6
Estudo de Casos

Neste capítulo, apresenta-se como estudo de casos, a instanciação do QEEF para diferentes modelos de execução e de dados produzindo as MECs listadas na Tabela 5.

Nome	Modelo de Dados	Modelo de Execução
RQEE	Relacional	Tradicional
AQEE	Relacional	Adaptável
XQEE	XML	Baseado em streams

Tabela 5 - MECs produzidas pelo QEEF

6.1.
RQEE

Nesta seção instancia-se o framework QEEF para o modelo de dados relacional obtendo-se a MEC denominada RQEE (*Relational Query Execution Engine*). Esta MEC é utilizada como um componente do SGBD TINO (descrito no Capítulo 1), um SGBD relacional baseado em componentes.

Para demonstrar o funcionamento do RQEE utilizou-se um exemplo baseado em dados demográficos sobre cidades brasileiras onde se deseja saber qual a população das cidades do estado do Rio de Janeiro. O Quadro 8 apresenta a consulta SQL para este exemplo. A Figura 44 indica o modelo de execução tradicional e o PEC a ser executado pelo RQEE, e o Quadro 9 mostra o meta-PEC especificado em um documento XML.

```
Select   Estado.sigla, Cidade.nome, Cidade.populacao
From     Estado, Cidade
Where    Estado.id = Cidade.idestado and Estado.sigla = 'RJ'
```

Quadro 8 - Uma consulta SQL sobre dados demográficos

O PEC especificado é composto pelos operadores Scan (seleção), NestedLoop (junção) e Project (projeção). Para os operadores Scan e NestedLoop são fornecidos, como parâmetros, os predicados a serem avaliados em cada tupla

que processam. Os *alias* para as fontes de dados fornecidos como parâmetros nos dois operadores Scan são utilizadas para conectá-los às referidas fontes durante a fase de criação do PEC na memória.

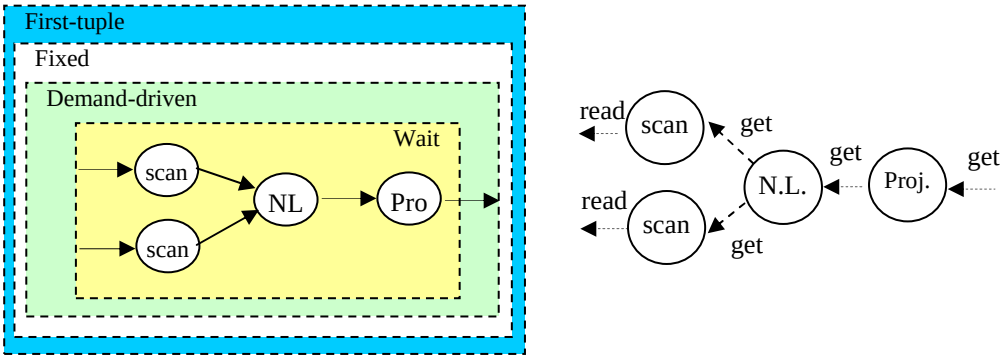


Figura 44 - Modelo de execução do PEC

```
<?xml version="1.0"?>
<METAPLANO>
  <listadeoperadores>
       <operador id="1" tipo="algebrico" nome="Project">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="relacional"/>
              </parametro>
              <parametro tipo="projecao">
              <itemparametro tipo="atributo" valor="estado.sigla"/>
              </parametro>
              <parametro tipo="projecao">
              <itemparametro tipo="atributo" valor="cidade.nome"/>
              </parametro>
              <parametro tipo="projecao">
              <itemparametro tipo="atributo" valor="cidade.populacao"/>
              </parametro>
       </operador>
       <operador id="2" tipo="algebrico" nome="NestedLoop">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="relacional"/>
              </parametro>
              <parametro tipo="predicado">
              <itemparametro tipo="esquerda" valor="estado.idestado"/>
              <itemparametro tipo="operacao" valor="="/>
              <itemparametro tipo="direita" valor="cidade.idestado"/>
              </parametro>
       </operador>
       <operador id="3" tipo="algebrico" nome="Scan">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="relacional"/>
              </parametro>
              <parametro tipo="fonte">
              <itemparametro tipo="nome" valor="estado"/>
              </parametro>
              <parametro tipo="predicado">
              <itemparametro tipo="esquerda" valor="estado.sigla"/>
              <itemparametro tipo="operacao" valor="="/>
              <itemparametro tipo="direita" valor="RJ"/>
              </parametro>
       </operador>
       <operador id="4" tipo="algebrico" nome="Scan">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="relacional"/>
              </parametro>
              <parametro tipo="fonte">
              <itemparametro tipo="nome" valor="cidade"/>
```

```
            </parametro>
      </operador>
  </listadeoperadores>
  <MODULO>
      <FIXED>
      <MODULO>
      <DEMAND-DRIVEN>
      <MODULO>
      <FIRSTTUPLE>
      <MODULO>
      <WAIT>
            <ALGEBRICO nome="Project" ref="1">
                  <ALGEBRICO nome="NestedLoop" ref="2">
                        <ALGEBRICO nome="Scan" ref="3"/>
                        <ALGEBRICO nome="Scan" ref="4"/>
                  </ALGEBRICO>
            </ALGEBRICO>
      </WAIT>
      </MODULO>
      </FIRSTTUPLE>
      </MODULO>
      </DEMAND-DRIVEN>
      </MODULO>
      </FIXED>
      </MODULO>
</METAPLANO>
```

Quadro 9 - Especificação do meta-PEC da consulta 1

O ambiente de execução é ilustrado na Figura 45 e é composto por:

a) arquivos de entrada:

- Plano – contém a especificação do plano;
- Metabase – contém a descrição das fontes de dados;
- Cidade e Estado – contém os dados da consulta;

b) arquivos de saída:

- Log – contém os registros da execução do PEC;
- Saida – contém os resultados da execução do PEC.

