4
A Máquina Extensível

Para a construção da MEC extensível utilizou-se a técnica de *framework* de *software*, obtendo-se o *framework* QEEF *(Query Execution Engine Framework)* (Ayres et al., 2003). A escolha da técnica de *framework* deve-se principalmente ao fato dela ter sido bastante utilizada na construção de sistemas flexíveis e que podem ser adaptados com maior rapidez e facilidade para atender aos requisitos de novas aplicações. Espera-se com isso que a flexibilidade oferecida pelo QEEF reduza o esforço de construção de novas MECs. Neste capítulo, é apresentada a técnica de *frameworks* de *software* e, a seguir, é descrito o *framework* QEEF.

4.1.
Introdução a *Frameworks* de *Software*

Segundo Fayad (Fayad et al., 1999), um *framework* é uma aplicação reusável e semi-completa que pode ser especializada para produzir aplicações customizadas. A reutilização de um framework se dá através de sua instanciação que é o processo de adaptação da solução geral às características particulares de um determinado problema. Em termos estruturais, um framework é uma arquitetura de software composta por uma parte fixa (*frozen-spots*), que incorpora os aspectos estáticos de um domínio de aplicação, e por pontos de adaptação (*hot-spots*), onde estão presentes os aspectos variáveis do domínio. O projeto de um framework especifica uma coleção de classes concretas e abstratas, e um padrão de colaboração entre estas classes que definem uma arquitetura de referência para toda uma família de aplicações.

Um *framework* deve ser construído através de sucessivas etapas de análise e síntese. Nas etapas de análise, o estudo das aplicações pré-existentes, somado ao conhecimento adquirido nas iterações anteriores, é usado para descobrir as estruturas básicas do domínio. A etapa de síntese produz, através de um processo de generalização, uma coleção de componentes abstratos que irá definir uma arquitetura específica para um domínio de aplicações.

A instanciação de um *framework* consiste em implementar código específico nos seus pontos de adaptação pré-definidos, através de duas técnicas básicas: herança e composição.

- na instanciação por herança a adaptação é feita inserindo-se subclasses para implementar os métodos abstratos, para redefinir os métodos que tenham implementação *default* ou para acrescentar novos atributos.
- Na instanciação por composição utiliza-se componentes pré-fabricados na adaptação do *framework*. Neste caso não é necessário conhecer o projeto do *framework* com grande profundidade, já que a adaptação será feita conectando-se os componentes, que muitas vezes são fornecidos com o próprio *framework*, nos seus *hots-spots*.

Os frameworks podem ser classificados de acordo com as técnicas usadas para instanciação podendo variar entre:

- *Frameworks* Caixa-Branca *(White-Box):* quando são adaptados através de herança. Isso requer do desenvolvedor que o projeto do *framework* seja bem compreendido para que este método seja eficaz.
- *Frameworks* Caixa-Preta *(Black-Box):* quando são adaptados através de composição. Isso requer do desenvolvedor o conhecimento das interfaces externas dos componentes para que a instanciação possa ser realizada, não havendo necessidade de se conhecer os detalhes internos de implementação dos componentes.

Além disso, um *framework* pode apresentar alguns *hot-spots* que são implementados por herança e outros que são implementados por composição. De um modo geral, à medida que um domínio vai se tornando mais conhecido e vão diminuindo as mudanças sofridas por eles, os *frameworks*, para estes domínios, tendem a ser predominantemente caixa-preta, já que o conjunto de variações existentes tende a se estabilizar.

Os *frameworks* podem ser classificados de acordo com a sua aplicação:

- *Frameworks* de Infra-estrutura: são utilizados para simplificar o desenvolvimento de sistemas de infra-estrutura portáteis e eficientes, tais como: sistemas operacionais e banco de dados;

- *Frameworks* de Integração (*middleware*): são utilizados para integrar aplicações e componentes distribuídos
- *Frameworks* de Aplicação: são utilizados como base das atividades de negócio de grandes domínios de aplicação tais como: finanças e telecomunicações.

Em geral, o processo de desenvolvimento de *frameworks* é mais complexo do que o processo tradicional de desenvolvimento de software devido a dois aspectos fundamentais:

- enquanto que no processo tradicional o objetivo é produzir uma única aplicação, o desenvolvimento de *frameworks* irá produzir um projeto genérico para toda uma família de aplicações.
- o processo de desenvolvimento de *frameworks* possui, adicionalmente, a etapa de instanciação, onde membros específicos de uma família de aplicações são produzidos.

