso seletivo aos registros é bastante demorado, podendo até mesmo inviabilizar o desenvolvimento de determinadas aplicações. Este trabalho apresenta o NanoBase, um mecanismo que fornece funcionalidades de um SGBD relacional para a plataforma JME. O mecanismo proposto fornece aos programadores uma visão relacional dos dados. Outras propriedades importantes do NanoBase são o suporte para execução de consultas, expressas através da linguagem SQL, e a criação de diversas estruturas de índices, como árvores B^+ e índices hash. A fim de comprovar os benefícios da utilização do NanoBase, foram realizados diversos testes de desempenho e uma análise comparativa com outras soluções.

1. Introdução

Os recentes avanços obtidos na tecnologia empregada nos dispositivos portáteis (tais como PDAs, telefones celulares e *smartphones*) estão contribuindo para diminuir o custo destes equipamentos, tornando-os acessíveis a um grupo cada vez maior de pessoas. Estima-se, por exemplo, que atualmente existam mais de um bilhão de assinantes de serviços de comunicação sem fio em todo o mundo [Per03] e que três bilhões de pessoas usarão dispositivos móveis em 2010 [Fac05].

Esses avanços têm contribuído para aumentar a capacidade computacional e de armazenamento dos equipamentos portáteis. No início de 2006, a Seagate, empresa fabricante de discos rígidos, apresentou um HD de 12 GB para dispositivos portáteis (Seagate ST1.3 Series) [Terra07]. Esses mini-HDs utilizam memórias *flash* (e.g., *Smart Media* e *MultiMediaCard*) para armazenar dados de forma permanente. Este tipo de memória não volátil utiliza a tecnologia EEPROM (*Electrically Erasable Programmable Read-Only Memory*) e se caracteriza por possuir pequeno tamanho físico, peso reduzido e resistência a choques. Entretanto, as operações de leitura e escrita são lentas [Mag04].

Com recursos cada vez mais avançados, e custo cada vez mais atraente, possibilita-se o desenvolvimento de novas e sofisticadas aplicações, cuja demanda deve seguir a tendência de crescimento. Como exemplos destas aplicações, podemos citar: automação da força de vendas, *banking*, bilhetagem, reserva de ingressos, *Data Warehouse* Móvel

(DWM), além do monitoramento e coleta de dados de pacientes [Pla03, Shan99]. Desta forma, muitas dessas novas aplicações necessitarão manipular um volume de informações cada vez maior, carecendo assim, de um mecanismo que facilite o armazenamento e a recuperação de dados em dispositivos móveis.

A plataforma JME (*Java Micro Edition*) tem se firmado como um padrão para o desenvolvimento de aplicações voltadas para equipamentos portáteis. Entretanto, umas das grandes dificuldades do desenvolvimento de software voltado para a plataforma JME é a persistência de dados. A fim de permitir o armazenamento e a recuperação de dados, a plataforma JME fornece a API RMS (*Record Management System*). Esta API permite ao programador criar agrupamentos de registros denominados “*RecordStores*”. A estrutura de armazenamento de um *RecordStore* é bem simplória, possui apenas um *id* e um *array* de *bytes* como local para armazenamento de dados. O acesso a esses registros se dá unicamente de forma seqüencial através do *id* que os identifica. Assim, para localizar um determinado registro, é necessário percorrer todos os registros de um determinado *RecordStore*. Além disso, a API RMS não apresenta suporte a tabelas, índices e nem a consultas de dados (e.g., utilizando SQL), oferecendo apenas recursos precários de filtro e busca [Mag04]. Tudo isso torna a persistência de dados na plataforma JME/CLDC uma tarefa bastante complexa e ineficiente. Outrossim, como os dispositivos móveis já conseguem armazenar um volume relativamente grande de dados, o acesso seletivo a estes registros (geralmente implementado através de uma busca seqüencial) torna-se bastante demorado, podendo até mesmo inviabilizar o desenvolvimento de determinadas aplicações.

Desta forma, a fim de viabilizar o desenvolvimento de aplicações centradas em dados para dispositivos portáteis, torna-se de fundamental importância a construção de um mecanismo que ofereça funcionalidades de um SGBD para a plataforma JME/CLDC. Este mecanismo deve possibilitar, por exemplo, a utilização da linguagem SQL para a manipulação de dados, e a utilização de estruturas auxiliares, como índices, que acelerem o processo de recuperação de informações. Neste sentido, este trabalho apresenta o *NanoBase*, um processador de consultas que provê uma visão relacional para a plataforma JME/CLDC, possibilitando a utilização de expressões DDL/DML, restrições de integridade, além de diferentes estruturas de índices (Árvore B⁺, *Hashing* Dinâmico, Índice *Bitmap* e Árvores Kd) para a recuperação de dados. Assim, o mecanismo proposto torna a localização e o acesso aos dados muito mais eficiente, possibilitando o desenvolvimento de aplicações que antes eram impraticáveis devido à baixa produtividade e à demora na recuperação dos dados.

Esse artigo está organizado da seguinte forma: a seção 2 apresenta o *NanoBase*. A seção 3 discute os trabalhos relacionados. Na seção 4 apresentamos os resultados dos testes realizados. A seção 5, conclui este trabalho e aponta direções para trabalhos futuros.

2. O NanoBase

Esta seção tem como objetivo descrever a arquitetura do *NanoBase* (Figura 1). Além disso, será apresentado um exemplo de sua utilização.

O *NanoBase* é constituído por três APIs básicas: *NanoBase API*, *Query Engine* e *Storage*. A *NanoBase API* constitui a camada entre as aplicações e o *NanoBase*. Seguindo uma sintaxe *JDBC-like*, esta API fornece um conjunto de classes e métodos que possibilita ao desenvolvedor submeter cláusulas DDL/DML e manipular o resultado

SBBD 2007

IV Sessão de Demos

de suas consultas. O *Query Engine* é composto por um processador, um otimizador e um executor de consultas. Sua principal função é transformar cláusulas SQL em planos de execução otimizados e proceder a execução desses planos. A *API Storage* é responsável por fornecer uma visão relacional dos dados armazenados, realizando de forma transparente, em RMS ou *FileChannel* (JSR 75), o armazenamento e a recuperação dos dados. Já *Table Metadata* representa os meta-dados utilizados no mapeamento do RMS, ou *FileChannel*, para o modelo relacional, e vice-versa.

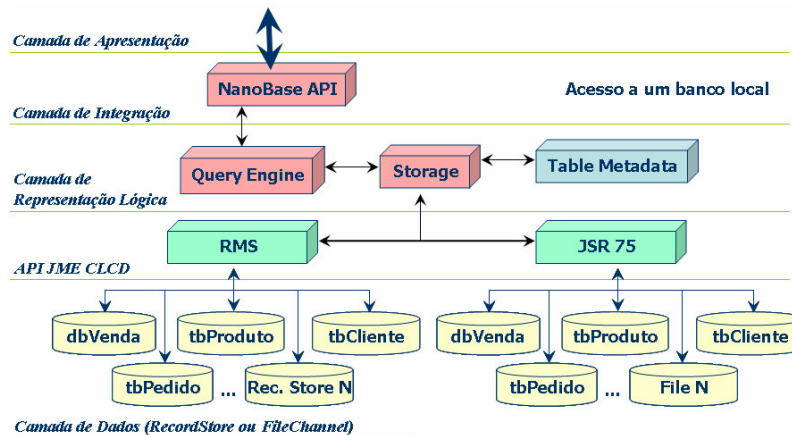


Figura 1 – Arquitetura do NanoBase.

Para ilustrar o uso do *NanoBase* e seus benefícios utilizaremos o seguinte cenário: considere a existência de dois *RecordStores* que armazenam dados de Empregados e de Departamentos. As colunas representadas no *RecordStore* Empregado são as seguintes: código, nome e cod_departamento. Já as colunas do *RecordStore* Departamento são: Código e Descrição. Suponha que a aplicação deseje realizar a seguinte tarefa, denominada *Tarefa 1*: recuperar os nomes de todos os Empregados e a descrição dos departamentos em que estes estão lotados. A seguir, mostraremos como essa tarefa seria implementada utilizando RMS e como foi implementada utilizando o *NanoBase*.

A figura 2 ilustra como a *Tarefa 1* seria implementada através da API RMS. Neste caso, teríamos que percorrer, para cada registro do *RecordStore* Empregado, todos os registros do *RecordStore* Departamento, verificando se a “coluna” cod_departamento do *RecordStore* Empregado é igual à “coluna” código do *RecordStore* Departamento. Já a figura 3 mostra o código necessário para executar a *Tarefa 1* através do *NanoBase*. Observe que para isso foram utilizadas as classes *Connection*, *Statement* e *ResultSet*.

3. Trabalhos Relacionados

Esta seção apresentará uma breve descrição de algumas ferramentas proprietárias relacionadas à persistência de dados para a plataforma JME CLDC/MIDP.

PointBase Micro Edition: Provê uma arquitetura compacta, consumindo somente 90Kb para plataformas JME CDC e 45Kb para JME CLDC/MIDP. Este banco de dados relacional é simples, mas tem como pontos fortes funcionalidades relevantes para dispositivos móveis, como criptografia em nível de coluna ou banco de dados e sincronização bi-direcional com qualquer dispositivo. Disponibiliza uma API que permite realizar conexões JDBC e fornece também uma *MIDlet* que acessa a base de

SBBD 2007

IV Sessão de Demos

dados. As classes disponibilizadas possuem uma terminologia semelhante à API JDBC. Última Versão: 5.5. Fabricante: DataMirror [Point07].

IBM DB2e: Provê uma interface, denominada *FastRecordStore*, que utiliza a API RMS, e efetua sincronização com o banco de dados no servidor. Possibilita a execução em diversas plataformas, como *Windows Mobile*, *Palm OS*, *Linux* e *JME CLDC/MIDP*. Parece ser eficiente, mas apresenta uma API difícil de utilizar. Última Versão: 9.1. Fabricante: IBM [Db2e07].

DB4o: Banco de Dados orientado a objetos. Executa com alguns dialetos JME que suportem reflexão como CDC, *PersonalProfile*, *Symbian*, *Palm OS*, *Savage* e *Zaurus*. Há um estudo para o suporte desta ferramenta a dialetos JME que não utilizem reflexão como MIDP. Possui fortes características de replicação e criptografia e implementa diversas linguagens de consulta, como SQL, *Query by Example* (QbE) e *Simple Object Data Access* (SODA). Última Versão: 6.2. Fabricante: db4objects [Db4o07].

A figura 4 apresenta os resultados de uma análise comparativa entre as principais ferramentas para armazenamento e recuperação de dados em dispositivos portáteis.

```
public void listarEmpregados(){
    RecordStore rsEmpregado = null;
    RecordStore rsDepartamento = null;

    try {
        rsEmpregado = RecordStore.openRecordStore("EMPREGADO",false);
        rsDepartamento = RecordStore.openRecordStore("DEPARTAMENTO",false);

        String regEmpregadoStr = "" , colCodDepartamento = "";
        String regDepartamentoStr = "" , colCodigo = "";

        for(int i=1;i<=rsEmpregado.getNumRecords();i++){

            regEmpregadoStr = new String(rsEmpregado.getRecord(i));
            colCodDepartamento = Util.split(regEmpregadoStr,Constantes.CARACTER_ESPECIAL)[2];

            for(int j=1;j<=rsDepartamento.getNumRecords();j++){
                regDepartamentoStr = new String(rsDepartamento.getRecord(j));
                colCodigo = Util.split(regDepartamentoStr,Constantes.CARACTER_ESPECIAL)[0];

                if(colCodigo.equals(colCodDepartamento)){

                    System.out.println(
                        "Nome:" + Util.split(regEmpregadoStr,Constantes.CARACTER_ESPECIAL)[1]+
                        "Departamento:" + Util.split(regDepartamentoStr,Constantes.CARACTER_ESPECIAL)[0]
                    );
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figura 2 – Tarefa 1 Utilizando a API RMS.

```
public void listarEmpregados() {
    Connection conn = DatabaseConfiguration.getConnection();
    Statement stat = conn.createStatement();
    ResultSet rs = stat.executeQuery("SELECT EMP.NOME,DEP.DESCRICAO " +
                                    "FROM EMPREGADO EMP, DEPARTAMENTO DEP " +
                                    "WHERE EMP.COD_DEPART=DEP.CODIGO");