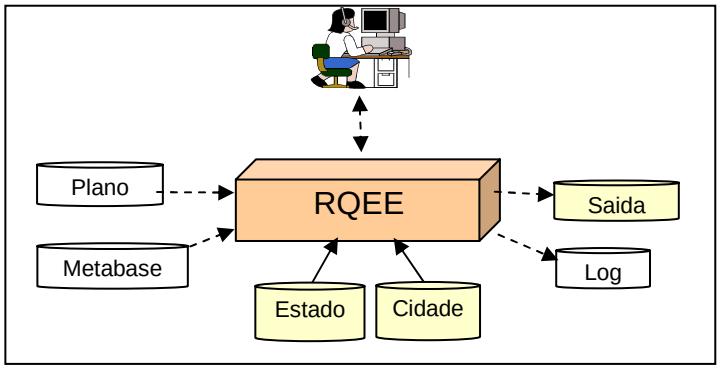


Figura 45 - Ambiente de execução do RQEE

Para uma maior simplicidade na execução deste exemplo, os dados e os resultados são armazenados em arquivos de formato ASCII como mostrado na Figura 46. A Figura 47 descreve a Metabase criada pelo sub-sistema de armazenamento do TINO para a descrição das fontes. Essa descrição é composta por: total de fontes, nome externo da fonte, nome interno da fonte (*alias*), total de atributos, nome do atributo, tipo do atributo (1-caracter, 2-inteiro, etc), tamanho do atributo, deslocamento do atributo (*offset*) e unicidade do atributo (chave primária).

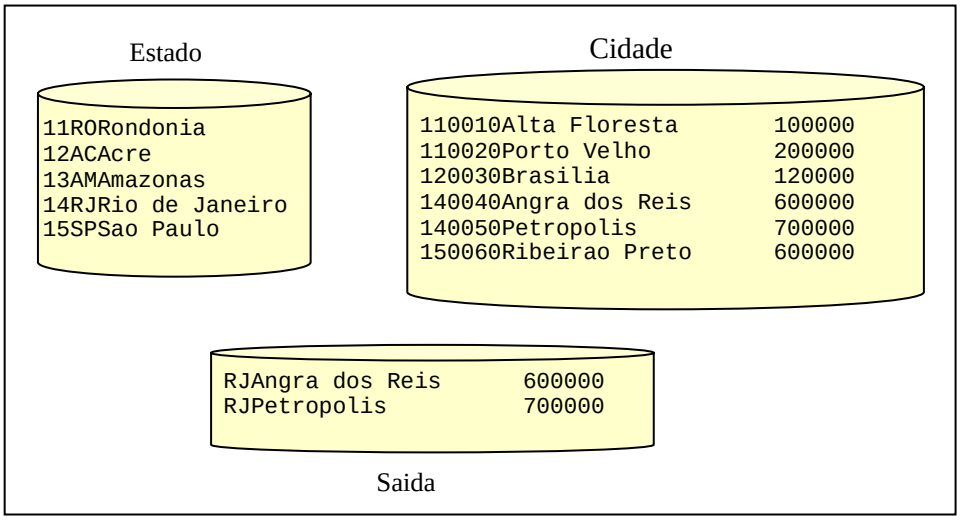


Figura 46 - Dados e resultados da consulta SQL

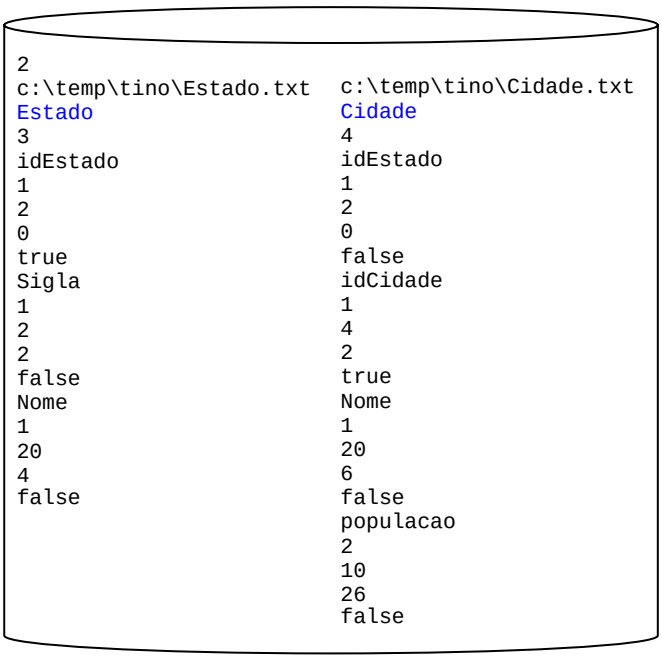


Figura 47 - Conteúdo da Metabase

O processo de instanciação do *framework* QEEF para produzir o RQEE é feito por herança como mostra a Figura 48. As classes instanciadas são:

- RQEE – que recebe as requisições de consultas do SGBD Tino;
- FabricaRelacional – responsável pela criação dos operadores da álgebra relacional;
- Scan, NestedLoop e Project – representam os operadores algébricos utilizados neste estudo de caso;
- TuplaRel – representa uma unidade de dados processada pelos operadores algébricos e contém um conjunto de atributos;
- Atributo – armazena cada valor da tupla e os respectivos metadados que são: nome, tipo e unicidade;
- FonteAscii – implementa uma interface para acesso, através de um Scan, a dados de arquivos ASCII.

