Um *Framework* para Construção de Máquinas de Execução de Consultas Relacionais

Fausto Ayres[1*] **Fabio Porto**[2] **Alvaro Barbosa**[3] **Rubens Melo**[1]

[1]Departamento de Informática
PUC-Rio
Rua Marques de São Vicente, 225, Gávea, 22453-900
Rio de Janeiro– Brasil

[2]Departamento de Engenharia de Sistemas
Instituto Militar de Engenharia – IME
Praça General Tibúrcio 80, Praia Vermelha, 22290-270
Rio de Janeiro – Brasil

[3] Departamento de Informática
Universidade Federal do Espírito Santo
Av. Fernando Ferrari, s/n, Goiabeiras, 29060-900
Vitória – Brasil

E-mail: fausto@inf.puc-rio.br , porto@inf.puc-rio.br , alvaro@inf.ufes.br , rubens@inf.puc-rio.br

Resumo

O acesso integrado à informações publicadas na *web* permite que sejam oferecidos novos serviços combinando *sites*, a princípio projetados autonomamente. Aplicações como essa envolvem a execução distribuída de consultas com acesso integrado aos *sites* w*eb*. Este processo, no entanto, pode apresentar problemas sérios de desempenho frente à imprevisibilidade do tempo de acesso a estes *sites*. Baseado num estudo das características encontradas nos modelos de execução de consultas, desenvolvemos o *framework* denominado *QEEF* para construção de máquinas de consulta para diferentes modelos de execução. Para sua validação, foi implementada a máquina de execução *AQEE,* incorporando o modelo adaptativo adequado à imprevisibilidade do tempo de acesso a *sites,* e foi realizado um estudo de caso baseado neste modelo. O *framework QEEF* foi integrado ao ambiente *CoDIMS* - uma arquitetura flexível para a geração de sistemas configurados para integração de dados.

Palavras–chave: banco de dados, integração de dados, *framework* de software, máquina de execução de consultas e sistemas configurados.

[*] Professor do CEFET-RJ (parcialmente licenciado)

1 Introdução

O acesso integrado a informações publicadas na *web* permite que sejam oferecidos novos serviços combinando *sites*, a princípio projetados para funcionamento independente. Suponha, por exemplo, três *sites web* de supermercados disponibilizando os preços de seus produtos. Um usuário de uma aplicação de integração destes *sites* poderia submeter uma consulta como esta: *"A partir de uma lista inicial de produtos, informar os seus preços em cada um dos três sites de supermercados"*. Aplicações como essa envolvem a execução distribuída da consulta com acesso integrado aos três *sites*.

O suporte para execução de consultas, como a sugerida acima, é provido por sistemas de mediadores, tais como *TSIMMIS*[GPQ+97] e *Le Selec*t[LeSe], que oferecem uma visão integrada das fontes de dados transformando geralmente os dados obtidos, segundo um modelo de dados semi-estruturado, para elementos no modelo de dados de integração. Nestes sistemas, a consulta formulada a partir do modelo de integração deve ser otimizada e executada pelo sistema mediador. O plano de execução de consultas (PEC) assim produzido, incluirá operações de *BindJoin*[FMLS99] que acessam os formulários *web*, modelados aqui como relações virtuais com restrições de acesso permitindo seu tratamento pelo processo tradicional de otimização de consultas baseado em programação dinâmica [Sel79]. Este processo, no entanto, pode apresentar problemas sérios de desempenho frente à situações características do domínio de serviços na *Web*. Em primeiro lugar, as fontes de dados estão sujeitas a grandes variações no tempo de resposta às consultas, principalmente nos horários de pico quando concentram-se os acessos dos navegadores potenciais. Adicionalmente, as constantes modificações feitas nos *sites web* tanto em seu conteúdo quanto em sua forma, devido a sua autonomia, tornam extremamente difícil, ou mesmo impossível, obter informações estatísticas sobre os seus dados. Essas duas características levam à geração de um PEC rígido, inadequado para o domínio.