    while (rs.next()) {
        String nome = rs.getString(1);
        String dpto = rs.getString(2);
        System.out.println("Nome: " + nome + "\nDepartamento: " + dpto);
    }
}
```

Figura 3 – Tarefa 1 Utilizando o NanoBase.

SBBD 2007
IV Sessão de Demos

	NanoBase	RMS	Db4o	PointBase	DB2e
Tamanho do Código	147kb	0 kb	~600kb	Entre 45Kb e 90Kb	*
JME CLDC/MIDP	Sim	Sim	Não	Sim	Sim
Armazenamento	RMS, JSR 75	RMS	*	*	RMS
DML/DDL	Sim	Não	Sim	Sim	Sim
Índices Implementados	B ⁺ , Hashing Dinâmico, Bitmap e Kd	Nenhum	B ⁺	*	*
Persistência dos Índices	Sim	Não se aplica	*	*	*
Índices sobre Múltiplos Atributos	Somente para Kd e Bitmap	Não se aplica	*	*	*
Suporte a Junção (JOIN)	Sim	Não	Sim	Sim	Sim
Operador LIKE	**	Não	Sim	Sim	Sim
Funções Agregadas	**	Não	*	Sim	*
ORDER BY	**	Não	*	Sim	Sim
Tipos de Dados de Colunas	Integer, Varchar, Decimal	Bytes	*	Blob, char, date, decimal, integer, time, timestamp, varchar	*
Chave Primária	Em uma única coluna	Não	*	Sim	*
Prepared Statement	**	Não	*	Sim	*
Suporte a JSR 220 (JPA)	**	Não	*	Não	*
Criptografia de Banco	Não	Não	Sim	Sim	*
Suporte a Transações ACID	Não	Não	Sim	Sim	Sim
Orientada a Objetos	Não	Não	Sim	Não	Não

Figura 4. Comparação entre os SGBDs para Dispositivos Móveis.

* O fabricante não informou ou não deixou claro em sua documentação.

** Previsto para a próxima versão.

4. Resultados Experimentais

Com a finalidade de comprovar os benefícios do *NanoBase* e avaliar os ganhos de desempenho obtidos pela utilização dos índices implementados, nós realizamos uma comparação de performance entre o *NanoBase* e uma busca sequencial via RMS. Para a realização dos testes foi utilizado um *RecordStore* cujos registros possuem apenas dois atributos: o identificador e um valor numérico. Os testes foram repetidos para diferentes quantidades de registros: 100, 200, 400 e 800 registros. A métrica utilizada na avaliação

da performance foi o tempo de resposta em milissegundos (ms). Foram utilizadas duas consultas: uma por igualdade e uma por faixa de valores (“range”), esta última recuperando 20 e 80% dos registros, respectivamente. O aparelho celular utilizado foi o Nokia 6111 (*Heap size*: 2 MB e *Shared Memory for Storage*: 22 MB).

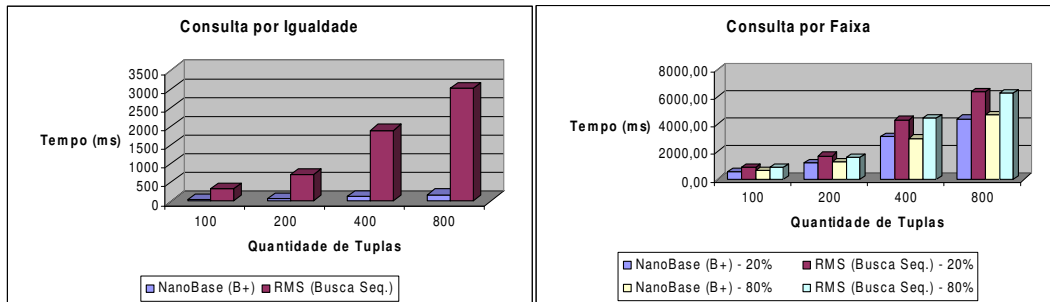


Figura 5 – Comparação de Desempenho entre o NanoBase e a API RMS.

5. Conclusões e Trabalhos Futuros

Neste artigo apresentamos o *NanoBase*, um processador de consultas para a plataforma JME CLDC/MIDP. O *NanoBase* fornece uma visão relacional para acesso a dados na plataforma JME CLDC/MIDP, permitindo a utilização de cláusulas DDL/DML, restrições de integridade, além de diversas estruturas de índices. Os índices existentes podem ser utilizados diretamente através da API do *NanoBase*. A ferramenta proposta torna o acesso aos dados muito mais eficiente, o que possibilita o desenvolvimento de aplicações que antes eram inviáveis devido à baixa produtividade e à demora na recuperação dos dados. Como trabalhos futuros, pretendemos fornecer suporte à criptografia, JSR 220 (JPA) e sincronização de dados entre dispositivos portáteis e servidores de bancos de dados.

Referências

- [Per03] F. Pereira, M. Valente, R. Bigonha e M. Bigonha. Chamada Remota de Métodos a Plataforma J2ME/CLDC. V Workshop de Comunicação Sem Fio e Computação Móvel, 2003.
- [Mag04] K. Magalhães, W. Viana, F. Lemos, J. Machado e R. Andrade. Um Framework de Persistência de Objetos em Aplicações para Dispositivos Móveis. XIX Simpósio Brasileiro de Bancos de Dados, 2004.
- [Shan99] J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran e K. Ramamritham. Efficient Concurrency Control for Broadcast Environments. Proceedings of the ACM SIGMOD Conference, 1999.
- [Fac05] <http://www.facom.ufba.br/ciberpesquisa/cibercidades/disciplinas/>. Acessado em 04/06/2007.
- [Terra07] <http://tecnologia.terra.com.br/interna/0,,OI878349-EI4801,00.html>. Acessado em 04/06/2007.
- [Pla03] Projeto e Construção de um Gerente de Data Warehouses Móveis. Plácido Marinho Dias. Dissertação de Mestrado. UFCG, 2003.
- [Point07] <http://www.pointbase.com/>. Acessado em 01/08/2007.
- [Db2e07] <http://www-306.ibm.com/software/data/db2/everyplace/>. Acessado em 01/08/2007.
- [Db4o07] <http://www.db4o.com/>. Acessado em 01/08/2007.

SimEval: Uma Ferramenta para avaliação de qualidade para funções de similaridade

Francisco N. A. Krieser, Carlos Alberto Heuser, Viviane Moreira Orengo

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

{fkrieser, heuser, vmorengo}@inf.ufrgs.br

Abstract. *Approximate data matching applications typically use similarity functions to quantify the degree of likeness between two data instances. There are several of similarity functions available, thus, it is often necessary to evaluate a number of them aiming at choosing the function that is more adequate to a specific application. This paper presents a tool that uses average precision and discernability to evaluate the quality of similarity functions over a data set.*

Resumo. *Aplicações que realizam casamento aproximado de dados geralmente utilizam funções de similaridade para quantificar o grau de semelhança entre duas instâncias. Existem dezenas de funções de similaridade disponíveis, sendo assim, surge a necessidade de avaliar um conjunto de funções, com o objetivo de escolher qual dessas funções é a mais adequada para uma aplicação específica. Este artigo apresenta uma ferramenta que utiliza as métricas de discernibilidade e precisão média, para avaliar qualidade das funções de similaridade aplicadas a uma base de dados.*

1. Introdução

Os sistemas que realizam casamento aproximado de dados, como junções por similaridade (*similarity join*) [Cohen 2000] e reconhecimento de entidades (*entity recognition*) [Benjelloun et al. 2006] baseiam-se em funções de similaridade para avaliar se duas instâncias (strings, tuplas, etc) representam a mesma entidade do mundo real. Uma função de similaridade $f(v1, v2) \rightarrow s$, recebe como entrada duas *strings* $v1$ e $v2$ e retorna um valor de escore s entre 0 e 1. Se o valor de s for maior do que um limiar t pré-estabelecido, os valores $v1$ e $v2$ são considerados representações de um mesmo objeto no mundo real.

Existe uma grande variedade de funções de similaridade, desde métricas simples para *strings*, como a distância de edição [Levenshtein 1966], até funções específicas para árvores XML [Dorneles et al. 2004]. Devido a esta grande variedade, os projetistas freqüentemente deparam-se com a tarefa de escolher qual a função mais apropriada para uma determinada aplicação. Neste artigo apresentamos a ferramenta *SimEval* que avalia a qualidade de funções de similaridade para um determinado domínio.

A fim de avaliar a qualidade das funções de similaridade, a ferramenta utiliza duas métricas: *precisão média* e *discernibilidade*. A precisão média é a medida mais amplamente utilizada em Recuperação de Informações (RI) para avaliar a qualidade do

resultado de consultas. A *discernibilidade* [Silva et al. 2007], é uma medida proposta especificamente para avaliar a qualidade de funções de similaridade. Uma importante diferença entre as duas medidas é que a *discernibilidade* leva em consideração os escores atribuídos aos resultados irrelevantes e relevantes e não só a posição dos itens no *ranking* apresentado ao usuário.

O restante deste artigo encontra-se organizado da seguinte maneira: a seção 2 trata da metodologia de avaliação de funções de similaridade, a seção 3 trata das características da ferramenta e por fim, a seção 4 faz as considerações finais.

2. Avaliação de funções de similaridade

A avaliação das funções de similaridade pode ser feita através de várias métricas, entre elas, *precisão média* e *discernibilidade*, apresentadas nas Seções 2.1 e 2.2. Ambas as métricas baseiam-se no conceito de relevância. Um item da base de dados que corresponde à mesma entidade da consulta é considerado um resultado relevante. Já, um item da base de dados que não corresponde à mesma entidade da consulta é considerado irrelevante.

A fim de calcular as duas medidas de avaliação de funções de similaridade são necessários: (i) um conjunto de instâncias da base de dados para serem utilizadas como consultas; e (ii) um conjunto de funções de similaridade que queremos avaliar. A seguir, é gerada, para cada consulta e cada função de similaridade, uma tabela contendo uma coluna com todas as entradas da base de dados e outra coluna contendo o escore de similaridade, atribuído pela função, entre o item e a consulta. Essa tabela, chamada de *ranking*, é ordenada de forma decrescente pelo escore. Em seguida, um usuário faz os julgamentos de relevância, i.e., marca para cada *ranking*, quais itens representam o mesmo objeto consultado, conforme o exemplo da Tabela 1.

2.1. Cálculo da Precisão Média

Para cada linha i da tabela que corresponda a uma entrada relevante, é calculada a razão $r_i = n / i$, onde n é o número de entradas relevantes até a linha i . Após calcular a razão r para todas as m linhas relevantes, definimos a precisão média da consulta como sendo a média aritmética de todas as m razões r . A precisão média de uma função de similaridade é a média aritmética das precisões médias de todas as consultas.

2.2. Cálculo da *discernibilidade*

A *discernibilidade* é uma medida de qualidade especificamente projetada para avaliar funções de similaridade proposta em [Silva et al. 2007]. Além de servir para avaliar funções de similaridade, esta medida também estima um limiar ótimo t para ser usado por uma função de similaridade para uma base de dados. Este limiar visa minimizar o número de falsos positivos e de falsos negativos recuperados em resposta a um conjunto de consultas. O cálculo da *discernibilidade* está descrito detalhadamente em [Silva et al. 2007]. Esta seção fornece uma breve descrição da medida.

O cálculo da *discernibilidade* leva em consideração dois aspectos: (i) Se a função de similaridade separou os itens relevantes dos itens não relevantes. Uma boa função de similaridade deve atribuir escores mais altos aos itens do que aos itens irrelevantes; e (ii) O grau de separação entre os itens relevantes e irrelevantes. Uma boa

função de similaridade deve claramente separar resultados relevantes e irrelevantes, criando dois conjuntos nitidamente disjuntos.

A fórmula para o cálculo da *discernibilidade* é apresentada na Equação 1.

$$discernibilidade^L = (t_{best}^{\min}, t_{best}^{\max}, f_{\max}) = \frac{c_1}{c_1 + c_2} (t_{best}^{\max} - t_{best}^{\min}) + \frac{c_2}{c_1 + c_2} \cdot \frac{f_{\max}}{2n} \quad (1)$$

onde: L é a função de similaridade usada; t_{best}^{\min} e t_{best}^{\max} são os limites do intervalo ótimo de limiares; c_1 e c_2 são coeficientes que permitem que o usuário dê importâncias diferentes aos dois aspectos considerados para o cálculo da *discernibilidade*; f_{\max} , que será explicado mais adiante, é o número de pontos atingido pelo intervalo de limiares $[t_{best}^{\min}, t_{best}^{\max}]$. A *discernibilidade* pode assumir qualquer valor dentro do intervalo $[-1, 1]$.

A partir dos julgamentos de relevância fornecidos pelo usuário, é possível identificar dois pontos importantes em um *ranking* gerado por uma função de similaridade para uma consulta: (i) S_{rel} – o menor escore de similaridade atribuído pela função a um item relevante; (ii) S_{irrel} – o maior escore de similaridade atribuído pela função a um item irrelevante.

Exemplo: Considere uma base de dados que contenha nomes de cidades. O objeto consultado “São José dos Campos” encontra-se representado sob 6 formas diferentes, conforme a Tabela 1. Supondo-se que a base de dados contenha apenas 8 instâncias, um *ranking* de similaridade de acordo com uma função de similaridade hipotética A é construído. De acordo com este *ranking* o escore mais baixo de um item relevante é $S_{rel}=0.83$ e o escore mais alto de um item irrelevante é $S_{irrel} = 0.5$.

Tabela 1 – Exemplo de *ranking* por similaridade para a consulta “São José dos Campos”

<i>Escore</i>	<i>Instância</i>	<i>Relevância</i>
1.00	São José dos Campos	Relevante
0.95	São José Campos	Relevante
0.93	São José dos Camps	Relevante
0.90	S. José dos Campos	Relevante
0.85	S.J Campos	Relevante
0.83	S.J.C	Relevante
0.5	São Carlos	Irrelevante
0.4	Campos do Jordão	Irrelevante

A função do exemplo foi capaz de separar corretamente os itens relevantes dos irrelevantes para a consulta em questão, pois o último item relevante foi recuperado antes do primeiro item irrelevante ($S_{rel} > S_{irrel}$). Se uma função não separar corretamente o conjunto relevante do irrelevante, S_{irrel} será maior do que S_{rel} . Em consequência disso, a função de similaridade será penalizada com uma baixa *discernibilidade*.

Ao representar graficamente os valores de S_{rel} e S_{irrel} (ver Figura 1) é possível visualizar a distância entre os itens relevantes e irrelevantes, o que permite uma avaliação visual sobre a qualidade da função de similaridade.

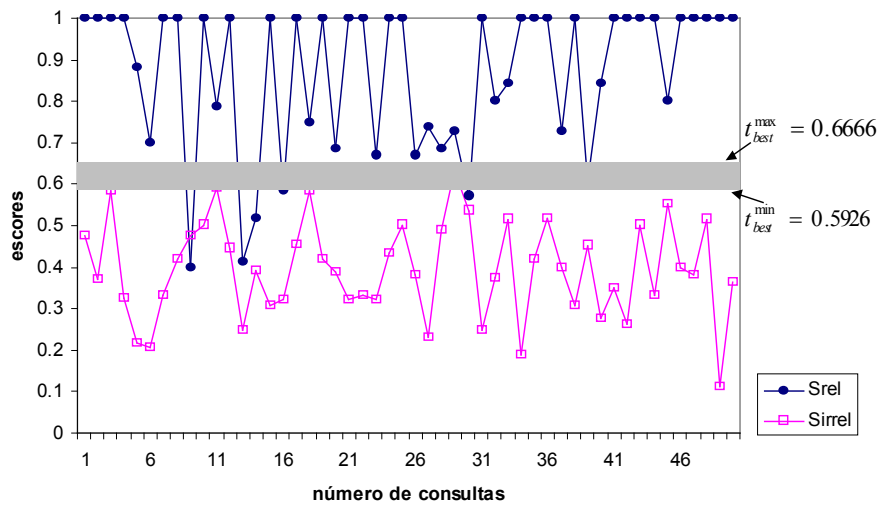


Figura 1 – Gráfico dos valores de S_{rel} e S_{irrel} para 50 consultas. O intervalo de limiares que melhor separa os itens relevantes dos irrelevantes está em cinza.