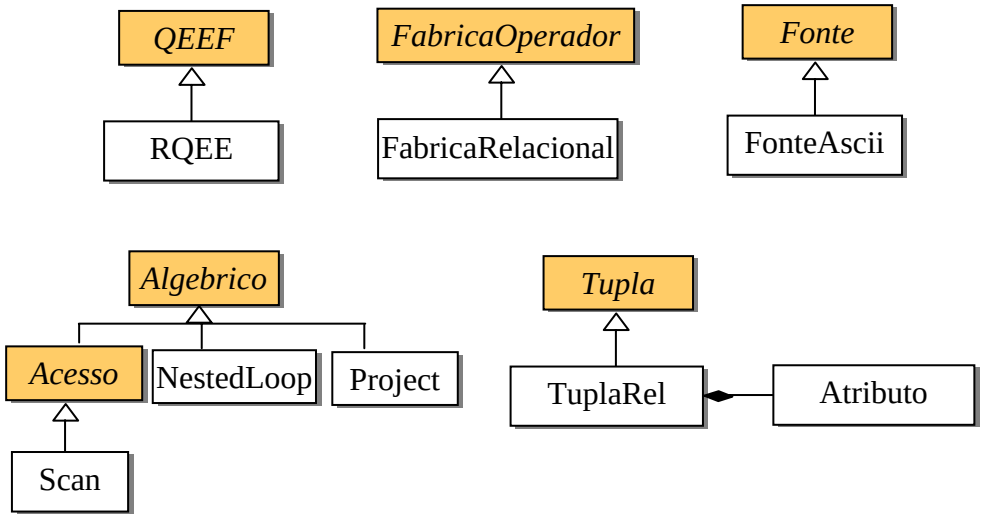


Figura 48 - Instanciação do QEEF para o RQEE

A implementação da classe RQEE é mostrada no Quadro 10.

```
public class RQEE extends QEEF {
 public RQEE ()  {
     super ("RQEE");
     this.inserirFabricaAlgebrico(new FabricaRelacional() );
     this.inserirFabricaFonte (new FabricaFonteRelacional());
 }
 public static void main (String[] args) {
       Log.open("Log.txt");
       RQEE maquina = new RQEE();
      /* inicializações...*/
       maquina.open("Metabase.txt");
       maquina.execute("Plano.xml");
       while (maquina.hasnext())  {
```

```
            Tupla t = maquina.getnext();
            out.write(t.toString());
      }
      maquina.close();
      Log.close();
      out.close();
 }
}
```

Quadro 10 - A classe RQEE

6.2. AQEE

Nesta seção é apresentada a máquina AQEE (*Adaptive Query Execution Engine*) que executa PECs baseados no modelo de execução adaptável discutido no Capítulo 5. Para demonstrar o funcionamento do AQEE utiliza-se uma aplicação de integração de dados descrita a seguir.

Considere 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 sites é composto das relações: L1(cod, nome), S1(cod, preco), S2(cod, preco) e S3(cod, preco). Utiliza-se a consulta SQL abaixo para recuperar, de forma integrada, os preços de produtos dos três *sites* de supermercados

```
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
```

Quadro 11 - Uma consulta SQL sobre preços de produtos

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 irão compor o PEC. Tem-se, para árvores profundas à esquerda, as seguintes possíveis ordens de avaliação dos predicados:

- O_1 = L1→ S1→ S2→ S3
- O_2 = L1→ S1→ S3→ S2
- O_3 = L1→ S2→ S1→ S3

- O_4 = L1→ S2→ S3→ S1
- O_5 = L1→ S3→ S1→ S2
- O_6 = L1→ S3→ S2→ 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, características deste cenário. Estratégias puramente estáticas (tradicionais) não são capazes de modelar tais variações. Já estratégias adaptáveis podem combinar os diversos planos possíveis durante a execução de uma consulta recorrendo aos operadores que estejam mais disponíveis a cada instante. Para este exemplo de integração, uma MEC executará o PEC equivalente à $O_1 \cup O_2 \cup O_3 \cup O_4 \cup O_5 \cup O_6$. O PEC produzido contém os operadores algébricos (relacionais) Scan, Bindaccess (Ba), BindJoin (Bj) e Project que são combinados de acordo com o modelo de execução indicado na Figura 49 que contém os módulos *Fixed, Demand-drive, Wait, Adaptive, Remote e First-tuple.* A Figura 50 mostra a execução do PEC adaptável descrita a seguir.

O módulo *Fixed* contém o operador Scan, que acessa a fonte local L1, o operador Project, que produz as tuplas resultantes da consulta e o *distributor* responsável pela adaptabilidade dos BindJoins. O modulo *Adaptive* instancia os seguintes operadores:

- os operadores *BindJoin* (Bj), os quais utilizam os operadores BindAccess para submeter, de forma distribuída, *bindings* (Florescu et al., 1999) às fontes S1 , S2 e S3, respectivamente;
- os operadores *active*, os quais coletam dados de seus produtores e os envia ao operador *distributor* agindo como escalonador ativo;
- os operadores *wait*, os quais solicitam dados ao operador *distributor*
- o operador *distributor* que gerencia a adaptabilidade das tuplas ainda não processadas armazenando-as na sua fila.

Além disso, para garantir que as tuplas estejam disponíveis ao usuário assim que sejam processadas por todos os BindJoins, elas são armazenadas na fila com uma prioridade, a qual é incrementada toda vez que a tupla for processada por um

dos operadores. Desta forma, as tuplas que completam primeiro o seu caminho são consumidas pelo Project. Os possíveis caminhos percorridos pelas tuplas, a partir do Scan, são equivalentes às ordens de avaliação dos predicados da consulta, ou seja, O_1 , O_2 , O_3 , O_4 , O_5 *e* O_6.

O Módulo *Remote* gerencia a execução distribuída entre os operadores BindJoin e BindAccess, inserindo os operadores de controle *send* e *receive* entre eles, sendo que o *send* é instanciado num computador remoto e o *receive*, num computador local, onde é feita a integração. Os demais módulos funcionam da mesma forma que no exemplo anterior.