Tal condição particular levou os pesquisadores a buscarem alternativas com variado grau de dinamismo na geração e execução de PECs sobre *sites web*, tais como: *Query Scrambling* [AFTU96], *Eddies*[AH00], *Tukwila*[IFFLW99], *Hybrid*[BFPV01] e [BFMV00].

Neste contexto, a principal contribuição deste trabalho é projetar o *framework* [FSJ99] denominado *QEEF (Query Execution Engine Framework)* para a construção de máquinas de execução de consulta para diferentes modelos de execução. Em particular, instanciaremos este *framework* para o modelo de execução adaptativo. Uma outra contribuição é a integração do *QEEF* ao ambiente *CoDIMS* [BP01], uma arquitetura flexível para geração de sistemas configurados para

integração de dados, que possibilitará a execução de consultas de integração de dados utilizando diferentes modelos de execução.

O restante deste artigo encontra-se dividido da seguinte forma. A Seção 2 apresenta brevemente o ambiente *CoDIMS* e, em seguida, a Seção 3 especifica o *framework QEEF,* a partir de uma análise de características de execução e sua integração ao *CoDIMS*. A Seção 4 instancia o *framework* para o modelo de execução adaptativo. Finalmente, a Seção 5 apresenta as conclusões finais.

2 O ambiente *CoDIMS*

CoDIMS (Configurable Data Integration Middleware System) é um ambiente flexível para a geração de sistemas *middleware* configurados para integração de dados heterogêneos, em desenvolvimento no Laboratório de Tecnologias em Banco de Dados (TecBD) do Departamento de Informática da PUC-Rio. Sistemas *middleware* de integração são responsáveis por fornecer acesso integrado aos dados que estão armazenados de forma distribuída em fontes de dados diversas. O desenvolvimento de tais sistemas é um atividade complexa, considerando a existência de diferentes perfis e interesses dos usuários, tipos e modelos de dados diversos, que requerem diferentes funcionalidades para os sistemas de integração.

Diferente da abordagem dos sistemas de integração encontrados na literatura - que são demasiadamente específicos, atendendo a uma única aplicação, ou extremamente genéricos para atender às diversas situações de integração - o *CoDIMS* permite a geração de sistemas denominados *light weight* [BT95]. O objetivo é gerar sistemas configurados para atender a requisitos específicos das aplicações, de forma que somente a funcionalidade necessária seja disponibilizada.

A ênfase da abordagem *CoDIMS* é na configuração e na flexibilidade, baseada no uso das técnicas de componentes e de *framework*. A configuração de um novo sistema é obtida através da seleção e a integração de um conjunto adequado de componentes e suas respectivas interfaces, de acordo com a aplicação. Já a flexibilidade é fornecida em três níveis: (i) na escolha dos componentes e interfaces; (ii) na adaptação do comportamento de cada componente, usando a técnica de *framework*; (iii) no uso de uma linguagem de *workflow* [JB96] para expressar diferentes escalonamentos para as operações oferecidas pelo conjunto de componentes.

Em geral, os componentes do *CoDIMS* implementam serviços típicos de gerência de dados, de forma análoga aos existentes nos SGBDs, como Gerência de Metadados, Processamento de Consulta, Gerência de Transações, Controle de Concorrência, Gerência de Regras e Comunicação.

A figura 1 apresenta a arquitetura do *CoDIMS* com alguns componentes. O componente *Control* é o coordenador de execução através do qual os componentes executam suas operações e interagem. Ele age como um *broker* [GHJV95], o que reduz as interdependências entre componentes, facilitando o processo de seleção, substituição e modificação dos mesmos em um sistema configurado. A comunicação entre o *Control* e os outros componentes é realizada através de suas interfaces registradas no *Control*. Este é um outro ponto de flexibilidade: novos componentes podem ser incluídos, um componente pode oferecer uma nova interface, ou ainda, para uma mesma interface implementar diferentes funcionalidades.