O algoritmo *BestThresh*, que encontra o intervalo ótimo de limiares (destacado em cinza na Figura 1) é baseado em uma *função ganho* (*reward function*) e procede da seguinte maneira: Cada limiar t , no intervalo $[0,1]$ (incrementado de acordo com uma precisão numérica pré-definida), é comparado com de S_{rel} e S_{irrel} para os *rankings* produzidos em resposta a um conjunto de consultas. Essas comparações resultarão em uma das possibilidades abaixo.

- i. O limiar t é simultaneamente menor do que S_{rel} e maior do S_{irrel} . Isto significa que o limiar consegue separar os itens relevantes dos irrelevantes. Neste caso t ganha dois pontos.
- ii. O limiar t é satisfaz apenas uma das condições. Isto significa que ambos S_{rel} e S_{irrel} estão do mesmo lado da linha traçada por t . Logo, o limiar não separa itens relevantes de irrelevantes. Neste caso t não ganha pontos.
- iii. O limiar t não satisfaz nenhuma das condições. Neste caso t perde dois pontos.

Após computar a pontuação para todos os limiares, o algoritmo busca pela maior pontuação atingida por um limiar (f_{max}). Uma vez que f_{max} é identificado, o algoritmo busca pelo intervalo de valores sucessivos de t ($[t_{best}^{min}, t_{best}^{max}]$) que atingem f_{max} .

2.3. Comparação entre precisão média e discernibilidade

Ao comparar as duas medidas de avaliação, observou-se que a *discernibilidade* é capaz de identificar diferenças entre funções de similaridade consideradas idênticas pela precisão média. Isto se deve ao fato de que a *discernibilidade* leva em conta os escores atribuídos pela função de similaridade e não só as posições dos itens relevantes e irrelevantes no ranking. Além disso, as duas medidas nem sempre concordam sobre qual a melhor função de similaridade. Os casos de conflito não significam que uma das medidas esteja errada na sua avaliação, já que elas foram projetadas para fins diferentes. A precisão média foi projetada para avaliar *rankings* produzidos em resposta a consultas no estilo RI. A *discernibilidade* foi projetada para avaliar *rankings* produzidos em resposta a consultas por abrangência.

3. A Ferramenta

A ferramenta *SimEval*, descrita neste artigo, tem como objetivo avaliar a qualidade das funções de similaridade, aplicadas a um domínio de dados. *SimEval* é uma aplicação web desenvolvida em linguagem JAVA, utiliza o banco de dados MySQL e atualmente está instalada em um servidor Tomcat. A URL para acessar a ferramenta é <http://www.inf.ufrgs.br/~heuser/simeval.html>

A entrada para a ferramenta é um arquivo com a base de dados para a qual a avaliação deve ser feita. O formato deste arquivo é o mesmo usado pela ferramenta “Febrl” [Christen et al. 2004], que gera dados sintéticos para serem usados em aplicações como consultas por abrangência e identificação de duplicatas. Uma das vantagens do formato adotado é que a informação de relevância está contida no próprio arquivo, nos identificadores das *strings*. Estes identificadores informam quais são os registros originais e quais as duplicatas, atribuindo um mesmo número de registro a todas as variações de um mesmo objeto, seguido de um número de ordem. Um exemplo de fragmento do arquivo de entrada é mostrado na Figura 2.

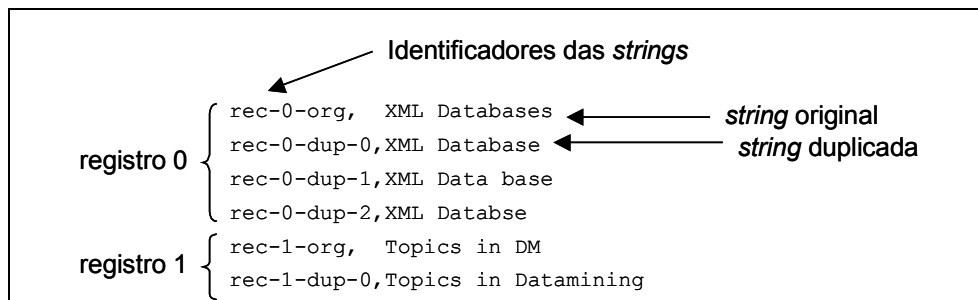


Figura 2 – Formato do arquivo de entrada da ferramenta

A versão atual da ferramenta está projetada para avaliar coleções que contém apenas um atributo. Portanto, se o arquivo de entrada contiver um conjunto de atributos, estes serão concatenados em uma única *string*.

A ferramenta *SimEval* disponibiliza para o usuário todas as 23 funções de similaridade disponíveis na API SimMetrics [Chapman 2007]. Ela permite a geração de um conjunto de amostras de consultas, para um grupo específico de funções de similaridade. Por exemplo, é possível processar informações para 10 amostras, tendo cada amostra 50 consultas, utilizando nessas amostras as funções Soundex, JaroWinkler e Levenshtein. Para visualizar os resultados dos processamentos, a ferramenta possui uma tela onde mostra uma tabela, comparando os resultados de todas as amostras para todas as funções de similaridade, utilizando as métricas da *discernibilidade* e precisão média. Também é possível visualizar os detalhes de cada amostra, mostrando os resultados para cada consulta individualmente e também o *ranking* de cada consulta.

A utilização da ferramenta é extremamente simples, sendo composta por 3 passos. O primeiro passo é a importação do arquivo de dados. O segundo passo é a geração das amostras, no qual se define o número de amostras, o número consultas por amostra e as funções de similaridade a serem utilizadas. O último passo é a visualização dos resultados. Os resultados ficam armazenados em um banco de dados MySQL. Assim, é possível visualizar resultados de amostras previamente geradas.

Na tela onde os resultados dos cálculos de precisão média e *discernibilidade* são exibidos, há uma célula para cada combinação amostra/função. Ao clicarmos na célula, é exibida uma tela com os detalhes da amostra, mostrando por exemplo o S_{rel} e S_{irrel} de cada consulta. Para cada consulta também é possível visualizar seus detalhes, clicando no *link* correspondente. A Figura 3 mostra um exemplo de resultado gerado pela ferramenta. Neste exemplo, quatro funções de similaridade foram aplicadas sobre três amostras de 50 consultas feitas sobre uma base de dados de títulos de artigos. A partir do resultado, pode-se concluir que a melhor função de similaridade para os dados em questão é JaroWinkler visto que obteve valores mais altos tanto de precisão média quanto de *discernibilidade* para as três amostras.

Each cell shows: [discernability] [average precision]						
Similarity Function	Sample 0		Sample 1		Sample 2	
JaccardSimilarity	0.0542	0.85759307652027	0.0942	0.87363973287204	0.0642	0.86033720627134
JaroWinkler	0.3708	0.99144686424686	0.3137	0.97498217564258	0.3008	0.98101468801849
MongeElkan	0.19105	0.96333096861392	0.1227	0.9593668976154	0.14885	0.96266116315913
Soundex	0.2544	0.95389950421182	0.2144	0.95042195660043	0.2244	0.94386189596657

Figura 3 – Visualização dos resultados das amostras

4. Conclusão

Para realizar consultas por abrangência, se faz necessário avaliar qual função de similaridade é mais apropriada para uma aplicação. A ferramenta *SimEval* apresentada nesse artigo visa suprir esta necessidade, utilizando as métricas *discernibilidade* e precisão média como formas de avaliação.

Entre as possíveis melhorias para a ferramenta estão: maior eficiência no processamento dos dados das amostras e a geração de gráficos para auxiliar a exibição dos resultados.

Agradecimentos

Este trabalho é parcialmente financiado pela FAPERGS (projeto PRONEX 0408993) e CNPq (projetos 473310/2004-0, 550.845/2005-4 e Gerindo).

Referências

- Benjelloun, B., et al. (2006). "Generic Entity Resolution in the serf project." *IEEE Data Eng. Bulletin*(June).
- Chapman, S. SimMetrics - Open Source Library <http://sourceforge.net/projects/simmetrics/>
Acessado em: 04/06/2007
- Christen, P., et al. (2004). FEBRL - a parallel open source data linkage system. *Proc. of PAKDD (LNAI 3056)*, Springer: 638-647.
- Cohen, W. W. (2000). "Data integration using similarity joins and a word-based information representation language." *ACM Trans Inf Syst* **18**(3): 288-321.
- Dorneles, C. F., et al. (2004). Measuring Similarity Between Collection of Values. *Proc. of WIDM'04*. Washington, DC, ACM.
- Levenshtein, V. I. (1966). "Binary codes capable of correcting deletions, insertions and reversals." *Soviet Physics Doklady* **10**(8): 707-710.
- Silva, R. d., et al. (2007). "Measuring quality of similarity functions in approximate data matching." *Journal of Informetrics* **1**(1): 35-46.

Litebase: um gerenciador de banco de dados para PDAs com índices baseados em árvores-B

Guilherme C. Hazan¹, Renato L. Novais², Sérgio Lifschitz²

¹SuperWaba – www.superwaba.com.br
guich@superwaba.com.br

²Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
{rnovais, sergio}@inf.puc-rio.br

***Abstract** Litebase is a database management system for PDAs that efficiently uses the available memory, providing good performances. Litebase uses a B-Tree-like index structure, implemented in such a way that it becomes feasible for reduced memory devices. This paper gives an overview of our tool and discusses some of our implementation choices.*

***Resumo.** Litebase é um sistema de gerência de banco de dados para PDAs que usa a pouca memória disponível com eficácia e provê um bom desempenho. Litebase utiliza uma estrutura de índices baseada em árvores-B, implementada de maneira que se torne viável sua utilização em dispositivos de memória reduzida. Este artigo apresenta uma visão geral do Litebase e discute algumas das escolhas de implementação efetuadas.*

1. Introdução

A SuperWaba (www.superwaba.com.br) é uma plataforma de desenvolvimento para computadores portáteis que permite que se criem programas usando a linguagem Java. Ela compreende uma máquina virtual e uma API. Sua maior diferença em relação à plataforma Java da Sun é a API, mais pragmática, buscando principalmente uma redução no tamanho do código ocupado em disco sem diminuir muito suas funcionalidades. A SuperWaba garante também a portabilidade dos programas desenvolvidos com sua API, fazendo com que eles sejam executados, sem modificações, em praticamente todos os sistemas operacionais para dispositivos portáteis, como Palm OS, Windows CE (2.x a 6.x), Symbian, Linux e Blackberry.

A SuperWaba possuía, até 2006, um sistema gerenciador de banco de dados (SGBD) chamado PDBDriver, que permitia que se manipulassem arquivos PDB usando a linguagem SQL padrão, ao invés da manipulação direta de registros. O formato de arquivo PDB (Palm DataBase) é a única forma de armazenamento de dados suportado pelo Palm OS (1.0 a 5.x), e consiste em um arquivo compartimentado em registros, sem suporte à SQL. A SuperWaba portou o formato de arquivo PDB para as outras plataformas, e adicionou o suporte à definição de metadados para tabelas e índices, permitindo o uso de SQL. O formato original dos índices no PDBDriver consiste basicamente em chaves ordenadas, com apontadores para as respectivas tuplas, armazenados em um vetor expansível.

Neste trabalho, apresentamos o Litebase, a nova versão do PDBDriver, que conta em particular com uma nova estrutura de indexação, baseada em uma extensão de árvore-B padrão. São apresentados argumentos consistentes e dados de experimentos que justificam a escolha feita para a implementação. Isto é de certa forma surpreendente dado que estruturas inspiradas em árvores B+ é que são normalmente consideradas em gerenciadores de bancos de dados.

Em outros SGBDs para PDAs, como Oracle Database 10g Lite (Oracle, 2003), DB2 Everyplace (Karlsson et al., 2001), Sql Anywhere Studio (SQL Anywhere, 2001), SQLite (SQLite, 2006), PointBase Micro (PointBase, 2006), é quase um consenso a utilização de árvores B+ ou variações. O Oracle Lite e o SQLAnywhere utilizam árvore B+ para manipulação de índices na sua forma padrão. Em Karlsson et al., 2001, é mostrado que o DB2 Everyplace utiliza árvores PATRICIA para armazenamento e consulta de índices, voltada para busca sobre cadeia de caracteres. Esta estrutura é uma variação de árvore B+ (Bumbulis & Bowman, 2002). Por sua vez, o SQLite utiliza árvore B para manipulação de índices, sem maiores justificativas. Não há também detalhamento algum da implementação, em particular para ambientes de PDA.

Esse artigo segue estruturado da seguinte forma: na Seção 2, apresentamos o Litebase em mais detalhes, justificando a sua estrutura de arquivos e índices. Na Seção 3 discutimos alguns aspectos de implementação do Litebase e sua estrutura de indexação, que trouxeram ganhos de desempenho e espaço de armazenamento. As considerações finais vêm na seção 4, encerrando o artigo.

2. Características do Litebase e Opção por Árvore B

Nesta seção será apresentado o Litebase e um exemplo de aplicação. Em seguida, argumentamos e justificamos a nova estrutura de índice em relação ao PDBDriver, que é baseada em árvores B.

2.1. Litebase

O Litebase é um sistema gerenciador de banco de dados para utilização em PDAs. Ele procura atender aos três requisitos básicos que aplicações para estes tipos de dispositivos devem satisfazer: portabilidade, desempenho e economia de espaço.

O Litebase, atualmente na versão 1.x, segue o padrão da linguagem SQL para a manipulação dos dados e dá suporte aos principais comandos da linguagem. A API do Litebase permite a execução de comando de definição e manipulação de dados, como, *creates*, *inserts*, *updates*, *deletes* e *select*; utilização de funções de agregação; utilização de *order by*; criação de índices; chave primária; funções sobre determinados tipos de dados, como por exemplo, *lower*, *upper*, *day*, *year*, entre outras; utilização de *prepared statements*; além de funcionalidades específicas de PDAs, como por exemplo, suporte à criação dos dados diretamente nos cartões de memórias; API para facilitar a sincronização dos dados, etc. A Listagem 1, a seguir, apresenta um exemplo de utilização do Litebase dentro de um programa SuperWaba.

A versão 2.0 do Litebase está em fase final de desenvolvimento e testes. Com lançamento previsto para Novembro de 2007, suporta novas funcionalidades, dentre elas, pode ser citado o suporte a valores *Default*, índices compostos e transações.

SBBD 2007

IV Sessão de Demos

Listagem 1 – Exemplo de aplicação utilizando o Litebase