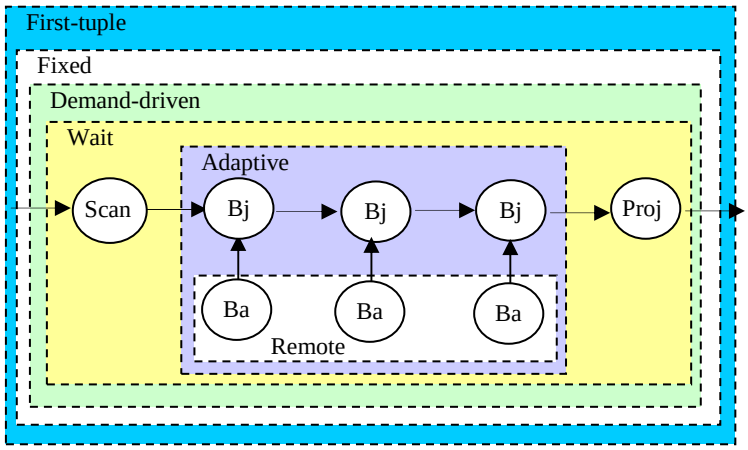


Figura 49 - Modelo de execução da consulta de integração

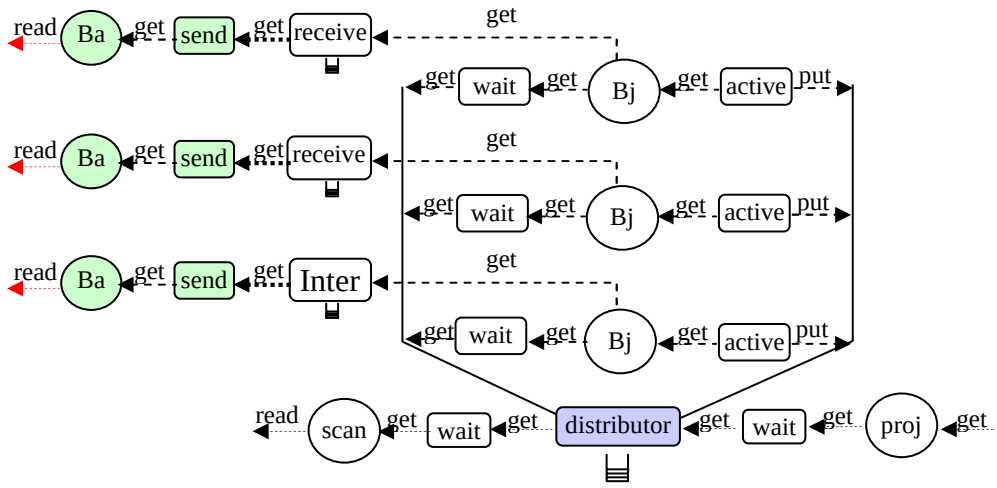


Figura 50 - Execução do PEC adaptável

```
<?xml version="1.0"?>
<METAPLANO>
<listadeoperadores>
       <operador id="1" tipo="algebrico" nome="Project">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="relacional"/>
              </parametro>
              <parametro tipo="projecao">
              <itemparametro tipo="atributo" valor="L1.nome"/>
              <itemparametro tipo="atributo" valor="S1.preco"/>
              <itemparametro tipo="atributo" valor="S2.preco"/>
              <itemparametro tipo="atributo" valor="S3.preco"/>
              </parametro>
       </operador>
       <operador id="2" tipo="algebrico" nome="BindJoin">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="relacional"/>
              </parametro>
              <parametro tipo="predicado">
              <itemparametro tipo="esquerda" valor="L1.cod"/>
              <itemparametro tipo="operacao" valor="="/>
              <itemparametro tipo="direita" valor="S3.cod"/>
              </parametro>
       </operador>
       <operador id="3" tipo="algebrico" nome="BindJoin">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="relacional"/>
              </parametro>
              <parametro tipo="predicado">
              <itemparametro tipo="esquerda" valor="L1.cod"/>
              <itemparametro tipo="operacao" valor="="/>
              <itemparametro tipo="direita" valor="S2.cod"/>
              </parametro>
       </operador>
       <operador id="4" tipo="algebrico" nome="BindJoin">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="relacional"/>
              </parametro>
              <parametro tipo="predicado">
              <itemparametro tipo="esquerda" valor="L1.cod"/>
              <itemparametro tipo="operacao" valor="="/>
              <itemparametro tipo="direita" valor="S1.cod"/>
              </parametro>
       </operador>
       <operador id="5" tipo="algebrico" nome="BindAccess">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="relacional"/>
              </parametro>
              <parametro tipo="fonte">
              <itemparametro tipo="nome" valor="S3"/>
              </parametro>
       </operador>
       <operador id="6" tipo="algebrico" nome="BindAccess">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="relacional"/>
              </parametro>
              <parametro tipo="fonte">
              <itemparametro tipo="nome" valor="S2"/>
              </parametro>
       </operador>
       <operador id="7" tipo="algebrico" nome="BindAccess">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="relacional"/>
              </parametro>
              <parametro tipo="fonte">
              <itemparametro tipo="nome" valor="S1"/>
              </parametro>
       </operador>
```

```
       </operador>
       <operador id="8" tipo="algebrico" nome="Scan">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="relacional"/>
              </parametro>
              <parametro tipo="fonte">
              <itemparametro tipo="nome" valor="L1"/>
              </parametro>
       </operador>
</listadeoperadores>
<MODULO><FIXED>
<MODULO><DEMAND-DRIVEN>
<MODULO><FIRSTTUPLE>
<MODULO><WAIT>
       <ALGEBRICO nome="Project" ref="1">
        <MODULO>
        <ADAPTIVE>
           <ALGEBRICO nome="Scan" ref="8"/>
              <ALGEBRICO nome="BindJoin" ref="2">
               <MODULO>
               <REMOTE>
                     <ALGEBRICO nome="BindAccess" ref="5"/>
               </REMOTE>
               </MODULO>
              </ALGEBRICO>
              <ALGEBRICO nome="BindJoin" ref="3">
               <MODULO>
               <REMOTE>
                     <ALGEBRICO nome="BindAccess" ref="6"/>
               </REMOTE>
               </MODULO>
              </ALGEBRICO>
              <ALGEBRICO nome="BindJoin" ref="4">
               <MODULO>
               <REMOTE>
                     <ALGEBRICO nome="BindAccess" ref="7"/>
               </REMOTE>
               </MODULO>
              </ALGEBRICO>
        </ADAPTIVE>
        </MODULO>
       </ALGEBRICO>
</WAIT></MODULO>
</FIRSTTUPLE></MODULO>
</DEMAND-DRIVEN></MODULO>
</FIXED></MODULO>
</METAPLANO>
```

Figura 51 - Especificação do meta-PEC

O ambiente de execução da consulta é similar ao do estudo de caso anterior exceto com relação a Metabase, cujas fontes são descritas na Figura 52 - as fontes S2 e S3 foram omitidas por serem similares a S1. A Figura 53 mostra a interface da aplicação para o estudo de caso, implementada utilizando-se *Servlets Java*. Observa-se nesta interface, que a ordem dos resultados não é a mesma dos dados locais, devido ao fato de que algumas tuplas percorrem caminhos diferentes.