No contexto deste trabalho estaremos enfocando apenas o componente *QueryProcessing*. Dele participam as atividade tradicionais de: análise semântica e sintática; otimização e execução de consultas [OV99]. Ele interage com o componente *Communication*, para acesso às fontes de dados, e com o componente *MetadataManager* para obtenção de dados sobre os esquemas exportados. Na próxima seção veremos a integração do *framework* proposto com o *CoDIMS*, através dessas três classes.

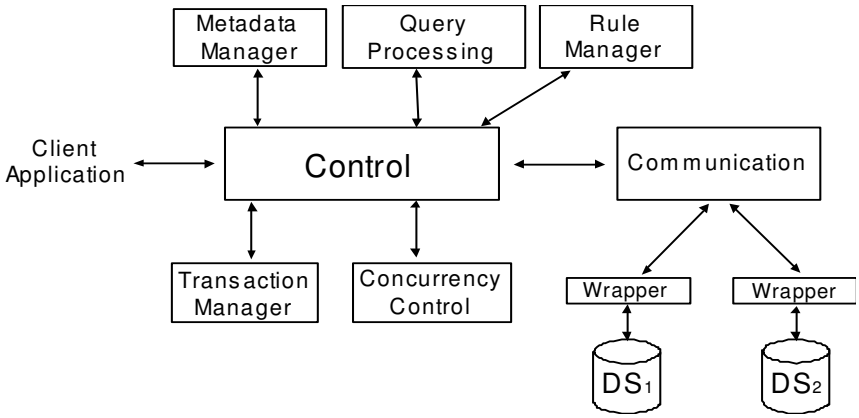

Figura 1— Arquitetura do CoDIMS

3 O *Framework QEEF*

Nesta seção apresentaremos o *QEEF - Query Execution Engine Framework* que foi projetado para construir máquinas de execução de consultas para diferentes modelos de execução. Inicialmente, apresentamos nossa visão desses modelos e, em seguida, especificamos o *QEEF* e sua integração ao ambiente *CoDIMS*.

3.1 Modelo de execução de consultas

Definimos um modelo de execução de consulta como uma combinação das características de execução[Grae93] associadas ao processamento dos operadores de um PEC. A seguir descreveremos essas características:

- Sincronismo – define o estado da execução de um operador quando ele requer serviços ou dados de um outro operador. Existem dois possíveis estados: *wait* (Síncrono), *no wait* (Assíncrono);
- Paralelismo – considera dois tipos de paralelismo: intra e inter operador;
- Distribuição – suporta o consumo de dados a partir de um operador remoto;
- Fluxo de Dados – especifica a ordem através da qual os dados são processados. Existem dois tipos: *Fixed*, onde os dados percorrem o mesmo caminho através dos operadores, e *Adaptive*, onde os dados percorrem caminhos diferentes através dos operadores dependendo da performance desses operadores.
- Fluxo de Controle – refere-se ao tipo de comunicação entre operadores formando um encadeamento de execução entre consumidores e produtores. As alternativas representadas na Figura 2 são, respectivamente, (a) *Demand-Driven* (b) *Data-Driven* (c) *Active-Scheduler* (d) *Passive-Scheduler*, e podem ser combinadas de acordo com o modelo de execução utilizado.

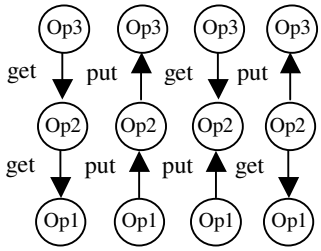

Figura 2 – Fluxos de controle

- Tempo de Resposta – refere-se ao momento no qual os resultados da consulta serão disponibilizados à aplicação do usuário. Existem dois tipos: *First Tuple,* que requer que os dados sejam enviados para o usuário imediatamente à medida que eles sejam processados, e *Last Tuple*, que requer que os dados sejam enviados para o usuário somente quando todos eles sejam processados.
- Materialização – especifica o nível de materialização de dados entre cada par de operadores. Pode variar entre os tipos *total* , *partial* ou *null*.

Considerando-se que o modelo de execução é definido a partir da seleção das características dentre as acima descritas, o *QEEF* tem como objetivo a instanciação de uma máquina de execução de consulta que suporte esse modelo.