```
// Abre uma conexão com o banco
LitebaseConnection driver = LitebaseConnection.getInstance("Test");
// cria uma tabela e um índice
try{
    driver.execute("CREATE TABLE person (name char(30), salary double, age
    int)");
    driver.execute("CREATE INDEX idx_name ON person(name)");
} catch (AlreadyCreatedException e) {}
// executa insertes/deletes e updates
driver.executeUpdate("INSERT INTO person VALUES ('Indira Gomes', 1000.0,
26)");
driver.executeUpdate("INSERT INTO person VALUES ('Jener e Zenes',
2000.0, 50)");
driver.executeUpdate("UPDATE person SET salary = 1500 WHERE name LIKE
'Indira%'");
driver.executeUpdate("DELETE FROM person WHERE age > 40");
// Cria um Result set a parti de um consulta na tabela
ResultSet rs = driver.executeQuery("SELECT * FROM person");
```

O Litebase roda atualmente com a linguagem SuperWaba. A construção de *drivers* para comunicação com outras linguagens também já está em processo de desenvolvimento.

2.2. Árvore B ou B+ ?

O desempenho e a economia de espaço proporcionado pelo Litebase estão diretamente associados à nova estrutura de índices considerada. Para o desenvolvimento dos índices do Litebase foram estudadas e avaliadas algumas das estruturas mais conhecidas na literatura, incluindo as árvore-B e árvore-B+. Uma descrição dessas estruturas e seus algoritmos de manipulação podem ser encontrados em (Elmasri and Navathe, 2000).

Em muitos dos livros-texto de bancos de dados encontram-se vários argumentos em favor de árvores B+ para explicar implementações de índices como estruturas de acesso aos dados auxiliares. Entre outros, a árvore B+ é interessante quando o espaço ocupado pela chave de busca é muito menor do que o tamanho da tupla à qual ela pertence. Isto porque as chaves de pesquisa podem ser repetidas nos nós internos da árvore B+, (além dos nós folha) e, por serem pequenos, não causariam um impacto grande no tamanho dos índices. Porém, em casos reais é possível que um usuário crie um índice sobre um atributo com muitos caracteres (por exemplo, `char(200)`). Nesses casos, a chave que será inserida no índice é o próprio campo, impactando fortemente no tamanho do índice, devido à repetição dos valores nos nós internos da árvore B+. De uma forma geral, uma árvore B+ exige um arquivo de tamanho maior em relação a uma árvore-B.

Dadas essas considerações, poderia ser imediata a escolha da árvore-B para implementar índices em PDAs, não fosse o fato da árvore B+ permitir que uma pesquisa seqüencial ordenada seja realizada com eficiência. Além disso, os dados dentro do mesmo bloco são normalmente armazenados de forma clusterizada, reduzindo o custo de acesso aos mesmos quando o banco é armazenado em disco. Isto é interessante, por exemplo, para consultas de faixa de valores em atributos que possuem índices associados.

Para contornar a questão de busca seqüencial ou sobre faixa de valores na árvore-B, implementamos um algoritmo de percurso ordenado nos índices da árvore B. Em dispositivos como PDAs, o acesso é sempre feito de forma direta, devido ao tipo de

mídia usado no armazenamento dos dados. Logo, a vantagem do acesso clusterizado aos dados armazenados provido pela árvore B+ perde o sentido nestes dispositivos. Considerando ainda a simplicidade de implementação, as árvores-B se mostram mais interessantes para dispositivos com recursos limitados como os PDAs, justificando assim a nossa escolha. Os resultados exibidos na seção 3 reforçam nossos argumentos.

3. Aspectos de Implementação

Em um levantamento feito pela SuperWaba com seus clientes, foi observado que mais de 99% das tabelas possuem até 100 mil tuplas, e os 1% restantes possuem até 1 milhão de tuplas. Essa distribuição é plausível para dispositivos com memória limitada, como PDAs. Com base nestas informações, relacionamos abaixo algumas otimizações aplicadas às estruturas de índices baseadas em árvores-B do Litebase, que resultaram em economia de memória e ganho no desempenho.

A primeira modificação em relação à árvore B padrão está relacionada às estruturas internas do índice. Inicialmente remodelamos essas estruturas de forma que ocupassem o menor espaço possível. Em uma árvore B de ordem N temos, em cada nó, N chaves e N+1 ponteiros. Em termos de implementação, cada nó precisa armazenar um contador indicando quantas chaves estão presentes, sendo que este contador varia de 0 à ordem da árvore. Normalmente, as implementações utilizam 4 bytes - uma palavra nas arquiteturas de 32 bits - para armazenar o contador, o que já endereça mais do que o total de tuplas que cabem na memória dos PDAs.

Em um primeiro momento, procuramos diminuir o tamanho deste contador. Difícilmente uma ordem maior que 65536 (2 bytes) é usada, pois se em um PDA pouquíssimas tabelas chegam a 100 mil tuplas, uma ordem dessa magnitude faria com que o número de nós da árvore fosse muito pequeno, não justificando a utilização da estrutura de árvore. Através de testes empíricos em PDAs, foi possível observar que árvores de ordem 50 ou superior resultam em perda de desempenho e aumento do espaço utilizado, não importando o número de nós persistidos (Figura 1). Desta forma, definimos que o espaço utilizado para armazenar a ordem e, consequentemente, os contadores, será de 2 bytes, o que é suficiente para dispositivos de memória limitada.

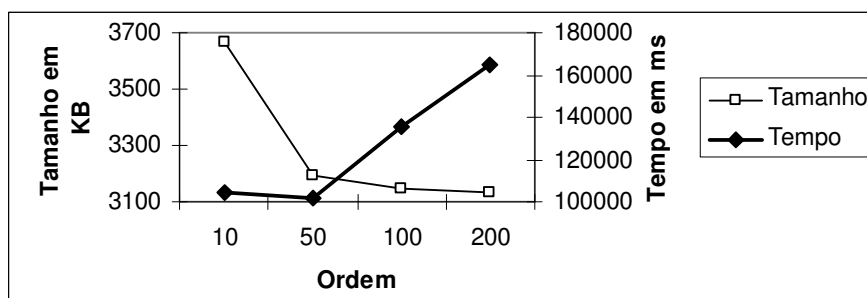


Figura 1 – Gráfico de Ordem x Tamanho/tempo (valores randômicos)

Outra decisão de implementação está relacionada ao tamanho dos ponteiros para as tuplas na tabela. Optamos por utilizar 3 bytes para cada ponteiro, alcançando um endereçamento aproximado para 16 milhões de tuplas, bem maior que o descrito antes.

Na Figura 2 pode ser observado um exemplo comparativo entre as árvores B e B+ tradicionais e o Litebase. No gráfico é possível observar a quantidade de *bytes* gastos em relação à taxa de ocupação dos nós. Neste exemplo simulamos a inserção de 1000 nós em uma árvore de ordem 10, sendo que, para as árvores B e B+, utilizamos 4 bytes para armazenar a ordem, os contadores e os ponteiros. Com as decisões propostas acima, obtivemos ganhos em bytes de aproximadamente 20%.

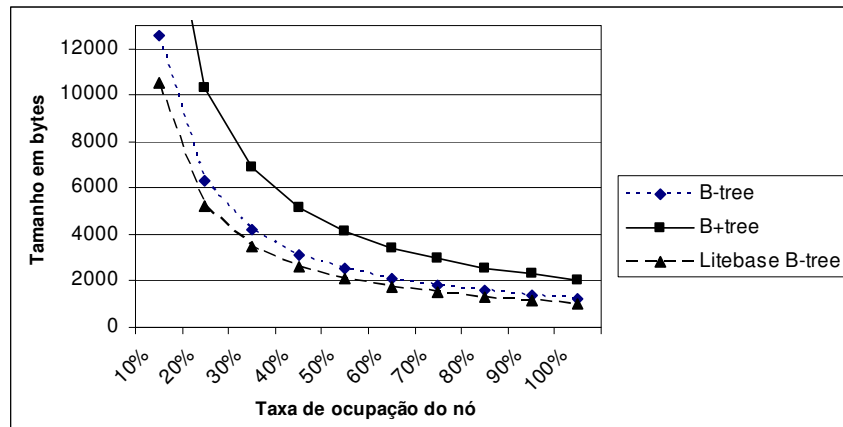


Figura 2 – Gráfico comparativo de espaço ocupado pelas árvores

Outra otimização no código implementado refere-se à forma com que são tratadas as repetições de chaves na árvore. Diferentemente de sugestões de implementação encontradas na literatura, não armazenamos os valores repetidos na própria árvore de índice. No Litebase, os valores repetidos, como por exemplo em índices secundários, são armazenados em um arquivo separado. Este arquivo contém as repetições das chaves ligadas através de uma lista encadeada. Quando um nó é criado no índice, são armazenados o valor do atributo de índice e o ponteiro para a posição física da tabela. Quando há repetição, o valor do atributo do índice é mantido no nó do índice e, no lugar do ponteiro para a posição física da tabela, é armazenado um ponteiro para uma posição do arquivo de repetição onde iniciará uma lista encadeada com todos os ponteiros daquele valor. Quando não há repetição (índices chave primária) o arquivo de repetição é desprezado. Na Figura 3 é apresentada a estrutura geral dos índices do Litebase.

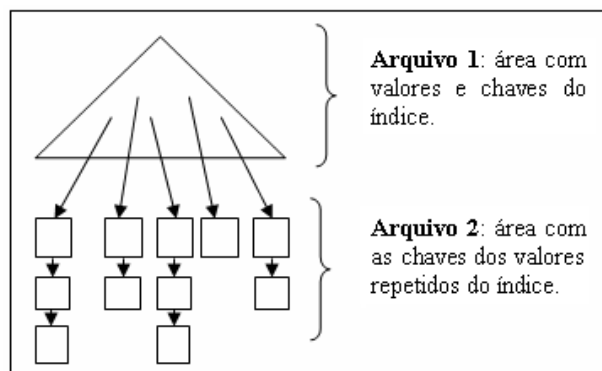


Figura 3 – Representação gráfica da estrutura do índice do Litebase

SBBD 2007

IV Sessão de Demos

Outras otimizações, pequenas, mas não menos relevantes, foram realizadas no Litebase:

- Acessos à mídia de armazenamento são feitos em blocos fazendo uso do *cache*.
- Quando uma chave repetida é inserida, acrescentando um novo valor à lista encadeada de valores, apenas a chave (e não todo o nó) é atualizada no disco.
- Um sistema de *cache* é usado para armazenar alguns nós já visitados, sendo o nó reutilizado jogado para o fim da fila. O *cache* é bastante interessante para fins de desempenho, mesmo quando a quantidade de nós for pequena.

Outro aspecto de implementação está relacionado às remoções de valores na árvore. Optamos por realizar uma remoção lógica dos dados na tabela, permitindo que, durante uma sincronização, um servidor seja informado das tuplas que foram removidas. Ao término da sincronização, o Litebase faz a remoção física, recriando em seguida os índices. Assim, a remoção nos índices também é feita de forma lógica, eliminando a implementação do algoritmo de remoção física para árvores B, que é bastante custosa.

4. Considerações finais

A necessidade de uma gerência de dados eficiente em dispositivos portáteis cresce juntamente com o seu uso. Nesse cenário, apresentamos o Litebase, um gerenciador de bancos de dados para PDAs com uma estrutura de índice apropriada para as características desses dispositivos. Foi apresentada a estruturação dos índices do Litebase, além de otimizações realizadas em seu código, que fazem dele um banco pequeno, eficaz e portátil. O leitor pode encontrar no site do SuperWaba uma versão pública do Litebase para avaliação em uso e testes de desempenho.

Referências

- [Bumbulis & Bowman, 2002] Peter Bumbulis, Ivan T. Bowman. A compact B-tree. SIGMOD Conference 2002.
- [Elmasri and Navathe, 2000] Elmasri, R.; Navathe, S. Fundamentals of database systems. 3rd Edition. Addison-Wesley, 2000.
- [Karlsson et al., 2001] J.S. Karlsson, A. Lal, C. Leung, and T. Pham. IBM DB2 Everyplace: A Small Footprint Relational Database System. In Proc. of 17th International Conference on Data Engineering, 2001.
- [Oracle, 2003] Oracle Database Lite InDepth, An Oracle White Paper, Dec 2003.
- [PointBase, 2006] PointBase. Disponível em <http://www.pointbase.com>. Último acesso em 11/09/2006.
- [SQL Anywhere, 2001] SQL Anywhere 5.x Cost-Based Query Optimizer, An SQL Anywhere White Paper, Sybase, Workplace Database Division, Sybase, Inc., California, USA, 2001.
- [SQLite, 2006] The Architecture Of SQLite, Disponível em <http://www.sqlite.org/arch.html>. Último acesso em 11/09/2006.

5SQual - A Quality Assessment Tool for Digital Libraries

Bárbara L. Moreira¹, Marcos A. Gonçalves¹,
Alberto H. F. Laender¹, Edward A. Fox²

¹ Department of Computer Science, Federal University of Minas Gerais,
31270-901 - Belo Horizonte - MG - Brazil

²Department of Computer Science, Virginia Tech,
Blacksburg - VA, 24061 - EUA

`{barbara, mgoncalv, laender}@dcc.ufmg.br, fox@vt.edu`

Abstract. *This work describes 5SQual, a quantitative quality assessment tool for digital libraries based on the 5S framework. 5SQual aims to help administrators of digital libraries during the implementation and maintenance phases of a digital library, providing ways to verify the quality of digital objects, meta-data and services. The tool has been designed in a flexible way, which allows it to be applied to many systems, as long as the necessary data is available. To facilitate the input of these data, the tool provides a wizard-like interface that guides the user through its configuration process.*

1. Introduction

Digital libraries (DLs) are complex systems that offer information through content and services designed for specific communities of users. Since DL evaluations can be expensive and diverse (when evaluating quality, people interested in DLs focus on different aspects [Fuhr et al. 2001]), usually quality evaluations are conducted just when the system presents failures and the administrator should interfere immediately, contradicting the fact that evaluation should be a continuous process throughout the life cycle of a computer-based system [Borgman 2003]. For DLs to be more reliable and easier to maintain, it is necessary to find ways to perform, in practice, cost effective and automated quality evaluations of these systems.

In this work, we describe 5SQual, a tool we have developed to automatically assess various quantitative aspects of a DL according to the 5S quality model [Gonçalves et al. 2007], a formal quality model for DL evaluation, built on top of the 5S framework. 5S, which stands for *Streams, Structures, Spaces, Scenarios* and *Societies*, is a theoretical framework to formally define DLs [Gonçalves 2004, Gonçalves et al. 2004]. The quality model provides a theoretical basis for developing and quantifying quality numeric indicators for 22 quality dimensions. It is important to notice that, despite some recent effort on DL evaluation [Borgman 2003], to the best of our knowledge 5SQual is the first tool that provides such a functionality.

However, 5SQual does not aim to perform a complete system evaluation, since the evaluation of many quality aspects of a DL would demand resources that cannot be automatically handled (e.g., user participation). Its main objective is to assist administrators

while designing and maintaining digital libraries, thus providing support for automatic and periodical evaluations that would help diagnose problems and possible improvements and demonstrate the system evolution over time.

2. 5SQual Overview

The 5SQual development has been initially based on a subset of dimensions defined in the 5S quality model. These dimensions cover the evaluation of digital objects, metadata and services. The subset includes:

- Regarding digital objects - *accessibility* (given an actor x, the accessibility of a digital object is given by the percentage of the streams of the object that x is allowed to access), *significance* (indicates the importance of digital objects according to a certain factor, such as number of accesses, citations, downloads, etc.), *similarity* (estimates how related two digital objects are, e.g., we can use co-citation measure as an indicator for this dimension), and *timeliness* (indicates how up-to-date the objects in the DL are. The numeric indicator for this dimension is the elapsed time between, for instance, the creation time and the current time).
- Regarding metadata - *completeness* (reflects how many metadata attributes of a standard schema have values specified) and *conformance* (percentage of the attributes in the metadata that follows the rules defined by a standard schema).
- Regarding services - *efficiency* (reflects the response time of the services) and *reliability* (is proportional to the frequency of successful operations, among the total number of executions of a service).

These dimensions have been chosen for a first implementation of 5SQual because the respective numeric indicators are user independent and objective enough to allow an automatic evaluation. In the future, other dimensions and numeric indicators can be added to the tool by simply implementing additional classes and methods.

Figure 1 shows the 5SQual architecture. The necessary information for the evaluation resides in the DL and should be retrieved through the DL application layer.

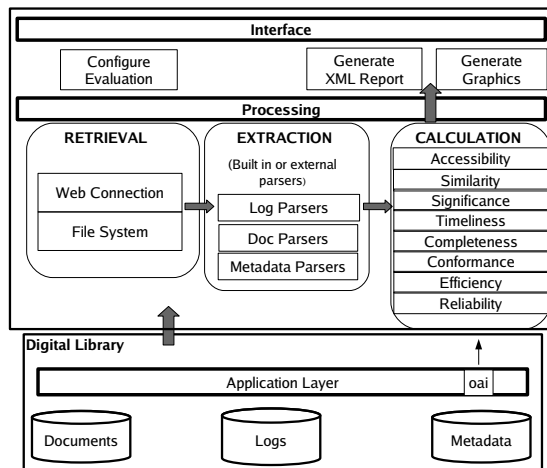


Figure 1. 5SQual Architecture

The **Processing Layer** consists of three modules: the **Retrieval module**, which obtains data on the Web or from the local file system by collecting log files, metadata (via OAI-PMH) or documents; the **Extraction module**, which uses parsers to extract the necessary data from the collected files and converts them to a standard format required for each dimension (the set of built-in parsers includes content parsers, e.g., PDF and PS files, specific metadata format parsers, e.g., Dublin Core and RFC1807, specific log format parsers, e.g., the XML log format [Gonçalves et al. 2003], and user-specified parsers for other file formats); and the **Calculation module**, which implements the set of numeric indicators for each quality dimension. The **Interface Layer** includes the **Configure Evaluation** module, which stores the parameters defined for the evaluation, and the modules **Generate XML Reports** and **Generate Graphics**, which generate the outputs of the evaluations (XML reports and charts).

3. 5SQual Inputs and Outputs

3.1. Inputs

Before using 5SQual, a user, typically the administrator of a DL, has to configure the parameters for the evaluation. Some of these parameters are mandatory and present a defined set of values. In this section, we show some screenshots of the setup wizard that was developed to guide the user through the necessary configuration steps. Once the configuration is done, an XML file with the configured parameters is generated and can be imported later through the same interface to repeat the evaluation, making it easier for the user to analyse the system over time.

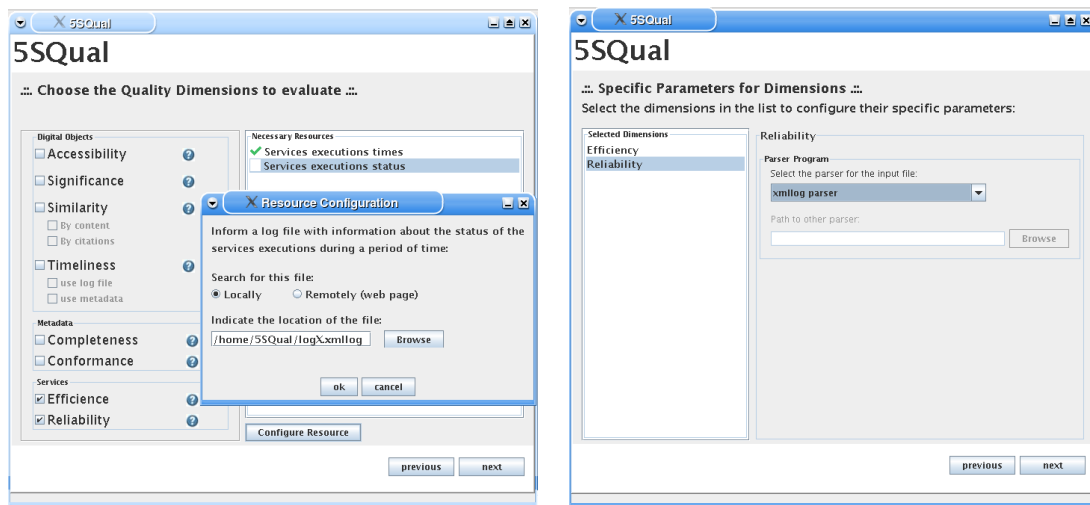
In order to configure an evaluation to be carried out using 5SQual, a user should follow six steps. In the first step, the user has two options: (1) start a new evaluation from the beginning by following all the steps to configure the necessary parameters or (2) import a previously generated file with all the parameters already specified. If she chooses the 1st option, in the second step, the user identifies the evaluation by giving the name of the DL that is being assessed and adding an optional description.

Figure 2 shows two of the wizard screens, corresponding to the third (Figure 2(a)) and fourth (Figure 2(b)) steps of the configuration, when the user selects which dimensions to evaluate and indicates the necessary resources (where 5SQual should find data for the evaluation) and parameters (how to extract information from the collected data and how to calculate the indicators of the selected dimensions).