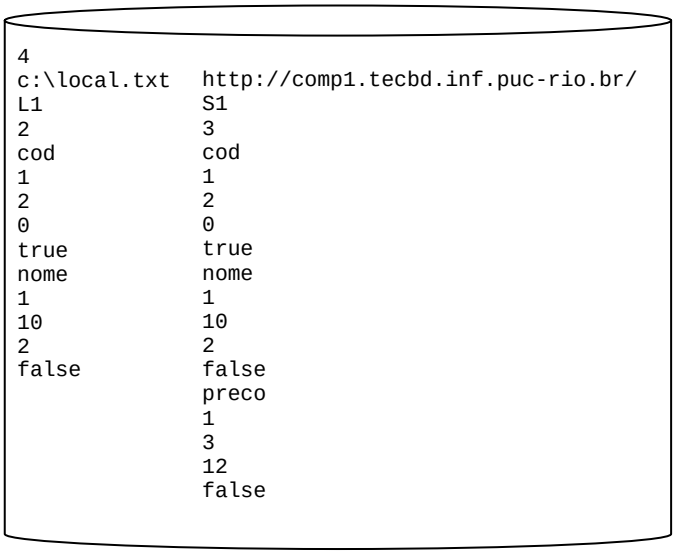


Figura 52 - A metabase da consulta de integração

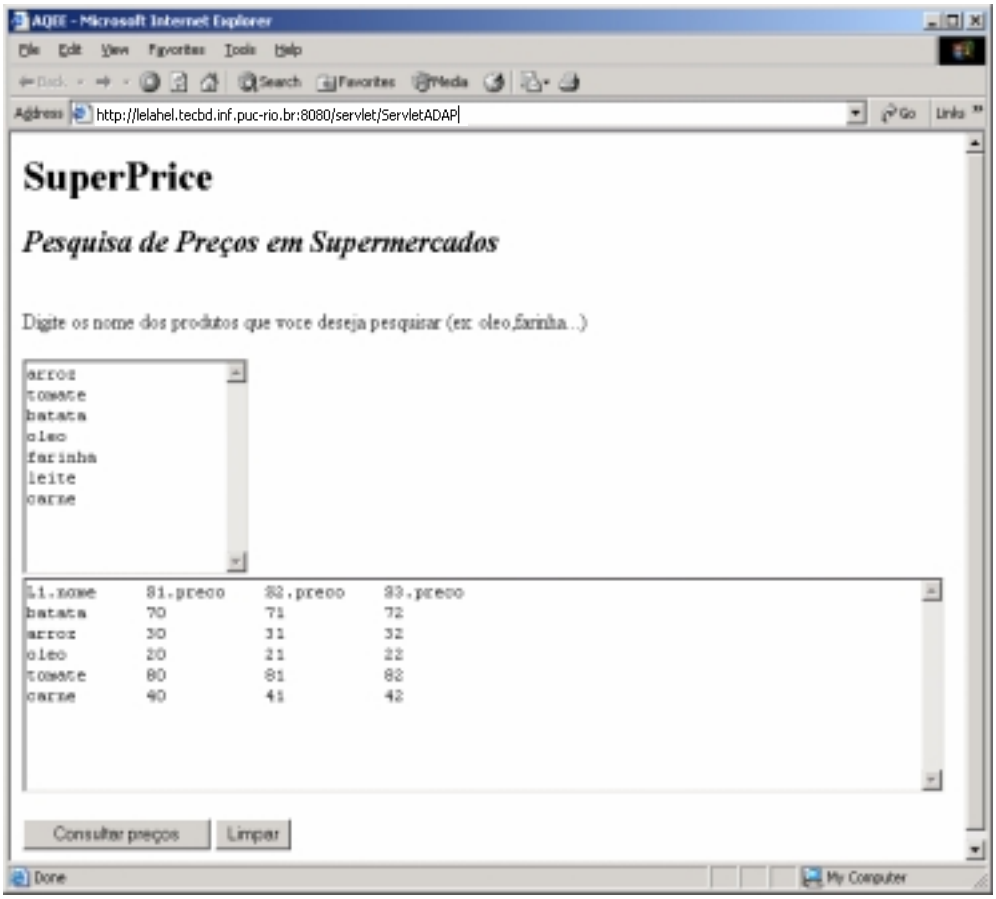


Figura 53 - A interface do usuário da aplicação

O processo de instanciação do *framework* QEEF para produzir o AQEE é similar ao do estudo de caso anterior, pois, utiliza-se também o modelo de dados relacional. As novas classes instanciadas são:

- AQEE – que recebe as requisições de consultas distribuídas;
- BindJoin e BindAccess – representam os novos operadores algébricos utilizados neste estudo de caso;
- Distributor, Wait e Active, Send e Receive – representam os operadores de controle homônimos
- TuplaAdp – especializa a classe Tupla para conter uma prioridade e um caminho, utilizados pelo operador distributor.
- FonteAscii – implementa uma interface para acesso, através de um Scan, a dados de arquivos ASCII.
- FonteWeb – implementa uma interface para acesso, através de um BindAccess, a dados de um *site web*.
- Queue e Buffer – representam as estruturas usadas pelos operadores distributor e receive, respectivamente.