3.2 Especificação do *QEEF*

A Figura 3 apresenta o diagrama de classes do *framework QEEF*. A classe *QueryEngine* é um padrão *facade*[GHJV95] responsável pela interface da máquina de execução com a aplicação usuária que submete um Plano de Execução de Consulta (PEC) na forma de uma árvore de consulta profunda à esquerda onde os nós folhas correspondem às fontes de dados e os nós internos correspondem aos operadores algébricos.

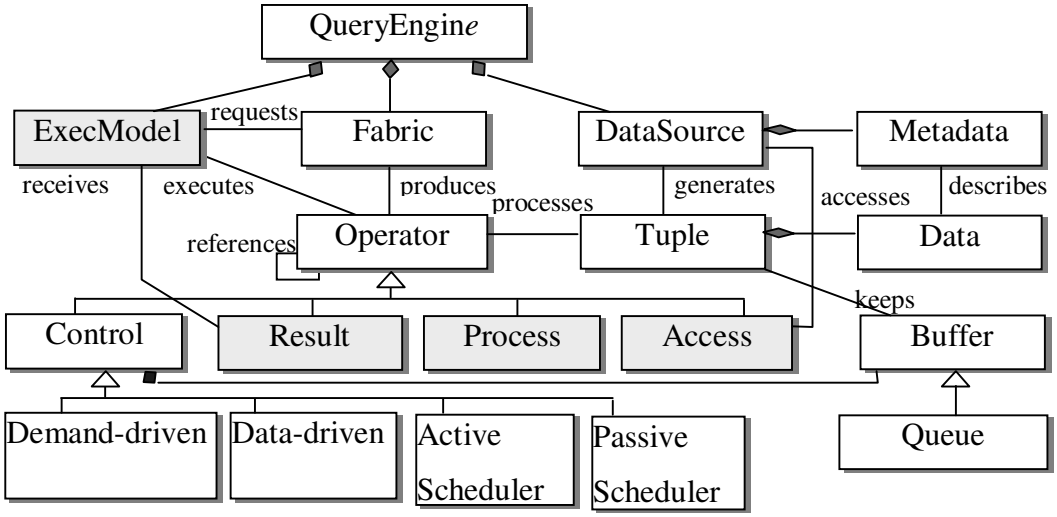

Figura 3 – Diagrama de classes do *framework QEEF*

A classe *Fabric* cria os operadores algébricos e de controle utilizados no PEC. A classe *Operator* implementa, no estilo *pipeline,* uma interface comum a todos os operadores exportando os métodos *open*, *get*, *put* e *close*. Os operadores processam instâncias da classe *Tuple* que representa uma unidade de processamento. As subclasses *Access*, *Process* e *Result* são pontos de flexibilização (*hot spot*) que implementam um dado conjunto de operadores algébricos de acesso, processo e resultado, respectivamente. A classe *Access* faz uso das classes *DataSource* e *Metadata* que descrevem o acesso e os metadados de cada fonte de dados utilizada na consulta. A classe *Result* formata e envia os resultados para a classe *ExecModel.* A classe Process representa os operadores dos nós internos do PEC.

A classe *Control* representa os operadores de controles que são responsáveis pelo desacoplamento entre os operadores algébricos.

A classe *ExecModel* é o principal ponto de flexibilização (*hot spot*) do *QEEF* e precisa ser especializada, via padrão *strategy*[GHJV95], para implementar diferentes modelos de execução. A classe especializada deverá selecionar corretamente os operadores para construir um PEC de acordo com as características presentes no modelo de execução. Essas características de execução são implementadas, a partir da instanciação do *framework,* da seguinte forma: (1) os quatro tipos de fluxo de controle são implementados pelas subclasses: *Demand-driven, Data-driven, Active-*

scheduler e *Passive-scheduler;* (2) os tipos de fluxos de dados são obtidos através da escolha correta da ordem de conexão desses operadores de controle; (3) assincronismo, distribuição e paralelismo entre os operadores são obtidos instanciando-se cada operador de controle com sua própria *thread* de execução; e (4) os tipos de materialização são implementados através da classe *Buffer* e instanciados pela classe *Control* de acordo com a política de materialização – por exemplo: para o tempo de resposta *last tuple,* utiliza-se um *Buffer* com materialização total e para o tempo de resposta *first tuple* e fluxo de dados adaptativo, utiliza-se a subclasse *Queue* para garantir a ordem de execução dos dados em *pipeline.*