After specifying the parameters for the evaluation of each desired dimension, on the fifth step the user should choose where to save the corresponding files that will be generated by the tool as the result of the evaluation (an XML report file and graphic charts for the selected dimensions) and the XML file with all the input parameters which can be imported in the future for a re-evaluation. On the sixth step, the user has the option to execute the evaluation.

3.2. Outputs

To exemplify the kind of output that 5SQual produces, we show some charts generated as a result of the evaluation of the Virginia Tech's Digital Library of Electronic Theses and



(a) Selecting Dimensions and Indicating Resources

(b) Specifying Parameters

Figure 2. Interface Sample

Dissertations (VT-ETD)¹, according to the dimensions *Timeliness*, *Accessibility* and *Completeness* (see Figure 3). Due to the lack of space, we just comment about the evaluation of these 3 dimensions, without showing the resulting evaluation XML report.

Accessibility: The VT-ETD metadata presents the *rights* field, with information about the policy for accessing the digital objects from the DL. Objects can be restricted (available only to the VT community), unrestricted (public), or mixed (parts of it are public and other parts are restricted). For quantitative evaluation, we associated a value of accessibility to each one of these categories - unrestricted: 1, restricted: 0 and mixed: 0.5. To define these values, we considered the view of an actor that does not belong to the VT community. The chart obtained from the 5SQual evaluation, shown in Figure 3(a), presents the number of objects with restricted, unrestricted, and mixed access. From the corresponding generated XML report, it is possible to get the identifiers of the documents for each access category. As we can see from the chart, almost 35% of the ETDs have restricted access outside of the university environment, which may reveal an apprehension from some of the new graduates that free availability of the material may cause problems in future attempts for publishing the result of their research as scientific papers or patents. Also, the small number of restricted ETDs (less than 2%) may be due to a lack of knowledge by these graduates about the possibility of releasing only parts of the ETDs they desire, an interesting mechanism that can, at the same time, protect part of the content while publicizing some of the results.

Completeness: For calculating *completeness*, we retrieved the VT metadata records which follow the Dublin Core format. The chart in Figure 3(b) shows that there are four distinct *completeness* values in the catalog, indicating that there are four groups of records with the same number of fields. The largest group (7,470 records) presents the highest level for *completeness* in the catalog. The records of this group include 13 of the 15 fields defined by the Dublin Core format, which corresponds to a *completeness* value near to 0.87. Looking at the other groups, 24 records present *completeness* equal to 0.67,

¹<http://scholar.lib.vt.edu/theses/>

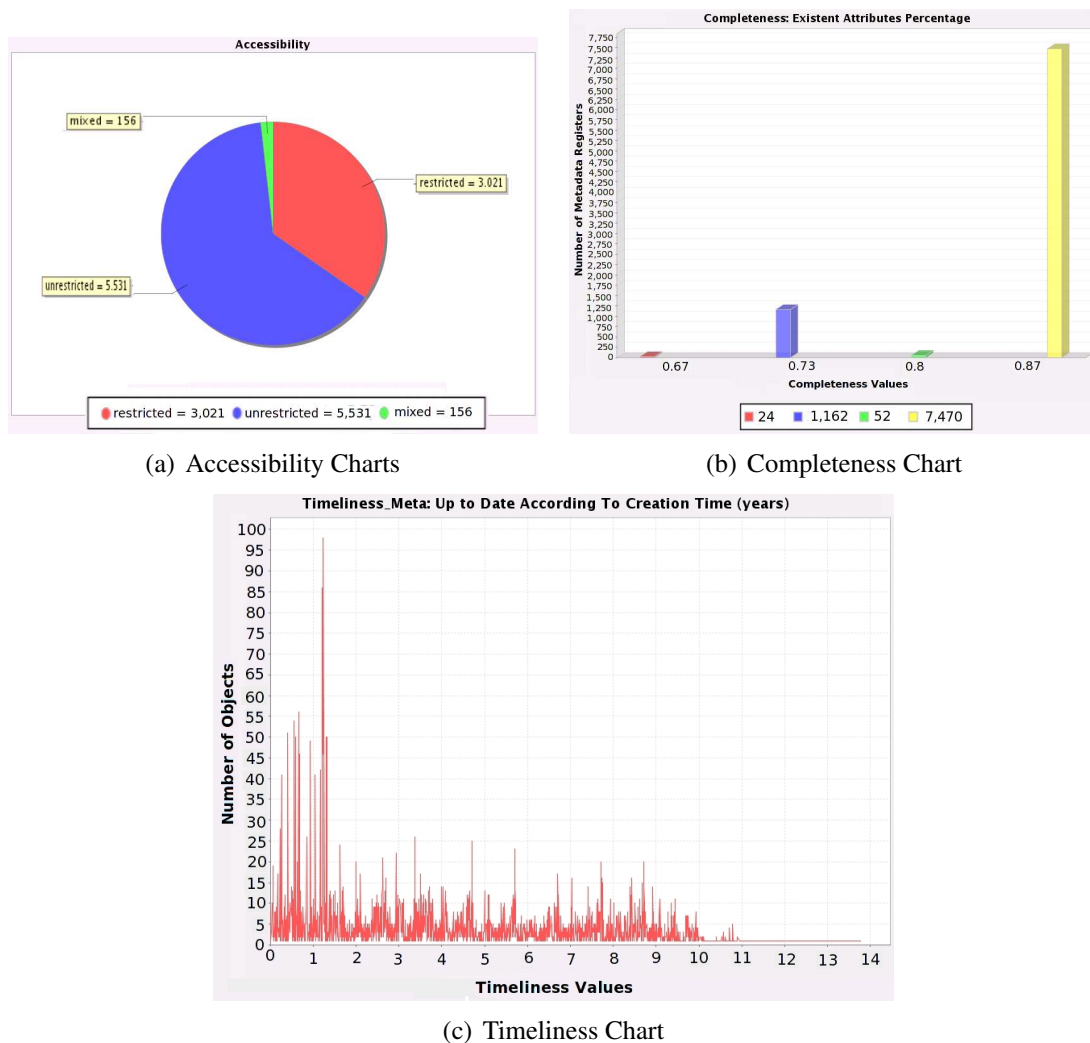


Figure 3. Output Examples

1,162 equal to 0.73, and 52 equal to 0.80. No record is totally complete. Retrieving the metadata records using their corresponding identifiers in the XML report, it is possible to check which fields are missing. This means that, analysing this dimension, the administrator of a DL has a clear idea of what is missing from its catalog and therefore of the required work for complementing it.

Timeliness: The creation time of the digital objects was extracted from the *date* field of the VT ETD metadata records. *Timeliness* in this case was measured in years, given by the difference between the current time and the obtained creation time of the corresponding digital object. Figure 3(c) shows the number of items concentrated under each of the shown timeliness values that were calculated based on the current time (date of this evaluation was January 9, 2007). We can see that objects have been continuously created in this DL over the last 10 years and that many objects (almost 100) were inserted on the same date, approximately 1.3 year ago (when scanning of the backfile was speeding up, according to information provided by the DL administration). We also notice that, in the early days of this DL, there was a very stable insertion pattern over the years, which might indicate that the insertion of new objects into the collection was related to some

academic events. However, in the last three years this pattern has changed, increasing not only the number of objects per insertion, but also the frequency in which these insertions take place.

4. Conclusions

5SQual has been developed to help administrators when building and maintaining a DL, thus providing data to guide its design, development and improvement continuously. To evaluate how useful the tool is, we conducted an interview with potential users (real DL administrators) to assess their expectations regarding its functionality. This interview has shown that the tool can be really useful while maintaining a DL, but that other aspects should also be covered in next versions, such as facilities for evaluating higher level components (e.g., collections and catalogs) and the availability of additional parsers.

The tool requires certain sets of data to evaluate each quality dimension. Though a DL may not produce all the necessary information to assess the complete set of dimensions implemented by 5SQual, it still can have many of its aspects evaluated. This suggests that DL administrators should be stimulated to produce more data in order to carry out broader evaluations, therefore recording more systematically their system's behavior over time.

Acknowledgments.

This work was partially supported by the 5S-QV project (MCT/CNPq/CT-INFO grant number 551013/2005-2) and by the project grant NSF DUE-0121679.

References

- Borgman, C. L. (2003). NSF report on DELOS Workshop on DL evaluation. Technical report, Hungarian Academy of Sciences Computer and Automation Research Institute (MTA SZTAKI).
- Fuhr, N., Hansen, P., Mabe, M., Micsik, A., and Solvberg, I. (2001). Digital libraries: A generic classification and evaluation scheme. In *Proceedings of the European Conference on Digital Libraries*, pages 187–199, Heidelberg. Springer.
- Gonçalves, M. A. (2004). *Streams, Structures, Spaces, Scenarios, and Societies: A Formal Framework for Digital Libraries and Its Applications*. PhD thesis, Virginia Tech CS Department, Blacksburg, Virginia. URL - <http://scholar.lib.vt.edu/theses/available/etd-12052004-135923/>.
- Gonçalves, M. A., Fox, E. A., Watson, L. T., and Kipp, N. (2004). Streams, structures, spaces, scenarios, societies (5S): A formal model for digital libraries. *ACM Transactions on Information Systems*, 22(2):270–312.
- Gonçalves, M. A., Moreira, B. L., Fox, E. A., and Watson, L. T. (2007). What is a good digital library? - defining a quality model for digital libraries. To appear in *Information Processing and Management*.
- Gonçalves, M. A., Panchanathan, G., Ravindranathan, U., Krowne, A., Fox, E. A., Jagodzinski, F., and Cassel, L. (2003). The XML log standard for digital libraries: analysis, evolution, and deployment. In *Proceedings of the 3rd ACM/IEEE-CS joint conference on digital libraries*, pages 312–314, Washington, DC, USA. IEEE Computer Society.

WhizKEY: A wizard-based tool for installing digital libraries*

Rodrygo L. T. Santos, Marcos André Gonçalves, Alberto H. F. Laender

¹Department of Computer Science – Federal University of Minas Gerais
31270–901 Belo Horizonte MG, Brazil

`{rodrygo,mgoncalv,laender}@dcc.ufmg.br`

Abstract. *Several initiatives have been taken in the last decade towards the development of reusable software frameworks to serve as a basis for the construction of digital libraries. The instantiation of these frameworks, however, is usually not or barely assisted and often requires specialized computer skills from their users, what is clearly undesirable in a broader context of digital library utilization. In this paper, we present WhizKEY, a wizard-based tool for assisting users in the installation of digital libraries. The tool has been designed to provide users with a unified environment for constructing and personalizing digital libraries from possibly several different software frameworks, shielding these users from the intricacies of the underlying installation process.*

1. Introduction

In the past few years, the development of digital libraries (DLs) began to incorporate consolidated practices from software engineering, such as the utilization of software frameworks, sometimes implemented as networks of software components – independent units that can be composed together in order to build more complex systems. In the DL field, in particular, many software frameworks have been developed. The best known examples are Greenstone (University of Waikato and UNESCO, 1997), EPrints (Southampton University, 2000), DSpace (MIT and HP, 2002), and Fedora (Cornell University and University of Virginia, 2003). Usually built from scratch, DLs began to be constructed upon functional building blocks, what resulted in more efficient, reliable, and low-cost development projects.

On the other hand, the instantiation of these software frameworks into running systems is not a trivial task, particularly for non-specialist users. For such, in this paper, we present *WhizKEY – Wizard-based blocK Ensemble Yelder*, a new tool we have developed and evaluated to assist users in the DL construction task [Santos et al. 2006]. WhizKEY aims at allowing DL designers to concentrate on the DLs being built rather than on the intricacies of the underlying installation process. It implements a setup workflow, driving the DL designer along well-defined steps towards the construction of fully-functional systems. This workflow can be fully customized without any code rewriting in order to allow the construction of systems based on different software frameworks.

The remainder of this paper is organized as follows. Section 2 presents some background on the task of instantiating software frameworks into running systems and discusses the inherent complexity involved in building DLs from these frameworks. Section 3 overviews the architecture and functioning of WhizKEY and shows how it could be adapted for the installation of different software frameworks. Finally, Section 4 presents concluding remarks and directions for future work.

*Research partially funded by MCT/CNPq/CT-INFO project 5S-VQ and by FAPEMIG project 5S+.

2. Background and Motivation

Following Pree's terminology [Pree 1994], software frameworks consist of frozen spots and hot spots. On the one hand, *frozen spots* stand for the ready-to-use portion of the framework, i.e., the part that remains unchanged in any instantiation of the framework. On the other hand, *hot spots* represent those parts that need to be adjusted in order to build tailored systems. These adjustments may require different procedures depending on the underlying framework. Here we differentiate two tasks, namely instantiation by implementation and by installation. The *instantiation by implementation* requires programmers to add their own code to specialize the hot spots in the framework. In the *instantiation by installation*, in turn, no adaptation to the underlying framework implementation is required. In this case, the implementation comprises only frozen spots, which need to be generic enough to allow the instantiation of the framework through the setup of parameterized hot spots, mainly in the form of configuration documents. Here we deal only with the instantiation by installation.

The installation task comprises the placement and configuration of several artifacts within the underlying framework: program files, installation directories, configuration documents, environment variables, links (or shortcuts), etc. Besides being laborious, this task is also error-prone: several different parts of the framework must be correctly set up and potentially interdependent parts must be correctly assembled together in order to ensure the proper functioning of the instantiated system. Despite this inherent complexity, existing software frameworks for building DLs generally require users to accomplish the installation task either manually or by interacting with simple command-line scripts. Both alternatives, however, seem inadequate in a broader context of digital library utilization, in which these systems are to be deployed for organizations with disparate requirements and to be administered by users with varying computer skill levels. For such, we resorted to a user assistance technique commonly employed for installation tasks: a wizard. According to Ekenstierna and Ekenstierna [Ekenstierna and Ekenstierna 2002], a wizard can be very useful, above all when tasks are performed infrequently, which is the case here.

Despite the vast amount of general-purpose wizard generation environments available, both proprietary (e.g., InstallShield, InstallAnywhere, Wise) and free (e.g., NSIS, Inno Setup, WiX, IzPack), none has been targeted to the installation of digital libraries as WhizKEY has, as discussed in the next Section.

3. WhizKEY Overview

WhizKEY's architecture extends the MVC (Model-View-Controller) [Burbeck 1987] framework with the addition of a persistence layer. The *model* layer was designed with generality in mind in order to support the definition of a digital library as provided by different software frameworks. This design was inspired on the definition of a digital library taken from a formal model for digital libraries [Gonçalves et al. 2004]. Accordingly to this model, a typical digital library is informally defined as a set of mathematical components (e.g., collections, services), each component being precisely defined as functional compositions or set-based combinations of formal constructs from the model. Our configuration model was devised regarding the components that make up a digital library as defined by this formal model as configuration *blocks*. By a configuration block we mean an instantiated piece of software whose behavior is defined as a set of user-configurable

parameters. This piece of software is an abstraction and may represent, for instance, a software component provided by the underlying framework, but may also span functionally distinct parts of the framework.

The diagram in Figure 1 shows a simplified view of WhizKEY's architecture. Accordingly to the MVC framework, the user interacts with the view layer (white circle, in the diagram). This interaction is handled by the controller layer (dark-gray circles), which performs the corresponding modifications to the model layer (light-gray circles). The model layer, in turn, notifies the view layer to update itself based on the new model state. The model layer is divided into two parts: types (solid light-gray circles) and instances (dashed light-gray circles). The *types* model the underlying framework with its constituent parts. As shown in the diagram, a *Framework* comprises a set of *Components*, being each component a representation of a piece of the underlying software framework. Each component comprises a set of *Parameters*, semantically organized into *ParameterGroups*. Both components and parameters may be declared dependent on other components and parameters, respectively, as discussed below. Also, each component may yield *instances* of itself, which we call *Blocks*. A set of blocks correspond to a *Configuration*.

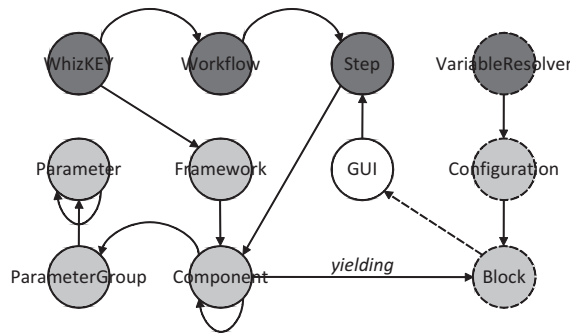


Figure 1. WhizKEY's architecture overview

The *persistence* layer is responsible for loading and saving blocks. Besides that, it is up to this layer to perform deployment tasks such as setting environment variables and preparing databases that support the execution of some components. Its working scheme is based on two XML documents: a configuration log and a framework descriptor. The first and simpler one, the *configuration log*, comprises information about the currently configured digital libraries running in the system. The second document, the *framework descriptor*, details every component in the framework, including all configuration parameters associated to them, as exemplified in Figure 2 for the WS-ODL's Searching service [Roberto et al. 2006]. The definition of this component includes the workflow *step* in which its instances should be configured as well as cardinality constraints (*min-max*) on the number of instances allowed for this component. It also includes a *label* and a *description* to be displayed by the view layer and a set of references to other components on which it is functionally dependent (e.g., in Figure 2, *search* instances are dependent on *admin* instances). The parameters associated to this component are grouped together according to their semantics. For instance, the *Indexation* group lists a set of parameters responsible for managing the inverted files used by the Searching service. The specification of each configuration parameter contains a set of flags on whether it could be edited (*fixed*), displayed (*visible*), and saved (*persistent*). It also carries cardinality (*min-max*)

constraints on the values it can be assigned and, in case of multi-valued parameters, a flag on whether these values should be hierarchically organized (*flat*). A parameter also carries XPath-based *path* entries that uniquely locate it in each of its source documents.