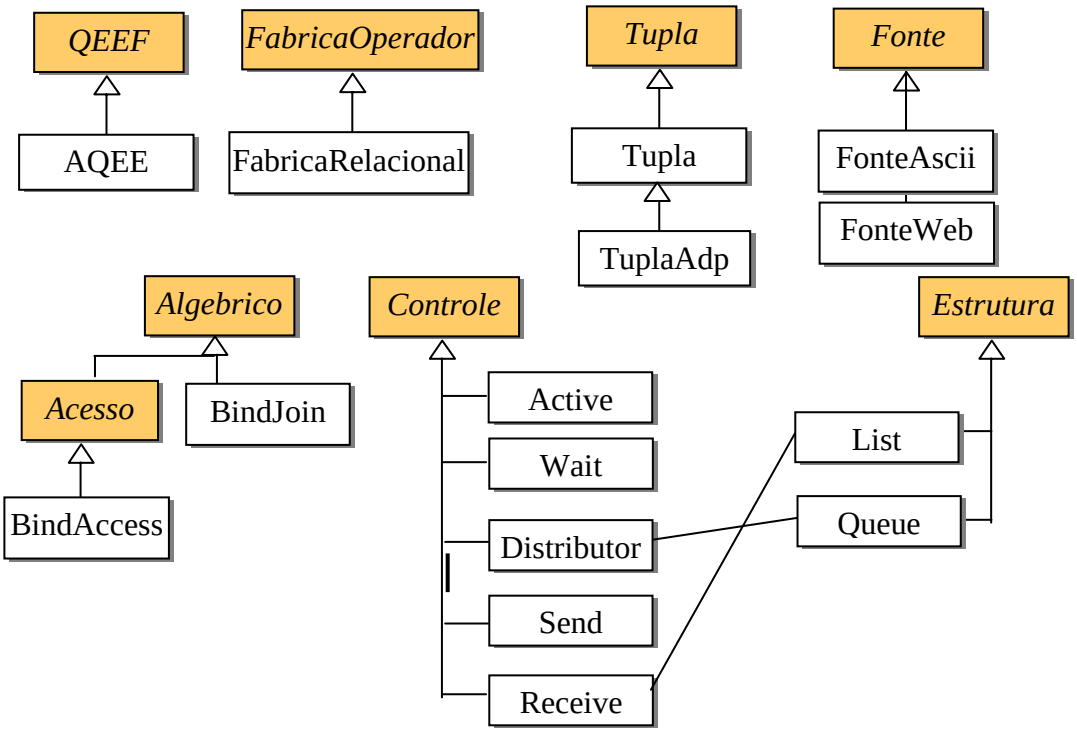


Figura 54 - Instanciação do QEEF para o AQEE

A implementação da classe AQEE é mostrada no Quadro 12. Neste caso, além das fábricas do exemplo anterior, é necessário a utilização de fábricas de módulos, de controles e de estruturas de dados.

```
public class AQEE extends QEEF {
 public AQEE ()  {
     super("AQEE");
     this.inserirFabricaAlgebrico(new FabricaRelacional() );
     this.inserirFabricaFonte (new FabricaFonteRelacional());
     this.inserirFabricaEstrutura(new FabricaEstrutura());
     this.inserirFabricaControle(new FabricaControle());
     this.inserirFabricaModulo(new FabricaModulo());
}
public static void main (String[] args)  {
      Log.open("Log.txt");
      AQEE maquina = new AQEE();
      maquina.open("Metabase.txt");
      maquina.execute("Plano.xml");
      while (maquina.hasnext()){
            Tupla t = maquina.getnext();
            System.out.println(t.toString());
      }
      maquina.close();
      Log.close();
}
```

Quadro 12 - A classe AQEE

6.3. XQEE

Nesta seção instancia-se o framework QEEF para o modelo de dados XML, obtendo-se a MEC denominada XQEE (*Xml Query Execution Engine*). Inicialmente, descreveremos o XQEE a partir de uma consulta XQuery e, a seguir, mostraremos a sua arquitetura e o seu modelo de execução.

Para demonstrar o funcionamento do XQEE, utilizou-se um dos exemplos (denominado de Q2) proposto pelo *XML Query Working Group* para testar aplicações da linguagem XQuery (XQuery, 2003). O Quadro 13 apresenta o conteúdo do documento bibliográfico *bib.xml* usado neste exemplo.

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price> 65.95</price>
    </book>
    <book year="1992">
        <title>Advanced Programming in the Unix environment</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
```

```
    </book>
    <book year="2000">
        <title>Data on the Web</title>
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price>39.95</price>
    </book>
    <book year="1999">
        <title>The Economics of Technology and Content for
                  Digital TV</title>
        <editor>
               <last>Gerbarg</last><first>Darcy</first>
               <affiliation>CITI</affiliation>
        </editor>
        <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
    </book>

</bib>
```

Quadro 13 - Documento bib.xml

A partir deste documento, deseja-se conhecer o título e autor de cada livro. A consulta XQuery para este exemplo é apresentada no Quadro 14.

```
<results>
  {
    for $b in doc("http://bstore1.example.com/bib.xml")/bib/book,
        $t in $b/title,
        $a in $b/author
    return
        <result>
            { $t }
            { $a }
        </result>
  }
</results>
```

Quadro 14 - Consulta XQuery sobre dados bibliográficos

As variáveis $b, $t e $a são denominadas variáveis de ligação (*bindings*) e contêm os caminhos usados para filtrar dados do documento num processo denominado de amarração. Embora a consulta acima contenha três variáveis, apenas as variáveis $t e $a são usadas na amarração cujos caminhos são, respectivamente “/bib/book/title” e “/bib/book/author”

O Quadro 15 mostra o resultado da consulta XQuery e identifica as tuplas XML produzidas pelo XQEE de acordo com o processo de amarração das variáveis $t e $a com o documento *bib.xml*

```
<results>
    <result>
        <title>TCP/IP Illustrated</title>
        <author>
            <last>Stevens</last>
            <first>W.</first>
        </author>
    </result>
    <result>
        <title>Advanced Programming in the Unix environment</title>
        <author>
            <last>Stevens</last>
            <first>W.</first>
        </author>
    </result>
    <result>
        <title>Data on the Web</title>
        <author>
            <last>Abiteboul</last>
            <first>Serge</first>
        </author>
    </result>
    <result>
        <title>Data on the Web</title>
        <author>
            <last>Buneman</last>
            <first>Peter</first>
        </author>
    </result>
    <result>
        <title>Data on the Web</title>
        <author>
            <last>Suciu</last>
            <first>Dan</first>
        </author>
    </result>
</results>
```

Tupla 1

Tupla 2

Tupla 3

Tupla 4

Tupla 5

Quadro 15 - Resultado da consulta XQuery

O Quadro 16 apresenta o PEC contendo os operadores envolvidos na execução da consulta XQuery acima. Os operadores ScanXML e ProjectXML foram implementados segundo a álgebra do modelo de dados XML (ver Capítulo 2).