3.3 Integração do *QEEF* ao *CoDIMS*

Na instanciação do *framework QEEF* o componente *QueryEngine* será integrado ao componente *QueryProcessing* do ambiente *CoDIMS*. Esta integração ocorre durante a etapa de configuração de um novo sistema configurado onde cada componente participante do sistema configurado publica sua interface contendo as operações oferecidas (*offered-operations*) e as operações requeridas (*required-operations*).

A Figura 4 mostra a configuração do componente *QueryEngine* onde as operações oferecidas são usadas para submissão de consulta (PEC) e para acesso aos seus resultados, e as operações requeridas sugerem ao ambiente *CoDIMS* a instanciação dos componentes *MetadataManager* e *Communication*, como parte do sistema configurado alvo. As classes *Metadata* e *DataSource* do *QEEF* são *proxies* [GHJV95] para acesso às operações requeridas.

```
Define-Component QueryEngine
  Offered-Operations
        executequery (parameters)
        first (): string
        next (): string
        eof (): boolean
  Required-Operations
        MetadataManager, get-object-MD (parameters)
        Communication, exec-sub-query (parameters)
        Communication, get-next-data (parameters)
End-Component
```

Figura 4 – Configuração do componente *QueryEngine* no *CoDIMS*

4 Estudo de Caso

Nesta seção instanciamos o *framework QEEF* para produzir uma máquina de execução de consulta baseada no modelo de execução adaptativo. Inicialmente, apresentamos o modelo de execução adaptativo motivado pelo cenário descrito na introdução deste trabalho. A seguir, detalharemos as funcionalidades da máquina instanciada e sua iteração com o CoDIMS.

Consideramos uma fonte de dados local (L1) contendo a lista inicial de produtos e três *sites* de supermercados (S1, S2 e S3) com informações publicadas na *web*. O esquema relacional exportado pelos seus respectivos *wrappers* são: L1(cod, nome), S1(cod, preco), S2(cod, preco) e S3(cod, preco). Uma aplicação de integração que objetiva recuperar os preços de produtos dos três *sites* seria expressa pela consulta SQL abaixo:

Select L1.nome, S1.preco, S2.preco, S3.preco
From L1, S1, S2, S3
Where L1.cod = S1.cod and L1.cod = S2.cod and L1.cod = S3.cod

Exemplo 1 – Consulta de integração

O predicado p=($L1.cod$=$Si.cod$) exemplifica a necessidade do fornecimento de um critério de busca para acesso aos dados publicados pelo *site Si*. Neste caso, cada *site* requer a associação de um código de produto para que se tenha acesso ao preço deste produto. Sendo assim, a fonte L1 deve ser a primeira a ser acessada para obtenção da lista inicial de produtos. Define-se então, uma ordem parcial entre os operadores que comporão o plano de execução da consulta do Exemplo 1. Tem-se, para árvores profundas à esquerda, as seguintes possíveis ordens de avaliação dos predicados: $O_1 = L1 \rightarrow S1 \rightarrow S2 \rightarrow S3$, $O_2 = L1 \rightarrow S1 \rightarrow S3 \rightarrow S2$, $O_3 = L1 \rightarrow S2 \rightarrow S1 \rightarrow S3$, $O_4 = L1 \rightarrow S2 \rightarrow S3 \rightarrow S1$, $O_5 = L1 \rightarrow S3 \rightarrow S1 \rightarrow S2$ e $O_6 = L1 \rightarrow S3 \rightarrow S2 \rightarrow S1$. Outros planos possíveis incluiriam versões para planos *bushy*.