```
<component id="search" min="0" max="1" step="services">
  <label>Searching</label>
  <description>
    A service that provides both structured and keyword-based search.
  </description>
  <requirements>
    <component ref="admin"/>
  </requirements>
  <configuration>
    <group id="indexation">
      <label>Indexation</label>
      <description>Inverted file configurations</description>
      <parameters>
        <parameter id="fieldstoindex" fixed="false" visible="true" persistent="true"
          flat="true" min="1" max="unbounded" separator="|">
          <label>Fields to Index:</label>
          <description>A list of fields to be indexed.</description>
          <sources>
            <path src="fedoraCfg">/config/search/fieldstoindex</path>
          </sources>
          <constraints>
            <constraint type="default">dc:title|dc:subject|dc:description</constraint>
            <constraint type="enumeration" message="Value outside domain">
              eval('./md_fields -f #md_preffix')
            </constraint>
          </constraints>
        </parameter>
        ...
      </parameters>
    </group>
  </configuration>
  <deployment>
    <deploy>cp -r #baseDir/search #libraryHome</deploy>
    <undeploy>rm -rf #libraryHome/search</undeploy>
  </deployment>
</component>
```

Figure 2. Fragment from the WS-ODL framework descriptor document

For validation purposes, each parameter defines a set of *constraints* on the values it can be assigned. For instance, in Figure 2, the *fieldstoindex* parameter defines two constraints: a *default* constraint, which assigns the parameter a default value when no value is set by the user; and an *enumeration* constraint, which defines a valid domain for the values set by the user. Besides these two constraints, WhizKEY implements four others: *unique*, which verifies whether the parameter value is unique in a given domain; *check*, which evaluates a logic expression possibly based on the parameter value; *pattern*, which matches the parameter value against a given regular expression; and *resource*, which verifies whether the parameter value corresponds to a valid resource (e.g., a file pointer or a URL). Besides that, parameters may be declared complex, i.e., defined as a composition of other parameters. Also, pointers to the values of previously configured parameters in the workflow sequence may be used as terminals in component specifications, as well as values evaluated from external script calls. This way, for instance, the domain of a given parameter may be constrained based on values set in other parameters or even on values determined from outside the framework being configured (e.g., operating system properties). To illustrate this, in Figure 2, the enumeration constraint on the values of the *fieldstoindex* parameter is determined by the values returned by the *md_fields* script, which takes a previously configured parameter (*md_preffix*, associated to a component providing

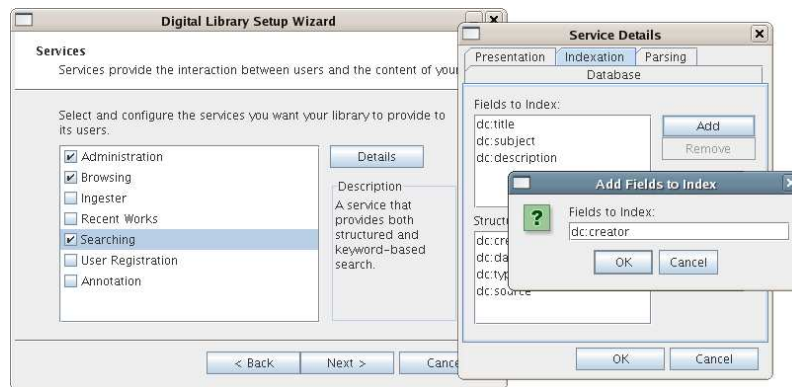


Figure 3. Configuring WS-ODL's Searching service

WS-ODL's repository infrastructure) as input and outputs the fields corresponding to the metadata format it represents (e.g., Dublin Core fields).

Finally, deployment procedures (e.g., preparing the system environment and placing framework artifacts into proper destination directories) are externalized from the implementation of WhizKEY, since they can be rather specific for different frameworks. For such, they are encapsulated as a sequence of *deploy* and *undeploy* procedures in the framework descriptor document (Figure 2). These procedures can include virtually any command-line executable instruction. Thus, specializing WhizKEY to assist the installation of different software frameworks can be done just by providing a description document for each framework to be configured, as well as eventual accessory scripts for performing deployment tasks.

The *controller* layer implements a parameterized configuration workflow, comprising a sequence of configuration steps specified in the framework descriptor document accordingly to the framework being installed. Each step is responsible for handling the configuration of instances of a given set of components (as specified by the *step* attribute for each component) and performs simple verifications concerning its completion, such as checking the number of configured instances for each component in the step. After the framework-specific steps have been completed, a fixed step is activated to execute deployment procedures and to save the configurations performed by the DL designer. Finally, a last step provides the user with a brief report on the produced configurations, including, among other things, links to the deployed services.

Since the configuration workflow is organized into steps, a wizard-like interface was a natural choice for the *view* layer. In this interface, workflow steps are represented by sequentially-organized panels. Figure 3 shows one of such panels for the configuration of the WS-ODL framework. WS-ODL's workflow includes three specific steps, each one comprising the configuration of a major aspect of a digital library according to this framework: the library itself, its collections or metadata catalogs, and the services it may provide. In each of these steps, the parameters associated to the component instances they list are presented in dynamically-created, tab-organized forms. Each tab corresponds to a parameter group. Form elements are designed according to the definition of the parameters they represent: multi-valued parameters are shown as either lists or trees (depending on whether they should be hierarchically organized or not), parameters representing file

resources present a file chooser dialog, strings and integers are simply shown as text fields. In Figure 3, for instance, the multi-valued, flat parameter *Fields to Index* (associated to the *Indexation* group, as specified in Figure 2) is presented as an editable list of items. Also, the semantics of every parameter is displayed as a tooltip near the parameter label. Constraint verification is performed against every value entered by the user; in case of an erroneous value, a corresponding exception is raised and the user is properly notified.

4. Conclusions and Future Work

In this paper, we presented WhizKEY, a new tool to assist users in the installation of digital libraries from existing software frameworks. From a user's viewpoint, WhizKEY aims at shielding DL designers from the intricacies of the underlying installation task, allowing them to concentrate on the requirements of the system being instantiated rather than on the installation task itself. For such, designers are guided through the task, and every value input by them is verified against a set of constraints in order to guarantee the correctness of the produced configuration. From a framework developer's viewpoint, WhizKEY implements the installation task as a workflow, regarding each step in this workflow as responsible for the configuration of instances of major aspects within a digital library (e.g., the library itself, its collections or catalogs, and the services it may provide). This workflow is fully-configurable to enable the installation of digital libraries based on different software frameworks.

As future work, we plan to further test WhizKEY's model generality based on other software frameworks as well as to enhance its interface based on usability tests we have carried on with some of its potential users, namely computer scientists and librarians [Santos et al. 2006].

References

- Burbeck, S. (1987). Applications programming in Smalltalk-80: How to use Model-View-Controller (MVC). Technical report, Softsmarts Inc.
- Ekenstierna, M. and Ekenstierna, M. (2002). Evaluation of user assistance in graphical user interface software. Master's thesis, Department of Communication Systems at Lund Institute of Technology, Lund, Sweden.
- Gonçalves, M. A., Fox, E. A., Watson, L. T., and Kipp, N. A. (2004). Streams, Structures, Spaces, Scenarios, Societies (5S): A formal model for digital libraries. *ACM Transactions on Information Systems*, 22(2):270–312.
- Pree, W. (1994). Meta Patterns - a means for capturing the essentials of reusable object-oriented design. In *Proceedings of the 8th European Conference on Object-Oriented Programming*, pages 150–162, Bologna, Italy. Springer-Verlag.
- Roberto, P. A., Santos, R. L. T., Gonçalves, M. A., and Laender, A. H. F. (2006). On RDBMS and workflow support for building componentized digital libraries. In *Anais do XXI Simpósio Brasileiro de Banco de Dados*, pages 87–101, Florianópolis, SC, Brazil. SBC.
- Santos, R. L. T., Roberto, P. A., Gonçalves, M. A., and Laender, A. H. F. (2006). Design, implementation, and evaluation of a wizard tool for setting up component-based digital libraries. In *Proceedings of the 10th European Conference on Research and Advanced Technology for Digital Libraries*, pages 135–146, Alicante, Spain. Springer-Verlag.

XVBA: A Tool for Semi-Automatic Generation of SQL/XML Views

Vânia Maria Ponte Vidal, Fernando Cordeiro Lemos

Department of Computing – UFC

Fortaleza – CE – Brazil

{vvidal, fernandocl}@lia.ufc.br

Abstract. *In this work, we present XVBA (XML View By Assertions), a tool that facilitates the task of generating XML view of relational data. In our approach, the XML view is defined by an XML schema and a set of Correspondence Assertions which specifies the mapping between the XML view schema and a base relational schema. Based on the view correspondence assertions, XVBA automatically generates the SQL/XML query that constructs the XML view elements from the relational tuples.*

1. Introduction

XML has emerged as the standard information exchange format for Internet-based business applications. However, since most business data is currently stored in relational database systems, the problem of publishing relational data in XML format has special significance. A general and flexible way to publish relational data in XML format is to create XML views of the underlying relational data.

With the introduction of the XML datatype [1] and the SQL/XML standard [1] as part of SQL:2003 [1], users may resort to the SQL/XML publishing functions to create virtual XML views over base relational schemas. Oracle [5] was the first DBMS to support, with its XML DB module [5], the creation of XML Views as SQL/XML queries over the relational data. The advantages of this approach rely on the use of a standard to publish relational data and on the capacity to process the SQL/XML publish functions within the SQL statements, which represents a gain in performance [5]. Thus, XML Query rewrite can be performed inside the DBMS [5], as opposed to non-integrate mid-tier solution.

However, creating SQL/XML view definitions demands advanced knowledge of SQL/XML, is time consuming and error-prone. This is because writing such SQL/XML queries involve restructuring and grouping of sub-queries, and implementing such operations can easily lead to large queries that are hard to comprehend and often hide the semantics of the transformation [4]. Moreover, users will have to redefine the XML view whenever the base relational schema changes. Therefore, tools that facilitate the task of XML view creation and the maintenance should be developed.

In this work, we propose the *XVBA (XML View By Assertions)*, a tool that facilitates the task of XML view creation and maintenance. A simplified view of *XVBA* architecture is shown in Figure 1. Through the *GUI component*, the user defines the XML view schema and loads the relational schema. Then the user defines graphically the mapping between the XML view schema and the relational schema with the help of view correspondence assertions [7], which are simple to understand.

The second component, the *Mapping Generator Component*, generates the mapping document, which contains a set of correspondence assertions that fully specifies the view schema in terms of the relational schema. In [7] we formally specify the conditions for a set of correspondence assertions to fully specify the view in terms of the relational schema and, if so, we show that the mappings defined by the view correspondence assertions can be expressed as SQL/XML view definition. As shown in [7], our formalism deals with the problem of the semantic heterogeneity.

The third component, the *SQL/XML Query Generator Component*, takes as input the mapping document and generates the SQL/XML view definition.

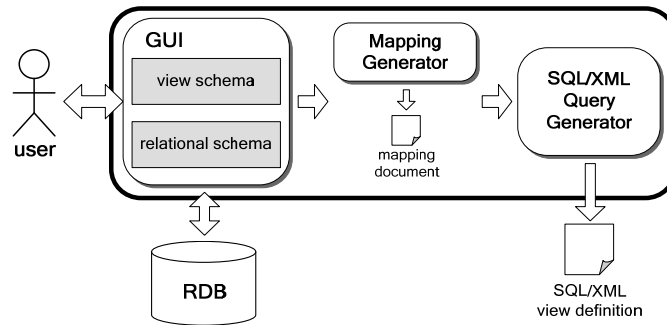


Figure 1. XVBA Architecture

Schema mappings have been used for data exchange [2] and data integration [3][6][8]. The Clio System [3] provides a declarative way of specifying schema mappings and uses these schema mappings to generate the query that captures the transformation semantics of the mappings. Clio allows users to specify a set of mappings between multiple source schemas and a target schema. However, the processes of schema mappings and query generation are much more complex than the process with *XVBA*. Therefore, *XVBA* is more appropriate for the specific task of generating XML view of a single relational data source.

This article is organized as follows. Section 2 presents the process of generating XML Views with *XVBA*. Section 3 discusses how to generate the SQL/XML view definition from the view correspondence assertions. Section 4 presents the conclusions.

2. Generating XML Views with XVBA

We propose to specify an XML view with the help of a set of correspondence assertions, which axiomatically specify how the XML view elements are synthesized from tuples of the base source. Let S be the base relational schema. An *XML view*, or simply, a *view* over S is a quadruple $V = \langle e, T, R, \mathcal{A} \rangle$, where: e is the name of the primary element of the view; T is the XML type of element e ; R is a relation (or view) scheme of S ; \mathcal{A} is a set of path correspondence assertions that fully specifies T in terms of R . We say that the pair $\langle e, T \rangle$ is the *view schema* of V and that R is the *pivot relation (or view) scheme* of the XML view, such that exists a 1-1 mapping between the tuples of the pivot table/view and the elements of the view. As discussed in [7], in case where there is no relation or view that satisfies the 1-1 mapping constraint, we first define a relational view that satisfies that constraint.

The process for generating an XML view with *XVBA* consists of the following steps:

Step 1: the user defines the XML type of the primary elements of the view. *XVBA* features a simple graphical interface to support the definition of XML types.

Step 2: the user selects, from a list of tables and relational views of the database schema, the *pivot table* or *pivot view*, such that exists a 1-1 mapping between the tuples of the pivot table/view and the elements of the view.

Step 3: the user specifies the correspondence assertions that define the relationships between the elements and attributes of the view's XML type and the attributes/path of the pivot table. Figure 3 shows a screenshot of the *XVBA* graphical interface that supports the definition of view correspondence assertions. The left-hand schema is the XML schema of *PurchaseOrder_XML* view whose primary element *<PurchaseOrder>* has type *PurchaseOrder_Type*, and whose pivot relation scheme is *ORDERS_REL* (right-hand schema) of the relational schema *ORDERS_DB* in Figure 2. It is important to notice that, from the pivot relation/view, the user can navigate to any related table through a foreign key or inverse of a foreign key.

The correspondence assertions of *PurchaseOrder_XML* are generated by: (1) matching the elements and attributes of *PurchaseOrder_Type* with attributes or paths of *ORDERS_REL*; and (2) recursively descending into sub-elements of *PurchaseOrder_Type* to define their correspondence assertions. For example, to define the assertion of the element *LineItem* ($\psi_{11}:[PurchaseOrder_Type/LineItem] \equiv [ORDERS_REL/FK_2^{-1}]$), the user selects the element *LineItem* on the view schema and the inverse foreign key FK_2^{-1} on the database schema. Then, the user should specify the correspondence assertions of the elements of *LineItem_Type* with attributes or paths of *LINE_ITEMS_REL* ($\psi_{12}, \psi_{13}, \psi_{17}$ and ψ_{18}). Finally, the user should specify the correspondence assertions of the elements of *Product_Type* with attributes or paths of *PRODUCT_REL* (ψ_{14}, ψ_{15} and ψ_{16}).

Step 4: *XVBA* automatically generates, based on view correspondence assertions, the SQL/XML query that constructs the XML view elements from the relational tuples. Figure 4 presents the SQL/XML definition of *PurchaseOrder_XML* view. In [7] we present an algorithm that generates, based on the view correspondence assertions, the SQL/XML view definition. Moreover, we showed that the SQL/XML query generated by the algorithm correctly represents the mapping defined by the view correspondence assertions.

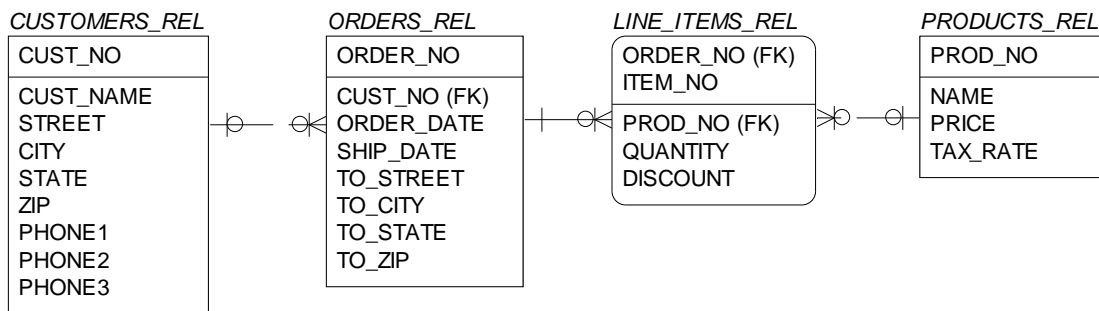


Figure 2. Relational Schema ORDERS_DB

SBBD 2007
IV Sessão de Demos

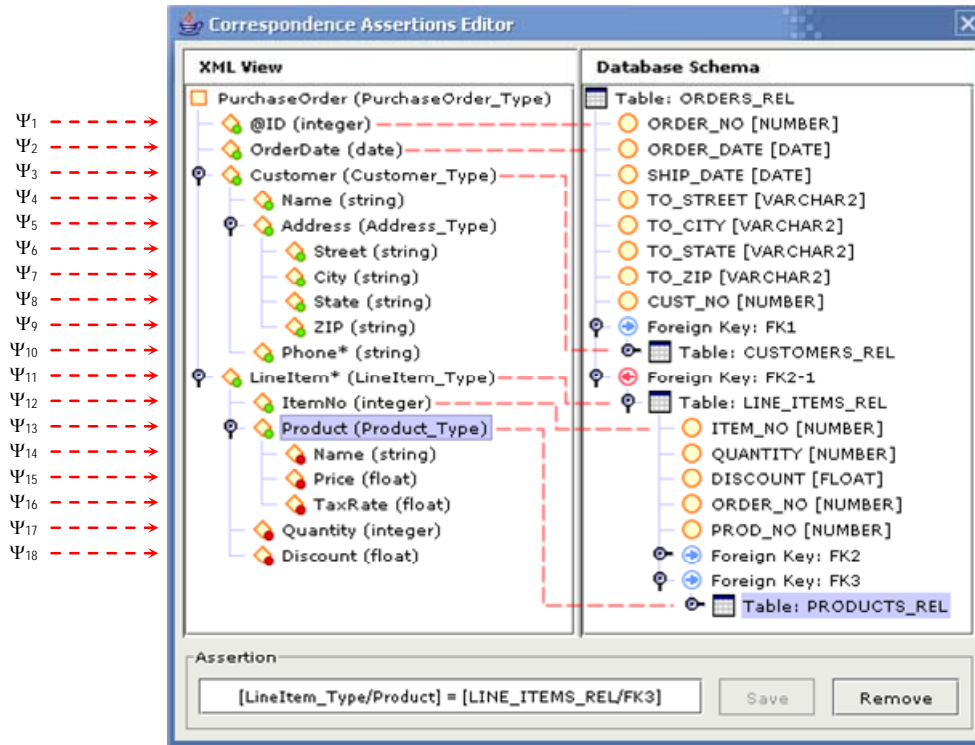


Figure 3. Correspondence Assertions Editor