```
<METAPLANO>
  <listadeoperadores>
       <operador id="1" tipo="algebrico" nome="ScanXML">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="xml"/>
              </parametro>
              <parametro tipo="fonte">
              <itemparametro tipo="nome" valor="bib.xml"/>
              </parametro>
              <parametro tipo="amarracao">
              <itemparametro tipo="caminho" valor="/bib/book/title"/>
              <itemparametro tipo="caminho" valor="/bib/book/author"/>
              </parametro>
       </operador>
       <operador id="2" tipo="algebrico" nome="ProjectXML">
              <parametro tipo="algebra">
              <itemparametro tipo="nome" valor="xml"/>
              </parametro>
              <parametro tipo="construcao">
```

```
                <itemparametro tipo="caminho" valor="/result”>
                </parametro>
       </operador>
  </listadeoperadores>
<MODULO><FIXED>
<MODULO><DEMAND-DRIVEN>
<MODULO><FIRSTTUPLE>
<MODULO><WAIT>
       <ALGEBRICO nome="ProjectXML" ref="2">
              <ALGEBRICO nome="ScanXML" ref="1"/>
       </ALGEBRICO>
</WAIT></MODULO>
</FIRSTTUPLE></MODULO>
</DEMAND-DRIVEN></MODULO>
</FIXED></MODULO>
</METAPLANO>
```

Quadro 16 - PEC da consulta XQuery

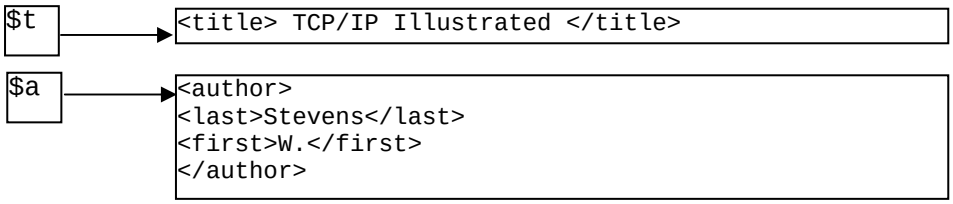
O operador ProjectXML consome as tuplas XML produzidas pelo ScanXML, a partir da extração de dados do documento original e sem o conhecimento prévio da estrutura do documento, isto é, sem a utilização de DTDs e na ordem em que os dados são lidos. Cada tupla é produzida através da combinação dos elementos amarrados às variáveis de ligação.

O operador ScanXML é executado da seguinte forma: na sua inicialização (método *open*) é instanciado um analisador baseado na API SAX (*SAXReader*) que analisa cada linha do documento. Neste momento, os nós do documento são identificados (nome, nível, caminho, etc) até que seja produzida uma tupla XML, contendo apenas os nós amarrados às variáveis.

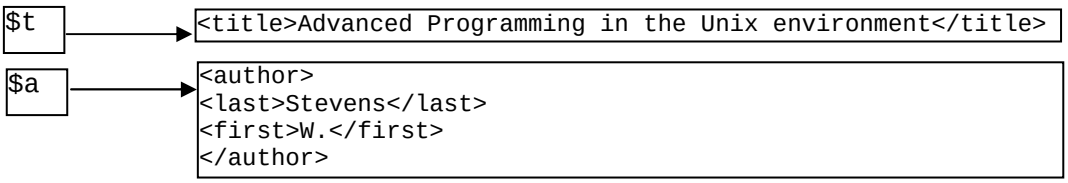
O processo de amarração consiste em comparar os caminhos das variáveis com os caminhos dos nós do documento. Quando um caminho denotado por um elemento casa com um dos caminhos especificados pelas variáveis, o elemento (ou sub-elemento) será amarrado à respectiva variável. Por exemplo, a Figura 55 mostra as amarrações possíveis para variáveis $t e $a durante o processamento do documento *bib.xml*, sendo que a amarração da variável $t contém o elemento <title> e a amarração da variável $a contém o elemento <author> e seus sub-elementos <last> e <first>. Isto significa que o caminho da variável $t casa com o caminho do elemento <title> e o caminho da variável $a casa com o caminho do elemento <author> e de seus sub-elementos. É importante notar que no caso (c) a variável $a terá três amarrações e no caso (d) ela não terá nenhuma amarração. Essas duas variáveis são atualizadas à medida que o documento é lido.

Uma tupla é construída a partir da combinação ta toda vez que as variáveis $t e $a completam uma amarração. A Figura 56 mostra todas as tuplas

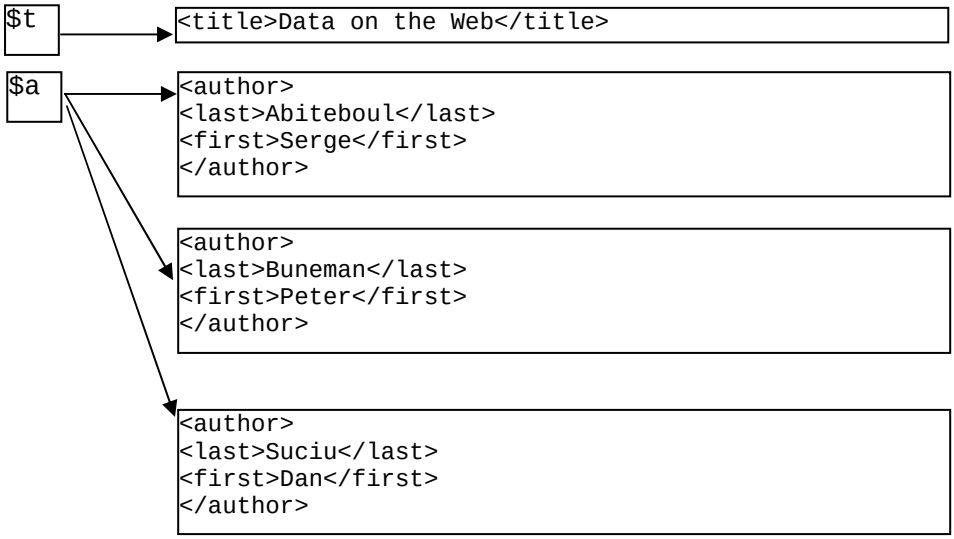
produzidas a partir das amarrações (Figura 55), onde cada uma armazena o seu nível hierárquico, o seu caminho no documento e o seu nome. As tuplas 1 e 2 são produzidas a partir das combinações ta das amarrações de (a) e (b) e as tuplas 3, 4 e 5 são produzidas, respectivamente, a partir das combinações $t(1)$a(1), $t(1)$a(2) e $t(1)$a(3) das amarrações de (c), onde $t(i)$a(j) refere-se a uma combinação da amarração i de $t com a amarração j de $a. Observa-se que a amarração da variável $t (d) não é suficiente para produzir tuplas XML.



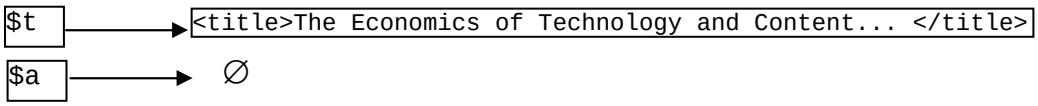
a)



b)



c)



d)

Figura 55 - Amarrações das variáveis de ligação

Tupla

1	/book/title	TCP/IP Illustrated
1	/book/author	
2	/book/author/last	Stevens
2	/book/author/first	W.

Tupla

1	/book/title	Advanced Programming in the Unix environment
1	/book/author	
2	/book/author/last	Stevens
2	/book/author/first	W.

Tupla

1	/book/title	Data on the Web
1	/book/author	
2	/book/author/last	Abiteboul
2	/book/author/first	Serge

Tupla

1	/book/title	Data on the Web
1	/book/author	
2	/book/author/last	Buneman
2	/book/author/first	Peter

Tupla

1	/book/title	Data on the Web
1	/book/author	
2	/book/author/last	Suciu
2	/book/author/first	Dan

Figura 56 - Tuplas XML produzidas a partir das amarrações

O processo de instanciação do *framework* QEEF para produzir o XQEE é feito por herança como mostra a Figura 57. As classes instanciadas são:

- XQEE – que recebe as requisições de consultas da aplicação usuária
- FabricaXML – responsável pela criação dos operadores da álgebra XML.
- ScanXML e ProjectXML – representam os operadores algébricos utilizados neste estudo de caso.
- TuplaXML – representa as unidades de dados processadas pelos operadores XML.
- ElementoXML e AtributoXML – compõem a estrutura da Tupla XML
- FonteXML – fornece dados para o ScanXML utilizando um analisador de documentos XML.

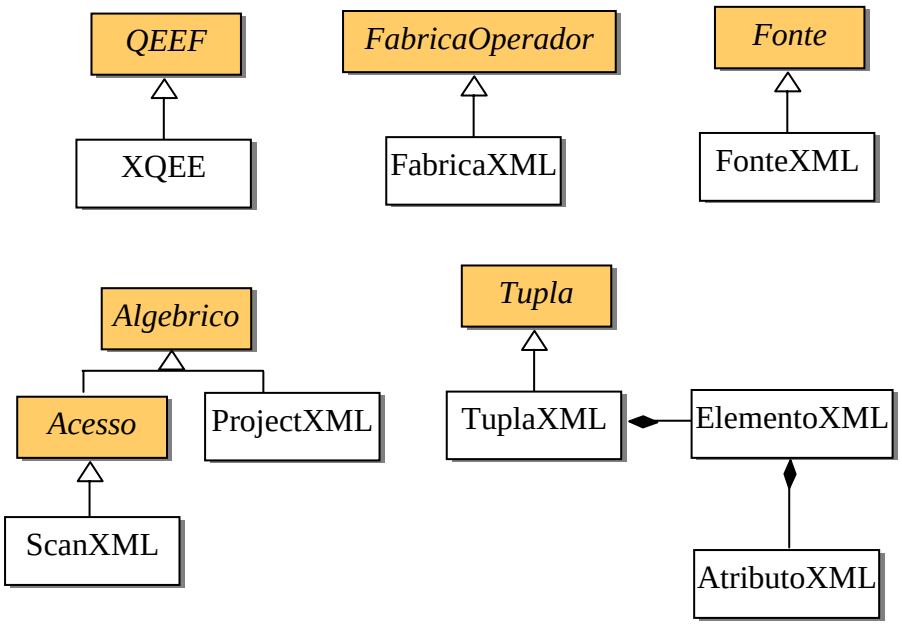


Figura 57 - As classes instanciadas do QEEF para o XQEE

O Quadro 17 mostra a implementação da Classe XQEE na linguagem Java. As tuplas XML são consumidas através do método *getnext* e visualizadas através do método *toString* usado para serializar as tuplas. Note que o consumo de tuplas é similar ao das outras máquinas. O modelo de execução utilizado neste exemplo é o modelo baseado em streams (ver Capítulo 5), dado que não se conhece o esquema (DTD) da fonte de dados.

Cada tupla produzida pelo ScanXML pode ser acessada por outros operadores de duas formas:

- o ScanXML recebe uma solicitação de seu consumidor via método *get* (no módulo *Demand-driven*) ou
- o ScanXML envia a tupla produzida para seu consumidor via método *put* (no módulo *Data-driven*)