A determinação do melhor plano a ser executado neste cenário é uma tarefa difícil considerando-se as grandes variações no tempo de resposta aos *sites* envolvidos. Estratégias puramente estáticas não são capazes de modelar tais variações. Já estratégias adaptativas podem combinar os diversos planos possíveis durante a execução de uma consulta recorrendo aos operadores que estejam mais disponíveis a cada instante. No Exemplo 1, uma máquina de execução adaptativa executaria um plano equivalente à $O_1 \cup O_2 \cup O_3 \cup O_4 \cup O_5 \cup O_6$.

De acordo com o cenário apresentado no exemplo 1, nós instanciamos o *framework QEEF* produzindo a máquina de execução adaptativa *AQEE* (*Adaptive Query Execution Engine*) que incorpora as seguintes características de execução: sincronismo = (*no wait*); paralelismo = (*inter-operator*); distribuição = (*null*); fluxo de dados = (*adaptive*); fluxo de controle = (*demand-driven, active and passive scheduler*); tempo de resposta = (*first tuple*) e materialização = (*partial*).

A instanciação dos *hot spots* do *framework* é feita da seguinte forma: a classe *ExecModel* é especializada na classe *AdaptiveModel e* as classes *Access*, *Process* e *Result* são especializadas nas classes *Scan*, *BindJoin* e *Project* respectivamente, correspondendo aos seus operadores algébricos homônimos. As classes de controle *Demand*, *Active* e *Passive* são instanciadas nas classes *Reader*, *Collector* e *Distributer*, respectivamente, as quais incorporam a característica de assincronismo.

A Figura 5 apresenta os operadores do PEC para o exemplo 1. Esses operadores são instanciados pela classe *AdaptativeModel* para proverem as seguintes funcionalidades: o operador *Scan* acessa a fonte local L1; os operadores *Binjoins* BJ1, BJ2 e BJ3 submetem, em paralelo, *bindings* aos *wrappers* S1 , S2 e S3, respectivamente; o operador Project produz as tuplas resultantes da consulta; os operadores *Collector* coletam dados de seus produtores (via *get)* e os envia a seus consumidor (via *put)*; os operadores *Bindjoins* e *Project* solicitam dados do operador *Distributer* através de um operador *Reader* e, por outro lado, o operador *Distributer* envia aos operadores *Reader* somente as tuplas que eles ainda não processaram. As tuplas processadas pelos Bindjoins retornam ao *Distributer* e são armazenadas na *Queue* segundo a ordem do *pipeline* para garantir que as tuplas estejam disponíveis ao usuário assim que sejam processadas por todos os operadores do PEC.

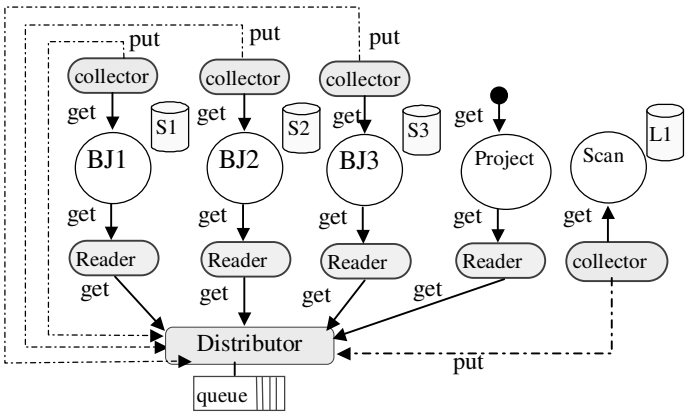

Figura 5 – Execução do exemplo 1

Com relação à integração da máquina *AQEE* com o *CoDIMS,* o componente *QueryProcessing:*

(1) divide a consulta do exemplo 1 em várias sub-consultas que são enviadas aos respectivos *wrappers* (através da operação *Communication.prepare-sub-query(parameters)),* onde a sub-consulta para L1 é expressa como "*select cod, nome from L1*" e as demais sub-consultas são expressas como "*select cod, preco from Si where cod=#c*", sendo *#c* o valor do *binding* e *Si*, a fonte de dados associada ao s*ite i.*;