```

1. CREATE OR REPLACE VIEW PurchaseOrder_XML OF XMLTYPE
2. XMLSCHEMA "PurchaseOrder.xsd" ELEMENT "PurchaseOrder"
3. AS SELECT XMLELEMENT("PurchaseOrder",
4.   XMLATTRIBUTES(O.ORDER_NO AS "ID"), ..... from Psi1:[PurchaseOrder_Type/ID] = [ORDERS_REL/ORDER_ID]
5.   XMLFOREST(O.ORDER_DATE AS "OrderDate"), ..... from Psi2:[PurchaseOrder_Type/OrderDate] = [ORDERS_REL/ORDER_DATE]
6.   (SELECT XMLELEMENT("Customer", ..... from Psi3:[PurchaseOrder_Type/Customer] = [ORDERS_REL/FK1]
7.     XMLFOREST(C.CUST_NAME AS "Name"), ..... from Psi4:[Customer_Type/Name] = [CUSTOMERS_REL/CUST_NAME]
8.     XMLELEMENT("Address", ..... from Psi5:[Customer_Type/Address] = [CUSTOMERS_REL/NULL]
9.       XMLFOREST(C.STREET AS "Street"), ..... from Psi6:[Address_Type/Street] = [CUSTOMERS_REL/STREET]
10.      XMLFOREST(C.CITY AS "City"), ..... from Psi7:[Address_Type/City] = [CUSTOMERS_REL/CITY]
11.      XMLFOREST(C.STATE AS "State"), ..... from Psi8:[Address_Type/State] = [CUSTOMERS_REL/STATE]
12.      XMLFOREST(C.ZIP AS "ZIP"), ..... from Psi9:[Address_Type/ZIP] = [CUSTOMERS_REL/ZIP]
13.      XMLFOREST(C.PHONE1 AS "Phone"), ..... from Psi10:[Customer_Type/Phone] = [CUSTOMERS_REL/{PHONE1,PHONE2,PHONE3}]
14.      XMLFOREST(C.PHONE2 AS "Phone"),
15.      XMLFOREST(C.PHONE3 AS "Phone"))
16.   FROM CUSTOMERS_REL C
17.   WHERE C.CUST_NO = O.CUST_NO),
18.   (SELECT XMLAGG( XMLELEMENT("LineItem", ..... from Psi11:[PurchaseOrder_Type/LineItem] = [ORDERS_REL/FK2-1]
19.     XMLFOREST(L.ITEM_NO AS "ItemNo"), ..... from Psi12:[LineItem_Type/ItemNo] = [LINE_ITEMS_REL/ITEM_NO]
20.     (SELECT XMLELEMENT("Product", ..... from Psi13:[LineItem_Type/Product] = [LINE_ITEMS_REL/FK3]
21.       XMLFOREST(D.NAME AS "Name"), ..... from Psi14:[Product_Type/Name] = [PRODUCTS_REL/NAME]
22.       XMLFOREST(D.PRICE AS "Price"), ..... from Psi15:[Product_Type/Quantity] = [PRODUCTS_REL/PRICE]
23.       XMLFOREST(D.TAX_RATE AS "TaxRate") ) ..... from Psi16:[Product_Type/ID] = [PRODUCTS_REL/TAX_RATE]
24.     FROM PRODUCTS_REL D
25.     WHERE D.PROD_NO = L.PROD_NO),
26.     XMLFOREST(L.QUANTITY AS "Quantity"), ..... from Psi17:[LineItem_Type/Quantity] = [LINE_ITEMS_REL/QUANTITY]
27.     XMLFOREST(L.DISCOUNT AS "Discount") ) ) ..... from Psi18:[LineItem_Type/Discount] = [LINE_ITEMS_REL/DISCOUNT]
28.   FROM LINE_ITEMS_REL L
29.   WHERE L.ORDER_NO = O.ORDER_NO))
30. FROM ORDERS_REL O;

```

Figure 4. SQL/XML Definition of PurchaseOrder_XML View

3. Mapping Assertions to SQL/XML

Let S be a relational schema and $V = \langle e, T, R, \mathcal{A} \rangle$ be a XML view over S . Given an instance σ of S , let $\sigma(R)$ denote the relation that σ associates with R . The correspondence assertions in \mathcal{A} define a functional mapping, denoted DEF_V , from instances of the source schema S to instances of the view schema.

Given an instance σ of S , the value of V on σ is given by:

$$DEF_V(\sigma) = \{ \$t \mid \$t \text{ is an } \langle e \rangle \text{ element of type } T \text{ and} \\ \exists r \in \sigma(R) \text{ such that } \$t \text{ is semantically} \\ \text{equivalent to } r \text{ as specified by } \mathcal{A} \ (\$t \equiv_{\mathcal{A}} r) \}$$

The SQL/XML definition of V is given by:

```
CREATE VIEW V OF XMLTYPE
AS SELECT XMLELEMENT( "e",  $\tau[R \rightarrow T][r]$  )
FROM R r
```

where $\tau[R \rightarrow T][r]$ is a constructor function, such that, given a tuple r of R , $XMLELEMENT("e", \tau[R \rightarrow T][r])$ returns an instance $\$t$ of T such that $\$t \equiv_{\mathcal{A}} r$. In [7] we present an algorithm that generates, based on \mathcal{A} , the constructor function $\tau[R \rightarrow T][r]$.

In Figure 4, the constructor function $\tau[ORDERS_REL \rightarrow PurchaseOrder_Type](O)$ (lines 3 to 29) constructs the content of an instance of *PurchaseOrder_Type* from a tuple of *ORDERS_REL*. The constructor function contains four sub-queries, one for each element or attribute of *PurchaseOrder_Type*. Each subquery is generated from the assertion of the corresponding element or attribute. Figure 4 shows the assertions that generates each SQL/XML subquery of $\tau[ORDERS_REL \rightarrow PurchaseOrder_Type](O)$. In [7] we show that given a tuple O of *ORDERS_REL*, $XMLELEMENT("PurchaseOrder", \tau[ORDERS_REL \rightarrow PurchaseOrder_Type](O))$ returns an instance $\$P$ of *PurchaseOrder_Type* such that $\$P$ is semantically equivalent to O , as specified by the assertions of *PurchaseOrder_XML*.

4. Conclusions

In this work we present *XVBA*, a tool that facilitates the task of generating XML view of relational data. The tool features a simple graphical interface to support the definition of the view correspondence assertions which define a mapping from instances of the relational schema to instances of the XML view schema. *XVBA* automatically generates, based on the view correspondence assertions, the SQL/XML query that constructs the XML view elements from the relational tuples. We showed in [7] that the SQL/XML query generated by the algorithm correctly represents the mapping defined by the view correspondence assertions.

The tool also supports the maintenance of XML views. Based on the view correspondence assertions, *XVBA* automatically identifies the XML views that are affected by the changes to the database schema. For each affected view, the user should define its new set of correspondence assertions, and then, *XVBA* generates the new SQL/XML view definition.

As future work, we envision a number of extensions to our mapping formalism to express broader types of view schema. We are working in generalizing our mapping formalism and algorithm to deal with XML views of object-relational data.

References

- [1] Eisenberg, A., Melton, J., Kulkarni, K., Michels, J.E. and Zemke, F., *SQL:2003 has been published*. In: ACM SIGMOD Record, v. 33, n. 1, pp. 119–126, 2004.
- [2] Fagin, R., Kolaitis, P. G., Miller, R. J., Popa, L., *Data Exchange: Semantics and Query Answering*. In: Proc. of the 9th Intern. Conf. on Database Theory, pp. 207–224, 2003.
- [3] Haas, L. M., Hernandez, M. A., Ho, C.T.H., Popa, L., Roth, M., *Clio Grows Up: From Research Prototype to Industrial Tool*. In: Proc. of the 2005 ACM SIGMOD Intern. Conf. on Management of Data, pp. 805–810, 2001.
- [4] Jiang, H., Ho, H., Popa, L., Han, W., *Mapping-driven XML Transformation*. In: Proc. of the 16th International Conference on World Wide Web, pp. 1063–1072, 2007.
- [5] Liu, Z. H., Krishnaprasad, M., Arora, V., *Native XQuery Processing in Oracle XMLDB*. In: Proc. of the ACM SIGMOD Intern. Conf. on Management of Data, pp. 828–833, 2005.
- [6] Miller, R. J., Haas, L. M., Hernandez, M. A., *Schema Mapping as Query Discovery*. In: Proc. of the 26th Intern. Conf. on Very Large Data Bases, pp. 77–88, 2000.
- [7] Vidal, V. M. P., Casanova, M. A., Lemos, F. C., *Automatic Generation of SQL/XML Views*. In: Proceedings of the 21st Brazilian Symposium on Databases, pp. 221–235, 2006.
- [8] Yu, C., Popa, L., *Constraint-Based XML Query Rewriting For Data Integration*. In: Proc. of the 2004 ACM SIGMOD Intern. Conf. on Management of data, pp. 371–382, 2004.

GeoDWCASE: Uma Ferramenta para Projeto de Data Warehouses Geográficos

Rafael Fonseca¹, Robson Fidalgo¹, Joel da Silva¹, Valéria Times¹

¹Centro de Informática – UFPE – Universidade Federal de Pernambuco caixa postal
7851 – 50732-970 – Recife – PE – Brasil

{rlf,rdnf,js,vct}@cin.ufpe.br

***Abstract.** This paper describes GeoDWCASE, a CASE tool for designing Geographical Data Warehouses. GeoDWCASE is based on the GeoDWM metamodel and helps the database developer in building the application conceptual data model. It offers a rich modeling interface and a validation mechanism to verify the consistency of the designed data model. Moreover, it is possible to automatically generate the logical schema based on the conceptual model. Finally, some issues concerning to the application of the proposed tool are also discussed together with some descriptions for future work.*

1. Introdução

Um DWG (Data Warehouse Geográfico) pode ser definido como uma extensão da abordagem tradicional de DW (Data Warehouse) [Inmon 1997, Kimball 1996], na qual são acrescentados componentes geográficos (descritivos e geométricos) nas tabelas de dimensões e/ou nas tabelas de fatos do esquema dimensional e geográfico do DWG [Fidalgo et al. 2004]. A modelagem e implementação de esquemas para um DWG ainda é um problema em aberto. Assim, a especificação de metamodelos e o desenvolvimento de ferramentas CASE (Computer-Aided Software Engineering) para este fim são contribuições importantes. Neste sentido, fazendo uso do metamodelo GeoDWM (Geographical DW Metamodel) [FONSECA et al. 2007], este trabalho propõe a ferramenta GeoDWCASE (Geographical DW CASE). Ressalta-se que GeoDWM define como dimensões, medidas e tipos geográficos podem ser organizados e relacionados de forma a se obter um modelo de DWG livre de inconsistências sintáticas. Por sua vez, GeoDWCASE oferece um IDE (Integrated Development Environment) que visa auxiliar os projetistas de DWG nas fases de modelagem conceitual e projeto lógico de um esquema dimensional e geográfico. O restante deste artigo está organizado da seguinte forma: a seção 2 apresenta a arquitetura e os componentes de GeoDWCASE, enquanto que a seção 3 ilustra o uso de GeoDWCASE e a seção 4 discute um trabalho relacionado, algumas conclusões e trabalhos futuros.

2. A Ferramenta GeoDWCASE

GeoDWCASE é uma ferramenta que: 1) oferece uma interface amigável e com recursos gráficos que permitem abstrair detalhes de implementação do esquema do DWG, auxiliando os projetistas e usuários no entendimento do seu modelo dimensional e geográfico; 2) é implementada em Java sobre a plataforma aberta Eclipse [Eclipse 2007], garantindo assim sua portabilidade para diversos sistemas operacionais; 3) é baseada no padrão XMI (XML Metadata Interchange) [OMG 2005] para armazenamento, manipulação, recuperação e intercâmbio de metadados, 4) oferece suporte para modelagem de classes UML usando os

estereótipos com pictogramas de GeoDWM; 5) permite que o modelo do DWG em desenvolvimento seja validado através de restrições OCL (Object Constraint Language) [OMG 2006] de GeoDWM; 6) provê a transformação automática entre um modelo conceitual e um esquema lógico compatível com o SGBDE (Sistema Gerenciador de Banco de Dados Espacial) que será utilizado e 7) visa oferecer suporte a engenharia reversa de esquemas lógicos conformes a GeoDWM. A Figura 1 apresenta a sua arquitetura.

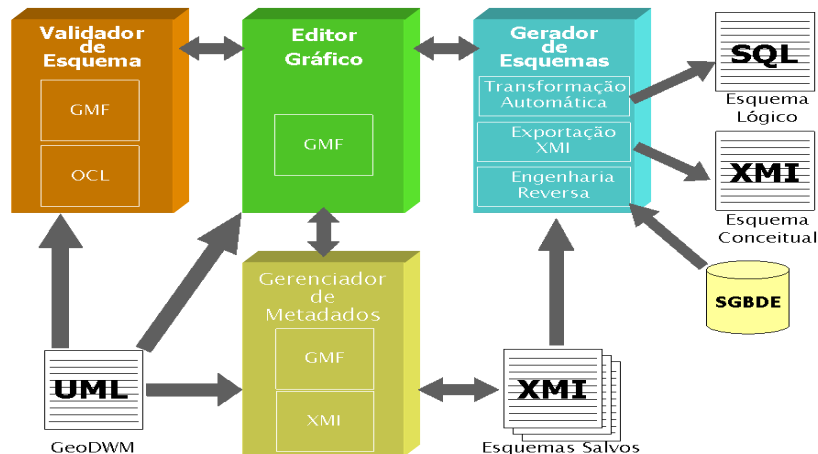


Figura 1. Arquitetura da Ferramenta GeoDWCASE.

Os principais componentes da arquitetura de GeoDWCASE são assim descritos:



Editor Gráfico – Representa a interface de modelagem pela qual o usuário interage com seus esquemas. A plataforma Eclipse compreende uma série de *frameworks* extensíveis para construir, disponibilizar e gerenciar software. Dentre esses *frameworks*, o GMF (Graphical Modeling Framework) [GMF 2007] se destaca por prover uma infraestrutura para o desenvolvimento de editores gráficos guiados por um metamodelo, sendo utilizado no desenvolvimento do editor gráfico de GeoDWCASE.

Gerenciador de Metadados – Usa XMI para gerenciar os metadados de GeoDWM e do esquema do DWG sendo modelado. O metamodelo GeoDWM guia este componente para permitir o armazenamento e a manipulação dos metadados dos esquemas dos DWG salvos. Cada esquema salvo faz parte de um projeto e é definido por metadados gráficos (e.g., posições e desenhos dos elementos de modelagem) e estruturais (e.g., nomes e tipos das dimensões, nomes e tipos das medidas, nomes e tipos das colunas, relacionamentos).

Validador de Esquema – Usa as restrições OCL definidas em GeoDWM para verificar a consistência do esquema do DWG. Por exemplo, a validação de relacionamentos incorretos e a garantia de unicidade de nomes de dimensões, fatos e atributos são verificadas.

Gerador de Esquema – Faz a transformação, importação e exportação de esquemas. Este componente contém submódulos para que os esquemas conceituais sejam automaticamente transformados em esquemas lógicos de um SGBDE. Além disso, estes esquemas lógicos podem ser importados e transformados em esquemas conceituais (engenharia reversa). Esse componente também permite a exportação dos esquemas conceituais em XMI.

3. Exemplo de Aplicação

GeoDWCASE implementa os estereótipos definidos por GeoDWM. Por exemplo, o estereótipo  representa uma dimensão geográfica composta e o estereótipo  uma medida convencional. A descrição completa de todos os estereótipos usados em GeoDWCASE pode ser encontrada em [FONSECA et al. 2007]. Com o uso de GeoDWCASE, o projetista é capaz de realizar manipulações sobre os elementos que representam fatos e dimensões de um DWG, tais como: 1) inserir; 2) excluir; 3) editar (e.g., cores e fontes); 4) visualizar sob diferentes níveis de zoom; 5) organizar (e.g., auto-organizar e alinhar); 6) exportar o modelo como figura (e.g., JPG, GIF, SVG). A Figura 2 apresenta o ambiente de modelagem de GeoDWCASE. Na área 1 da Figura 2, o projetista tem a visão em árvore de todos seus projetos de DWG, juntamente com seus respectivos modelos, organizados em pastas. A área de edição (área 2 da Figura 2) exhibe, como exemplo, o esquema de um DWG sobre dados do DataSUS (Banco de Dados do Sistema Único de Saúde) e do IBGE (Instituto Brasileiro de Geografia e Estatística).

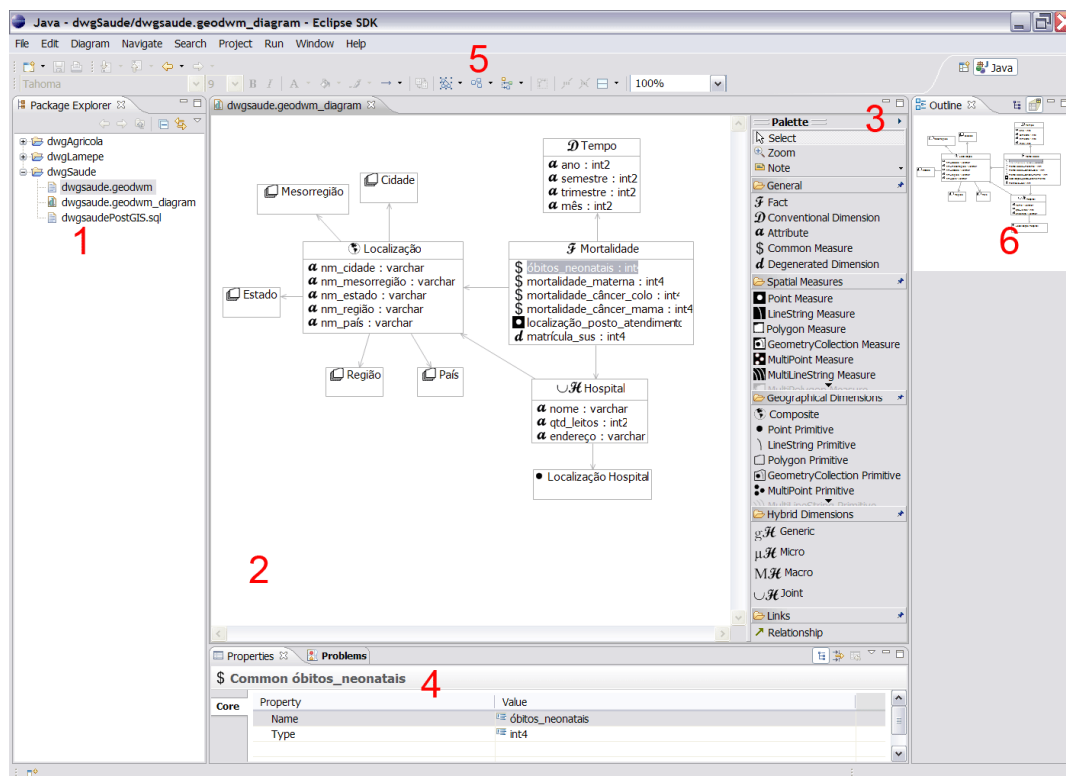


Figura 2. Ambiente de Modelagem de GeoDWCASE.

A paleta de elementos de GeoDWCASE (área 3 da Figura 2) possui construtores para os estereótipos utilizados na modelagem de DWG definidos por GeoDWM. Note que a semântica de cada tipo de elemento é facilmente percebida pelo seu estereótipo com seu pictograma. A tarefa de modelar um DWG consiste em clicar sobre o elemento desejado na paleta e colocá-lo no diagrama. No DWG da Figura 2, tem-se a tabela de fatos *Mortalidade*, a qual possui várias medidas convencionais (e.g., *óbitos_neonatais* que quantifica os óbitos de recém-nascidos), uma medida espacial (*localização_posto_atendimento*) e uma dimensão degenerada (*matrícula_sus*). Relacionadas a essa tabela de fatos tem-se três tabelas de dimensões: *Tempo*, *Localização* e *Hospital*, as quais são utilizadas para representar,

respectivamente, a data em que determinado fato ocorreu, sua localização no mapa e o endereço do hospital. *Localização* é uma dimensão geográfica composta e seus atributos representam as informações descritivas dos objetos geográficos contidos nas dimensões primitivas associadas (e.g., *País*, *Região*, *Estado*, *Mesorregião* e *Cidade*). Por sua vez, *Hospital* é uma dimensão híbrida conjunta, estando relacionada a dimensão composta *Localização* e a dimensão primitiva *Localização Hospital*, a qual indica a geometria da localização de cada hospital. Vale ressaltar que para utilizar GeoDWCASE, a definição de chave primária e estrangeira, em tempo de modelagem, não é necessária. Pois, estas chaves são geradas automaticamente no momento da transformação para o modelo lógico. Isto é, no caso de chave primária, inseri-se, na tabela que está sofrendo a transformação, uma coluna com o nome da própria tabela mais o prefixo “PK”. Por sua vez, no caso de chave estrangeira, inseri-se, na tabela de origem do relacionamento, um campo com o nome da tabela destino mais o prefixo “FK”.

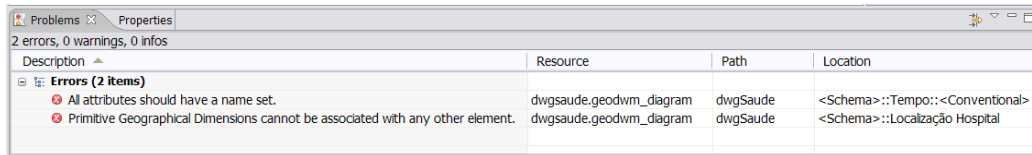
A edição das propriedades de cada um dos elementos esquema da Figura 2 é feita selecionando-se o elemento e alterando o valor da referida propriedade na aba *Properties* (Propriedades) acessada na área 4 da mesma Figura. Essas propriedades podem ser relacionadas à aparência do elemento (e.g., estilo da fonte, tamanho da fonte, cor da fonte, cor da linha, preenchimento) ou aos metadados do elemento (e.g., nome, tipo). A interface possui ainda uma barra com botões para seleção de todos os elementos, organização automática e alinhamento de elementos (área 5 da Figura 2). Além disso, existe uma aba que permite a visão em *outline* do modelo (área 6 da Figura 2), permitindo que se navegue pelo esquema arrastando o *mouse*, facilitando assim a visualização de esquemas grandes.

Como pode ser observado na área 1 da Figura 2, cada esquema sendo projetado é composto por dois arquivos: um arquivo contendo as representações gráficas do modelo (extensão “.geodwm_diagram”) e um arquivo contendo os metadados de GeoDWM usados no modelo (extensão “.geodwm”). O conteúdo desses arquivos é armazenado em XMI, no qual cada elemento é delimitado por uma *tag* (i.e., marcação) que contém seu nome, seu estereótipo, seu tipo e seus relacionamentos. A *tag* “*table*” armazena o estereótipo da tabela, seu nome e as referências dos elementos para os quais a tabela possui chaves estrangeiras (com base nos relacionamentos definidos no modelo). A *tag* interna “*column*” armazena o estereótipo da coluna, seu nome e seu tipo de dado. O arquivo GeoDWM contendo as definições de metadados para este DWG pode ser visto em <http://www.cin.ufpe.br/~golapware/geodwcase/dwgsaude.geodwm>.

O projetista pode, a qualquer instante, checar a consistência de seu modelo (i.e., verificar se o mesmo respeita as regras definidas por GeoDWM). Isso pode ser feito acessando o *menu Diagram* (Diagrama) e selecionando a opção *Validate* (Validar). O componente de validação realiza uma checagem do modelo com base nas regras OCL de GeoDWM e todas as inconsistências de modelagem encontradas são listadas na aba *Problems* (Problemas) acessada na área 4 da Figura 2. Nessa aba, uma descrição dos problemas encontrados é mostrada e os elementos que estão com problemas são marcados com um X no modelo. A Figura 3 ilustra a aba *Problems* com exemplos de alguns erros de modelagem que podem ser encontrados. Nessa aba, a descrição do erro (*Description*), o arquivo (*Resource*), o projeto (*Path*) e a localização no esquema (*Location*) são exibidos. Um clique duplo em qualquer uma das linhas dessa aba leva a seleção do elemento problemático no esquema, como ilustrado na Figura 4.

SBBD 2007

IV Sessão de Demos



Description	Resource	Path	Location
All attributes should have a name set.	dwgsaude.geodwm_diagram	dwgSaude	<Schema>::Tempo::<Conventional>
Primitive Geographical Dimensions cannot be associated with any other element.	dwgsaude.geodwm_diagram	dwgSaude	<Schema>::Localização Hospital

Figura 3. Aba com os Erros de Modelagem.

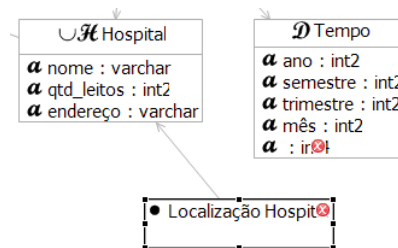


Figura 4. Elementos com Erro.

Ao finalizar a modelagem e validar o modelo, o projetista pode transformar o esquema conceitual do DWG no seu respectivo esquema lógico. GeoDWCASE facilita essa tarefa gerando automaticamente o esquema lógico a ser implementado. Esta implementação é codificada em Java e faz a transformação dos elementos XMI (esquema conceitual) em tabelas e suas respectivas colunas (esquema lógico) segundo a sintaxe da DDL (Data Definition Language) do SGBDE alvo. Pelo fato de cada SGBD espacial ter um conjunto de instruções particular para o tratamento de geometrias, a ferramenta GeoDWCASE precisa implementar um módulo de transformação automática específico para cada SGBD espacial. Um desses módulos já foi implementado e transforma esquemas GeoDWM em DDL do SGBD PostgreSQL [PostgreSQL 2007] com a extensão PostGIS [PostGIS 2007]. Este módulo tem como entrada o arquivo que contém as definições de metadados XMI (extensão “.geodwm”) do esquema a ser transformado. Para realizar a transformação, basta clicar com o botão direito do mouse no arquivo “.geodwm” dentro de um projeto e acessar a função “*Generate PostGIS SQL Code*”.

Após executar a transformação, é gerado um arquivo “.sql” com o modelo lógico. Esse arquivo pode ser acessado no projeto corrente e seu conteúdo pode ser visualizado em GeoDWCASE, não havendo necessidade de se utilizar um editor externo. O arquivo contendo o esquema lógico completo do DWG modelado para a área de Saúde pode ser encontrado em <http://www.cin.ufpe.br/~golapware/geodwcasedwgsaudePostGIS.sql>. Ressalta-se que acentos e caracteres especiais utilizados para nomear os elementos no modelo são desconsiderados na transformação, sendo substituídos por caracteres equivalentes. Um módulo de engenharia reversa está atualmente em desenvolvimento. Esse módulo permitirá que um modelo conceitual seja gerado a partir da leitura de um esquema lógico de um DWG implementado em um SGBD espacial.

4. Conclusões e Trabalhos Futuros

Usar uma ferramenta CASE no processo de desenvolvimento de um DWG faz com que o tempo de criação do projeto seja minimizado, o que representa uma redução de custos e um aumento na qualidade. GeoDWCASE foi desenvolvida para auxiliar o projetista a desenvolver um DWG com mais facilidade e qualidade, tendo como base um metamodelo específico para a especificação de DWG (i.e., GeoDWM). Apesar de ser

voltada para DWG, a modelagem e geração de Data Warehouses convencionais também é permitida, já que GeoDWM define estereótipos para tabelas de fato, medidas e dimensões convencionais. O uso da plataforma Eclipse garante que GeoDWCASE seja utilizada em toda a gama de sistemas operacionais para os quais Eclipse está disponível. Além disso, sua arquitetura modularizada e baseada no *framework* GMF permite que manutenções e extensões sobre GeoDWCASE sejam realizadas com facilidade.

Ferramentas CASE para DWG é um tópico ainda pouco explorado, o exemplo disto é que CASME [Zghal et al. 2003] é atualmente o único trabalho diretamente relacionado a GeoDWCASE. Em comparação a CASME, GeoDWCASE é fundamentada em um metamodelo que é definido a partir de padrões, é implementada sobre uma plataforma aberta que é baseada em Java e XMI e contempla a transformação de esquemas.

Algumas indicações de trabalhos futuros são: 1) o desenvolvimento de novos módulos de geração automática para SGBD espaciais ainda não contemplados, 2) o desenvolvimento de um módulo de engenharia reversa que permita ler esquemas de DWG diretamente do SGBD espacial, 3) a internacionalização de GeoDWCASE, dando suporte a outros idiomas e 4) a definição de uma metodologia para implementação (projeto conceitual, lógico e físico) de DWG segundo GeoDWM com a incorporação de GeoDWCASE nesse processo.

Referências

- Inmon, W. H. (1997). “Building the Data Warehouse”, John Wiley and Sons, 2 edition.
- Eclipse Platform (2007). <http://www.eclipse.org/>.
- GMF - Graphical Modeling Framework (2007). <http://www.eclipse.org/gmf/>.
- Fidalgo, R. N., Times, V. C., Silva, J., e Souza, F. F. (2004). “GeoDWFrame: A Framework for Guiding the Design of Geographical Dimensional Schemas”. In Proceedings of the International Conference on Data Warehousing and Knowledge Discovery, pages 26–37. DaWaK.
- Fonseca, R., Fidalgo, R. N., Silva, J., e Times, V. C. (2007). “Um Metamodelo para a Especificação de Data Warehouses Geográficos”. In Proceedings of the Brazilian Symposium on Databases (SBBB), João Pessoa, Brazil.
- Kimball, R. (1996). “The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses”. John Wiley and Sons, New York.
- OMG - Object Management Group (2005). “XML Metadata Interchange (XMI) Specification 2.1”, <http://www.omg.org/technology/documents/formal/xmi.htm>.
- OMG - Object Management Group (2006). “Object Constraint Language (OCL) Specification 2.0”, <http://www.omg.org/technology/documents/formal/ocl.htm>.
- PostGIS (2007). <http://postgis.refractory.net/>.
- PostgreSQL (2007). <http://www.postgresql.org/>.
- Zghal, H. B., Faiz, S., e Ghézala H. B. (2003). “CASME: A CASE Tool for Spatial Data Marts Design and Generation”, DMDW.



Índice por Autor

A. Brayner	03, 27	M. Biajiz	09
A. H. F. Laender	45, 51	M. Mattoso	21
A. Lopes	03	M. T. P. Santos	09
B. L. Moreira	45	R. da S. Torres	15
C. A. Heuser	33	R. Fidalgo	63
C. A. Yaguinuma	09	R. Fonseca	63
D. C. G. Pedronette	15	R. L. Novais	39
D. Meira	03	R. L. T. Santos	51
E. A. Fox	45	R. Menezes	03
F. C. Lemos	57	S. Lifschitz	39
F. N. A. Krieser	33	V. Araújo	27
G. C. Hazan	39	V. Braganholo	21
G. Figueiredo	21	V. M. Orengo	33
J. da Silva	63	V. M. P. Vidal	57
J. M. Monteiro	27	V. S. Faria	09
L. Eloy	27	V. Times	63
M. A. Gonçalves	45, 51		

SBBD 2007
IV Sessão de Demos