```
public class XQEE extends QEEF {
public XQEE() {
     super("XQEE");
     this.inserirFabricaAlgebrico(new FabricaXML() );
     this.inserirFabricaFonte (new FabricaFonteXML());
     this.inserirFabricaEstrutura(new FabricaEstrutura());
     this.inserirFabricaControle(new FabricaControle());
     this.inserirFabricaModulo(new FabricaModulo());
}
public static void main (String[] args)  {
      Log.open("Log.txt");
      XQEE maquina = new XQEE();
      maquina.open("Metabase.txt");
      maquina.execute("Plano.xml");
      while (maquina.hasnext()){
            Tupla t = maquina.getnext();
            System.out.println(t.toString());
      }
      maquina.close();
      Log.close();
}
```

Quadro 17 - A classe XQEE

6.4. Síntese do Capítulo

Neste capítulo, foram apresentadas as MECs RQEE, AQEE e XQEE, instanciadas a partir do QEEF para diferentes modelos de dados e de execução, de forma ortogonal. Em cada MEC, foram mostradas as classes especializadas, a partir do QEEF, e a especificação de um meta-PEC em XML, baseado numa consulta submetida pela aplicação. O uso de um modelo de execução é feito num meta-PEC através da especialização de classes referentes às fábricas de módulos e controles e da combinação de módulos com os demais operadores do plano algébrico da consulta. O uso de um modelo de dados é feito através da especialização de classes referentes às fábricas de operadores (algébricos e inter-algébricos) e de fontes de dados.