(2) cria uma instancia da máquina AQEE (componente *QueryEngine*) que por sua vez (a) cria instâncias da classe *DataSource* que encapsula o acesso aos *wrappers* (através das operações *Communication.exec-sub-query(parameters)* e *Communication.get-next-data (parameters)*) e (b) cria instâncias da classe *Metadata* que encapsula o acesso aos metadados (através da operação *MetadataManager.get-object-MD (parameters)*) ;

(3) chama a operação *QueryEngine.executequery(parameters)* para iniciar a execução da consulta

(4) chama as demais operações da máquina (*first, next e eof*) para obter os resultados da consulta.

As classes do *framework QEEF* e de sua instância *AQEE* foram implementadas na plataforma *Java*. A figura 6 mostra a interface do usuário para o exemplo 1 - implementada através de *Servlets*,

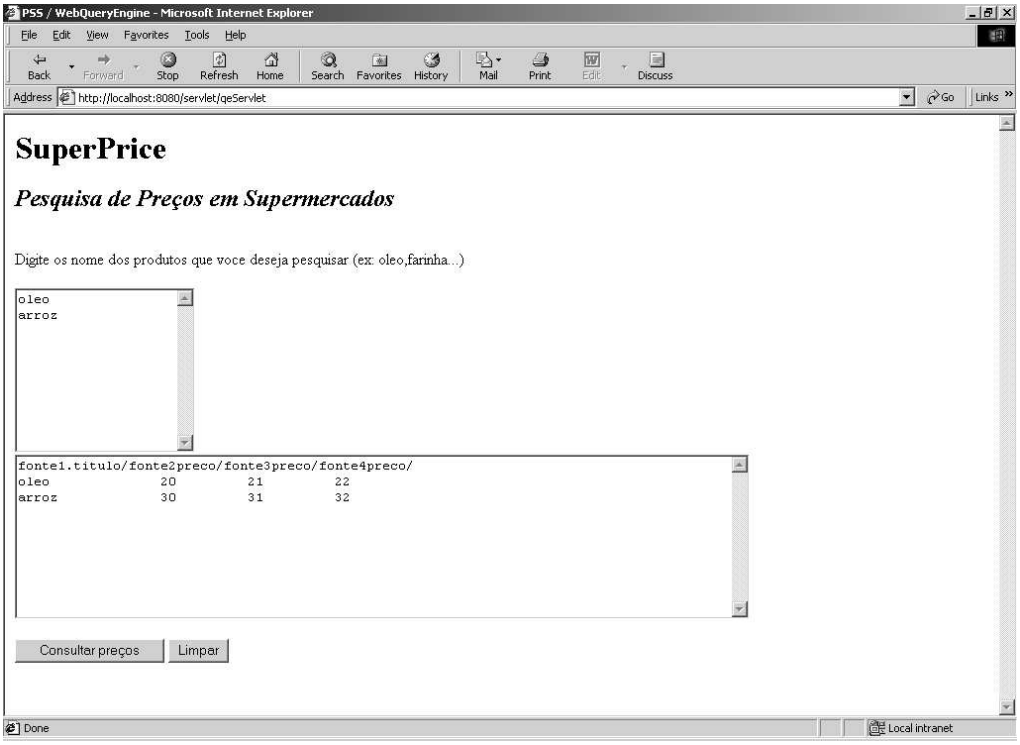

Figura 6 – Interface do usuário

5 Conclusões

Neste trabalho, apresentamos o *framework QEEF* para a construção de máquinas de execução de consultas para diferentes modelos de execução. Em particular, instanciamos o framework para o modelo adaptativo produzindo a máquina de execução adaptativa *AQEE*. Foi realizado um estudo aprofundado sobre as características de execução encontradas em vários trabalhos relacionados tais

como: *Query Scrambling*[AFTU96], *Eddies*[AH00], *Tukwila*[IFFLW99], *Hybrid*[BPFV01] e [BFMV00]. Deste estudo extraímos as principais funcionalidades do *framework.*

Para a validação do *framework* utilizamos um estudo de caso baseado numa consulta de integração de dados a *sites web* onde a adaptatividade está associada a imprevisibilidade do tempo de acesso a estes *sites*. Aplicações como essa apresentam desafios na medida em que o tempo de acesso às fontes de dados registram grandes variações e as informações estatísticas são escassas.

A integração do *AQEE* ao CoDIMS permitiu maior flexibilização do seu componente de Processamento de Consultas.

Por fim, este artigo apresentou as seguintes contribuições:

- um estudo das características de execução de consultas
- o *framework QEEF*
- a máquina *AQEE*
- a integração entre *AQEE* e *CoDIMS* para permitir consultas adaptativas a *sites web*

6 Bibliografia

[AFTU96] L. Amsaleg, M. J. Franklin, A. Tomasic e T. Urhan, "Scrambing Query Plans to Cope with Unexpected Delays", em PDIS Conf., Miami, USA, 1996.

[AH00] R. Avnur e J. Hellerstein, "Eddies: Continuously Adaptive Query Processing", em Proc. Intl. Conf. on Management of Data, Dallas, Texas, 2000.

[BFMV00] L. Bouganim, F. Fabret, C. Mohan e P. Valduriez, "Dynamic Query Scheduling in Data Integration Systems", em Proc. 16o. International Conference on Data Engineering, Sao Diego, CA, Março , 2000.

[BFPV01] L. Bouganim, F. Fabret, F. Porto e P. Valduriez, "Processing Queries with Expensive Functions and Large Objects in Distributed Mediator Systems", em Proc. 17o. Intl. Conf. on Data Engineering (ICDE), Heidelberg, Alemanha, Abril 2001.

[BP01] Barbosa, A.C.P. & Porto, F.A.M. "Configurable Data Integration Middleware System". Proceedings of International Workshop on Information Integration on the *Web* - Technologies and Applications (WIIW2001). Brazil, Abril 2001.

[BT95] Batory, D.S. & Thomas, J. "P2: A Light Weight DBMS Generator". Technical Report TR-95-26, Department of Computer Sciences, University of Texas at Austin, Agosto 1995. http://www.cs.utexas.edu - 10/04/1998.

[FMLS99] D. Florescu, I. Monolescu, A. Levy e D. Suciu, “Query optimization in the Presence of Limited Access Patterns”, em Proc. Intl. Conf. on Management of Data, Filadelfia, USA, Junho, 1999.

[FSJ99] Fayad, M.E.; Schmidt, D.C. & Johnson, R.E. “Building Application *Framework*s – Object-Oriented Foundations of *Framework*s”. John Wiley & Sons, Inc. 1999.

[GHJV95] E. Gamma, R. Helm, R. Johnson e J. Vlissides, “Design Patterns”, em Addison-Wesley, 1995.

[GPQ+97] H. Garcia-Molia, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagin, J. Ullman and J. Widom, “The TSIMMIS Project: Integration of Heterogeneous Information Sources”, em Journal of Intelligent Information Systems, 8(2), pp 177-132, Março, 1997.

[Grae93] G. Graef, “Query Evaluation Techniques for Large Databases”, em ACM Computing Surveys, 25(2), Junho, 1993.

[IFFLW99] Z. G. Ives, D. Florescu, M. Friedman, A. Levy e D. Weld, “An Adaptive Query Execution System for Data Integration”, em Proc. Intl. Conf. on Management of Data, Filadelfia, USA, Junho, 1999.

[JB96] Jablonski, S. & Bussler, C. "Workflow Management – Modeling Concepts, Architecture and Implementation". International Thomson Computer Press, 1996.

[LeSe] “A Mediator System Developed in the Caravel Project”, INRIA, França, em http://caravel.inria.fr/Fprototype_LeSelect.html.

[OV99] M. T. Ozsu e P. Valduriez, “Principles of Distributed Database Systems”, em Prentice Hall, Nova Jersey, USA, 1999.

[Sel79] Selinger P., et al, “Access path selection in a relational data base management system”, "Proc. Intl. Conf. on Management of Data, Maio, Boston, Massachusetts, USA, 1979