No entanto, o esforço de se construir uma arquitetura flexível através da técnica de *frameworks* pode se justificar na redução do custo de se desenvolver diversas aplicações para um mesmo domínio a partir do seu início. Além disso, a flexibilidade oferecida pelo *framework* permite se instanciar uma nova aplicação contendo uma combinação de requisitos não presentes nas aplicações pré-existentes ao *framework*.

A solução proposta nesta tese utiliza a técnica de *framework* no domínio de MECs devido aos seguintes motivos:

- reduzir o custo de desenvolvimento de MECs, a partir do seu início;
- criar uma MEC extensível para diferentes modelos de execução e para diferentes modelos de dados;
- permitir a instanciação de novas MECs com funcionalidades não presentes nas MECs existentes, tal como, a execução de um PEC que combine diferentes modelos de execução e diferentes modelos de dados (por exemplo, modelo relacional e modelo XML) de maneira ortogonal.

4.2. O *Framework* QEEF

O QEEF é um *framework* de *software* do tipo caixa-branca composto de uma parte fixa (*frozen-spots*), que permite o reuso de uma infra-estrutura básica de uma MEC, e de uma parte adaptável (*hot-spots),* que deve ser instanciada segundo os seus pontos de adaptação. O QEEF permite ser instanciado para diferentes modelos de dados e de execução, de forma ortogonal. Como resultado desta instanciação, tem-se uma MEC capaz de executar PECs baseados nestes modelos.

O desenvolvimento do framework QEEF seguiu as seguintes etapas:

1. Análise das arquiteturas das MECs existentes na literatura;
2. Especificação do QEEF com a identificação dos pontos fixos (*frozen spots*) e adaptáveis (*hot spots*);
3. Implementação (em *Java*) da infra-estrutura básica do QEEF contendo os principais conceitos associados à arquitetura de uma MEC;
4. Definição de um meta-modelo de execução para suportar novos modelos de execução;
5. Instanciação do QEEF para diferentes modelos de execução e de dados.

A seguir, é descrito o processo de desenvolvimento do QEEF de acordo com estas etapas, sendo que a etapa 4 será descrita no próximo capítulo e a etapa 5 será descrita no capítulo de estudos de casos.

4.2.1. Análise de Arquiteturas de MECs

Embora diferentes arquiteturas de Máquinas de Execução de Consultas (MEC) sejam encontradas na literatura, onde cada uma implementa um modelo específico (de execução e de dados), não é conhecido um estudo que as classifique segundo suas funcionalidades. Adicionalmente, não é fácil encontrarmos uma separação nítida entre as MECs e os demais componentes de um sistema de processamento de consultas. No entanto, considera-se neste trabalho que uma MEC pode ser definida pelos seguintes elementos:

- Unidade de Execução – são os operadores;

- Unidade de Dados – correspondem às tuplas de dados processadas pelos operadores;
- Estruturas de Dados – correspondem às estruturas necessárias para armazenar um conjunto de tuplas durante o processamento dos operadores;
- Modelo de execução – diz respeito às técnicas utilizadas para produzir um fluxo de dados que atenda aos requisitos de um cenário de aplicações;
- Modelo de dados – diz respeito a álgebra utilizada pelos operadores e os métodos de acesso às fontes de dados.

Quanto aos operadores, eles são classificados em algébricos e de controle. Conforme apresentado no Capítulo 2, os operadores algébricos implementam uma determinada álgebra e os operadores de controle, são meta-operadores (Graefe, 1993) que exercem a comunicação inter-operadores.

Quanto às unidades de dados, elas devem ser definidas segundo um modelo de dados.

Quanto às estruturas de dados, tais como, listas, sacolas, conjuntos e árvores, são usadas para suportar a implementação dos operadores da MEC.

Quanto aos modelos de execução e de dados, todos os PECs são executados segundo um mesmo modelo de execução e de dados.

Adicionalmente, do ponto de vista de uso, uma MEC pode ser especificada segundo uma interface que apresente os métodos necessários para a sua utilização, incluindo a submissão de um PEC. Uma mesma MEC pode apresentar diferentes interfaces.

4.2.2. Especificação do QEEF

A Figura 37 mostra a arquitetura do *framework* QEEF em um diagrama de classes UML (em alto nível), onde as classes Concretas representam os pontos fixos (na cor cinza) e as classes Abstratas representam os pontos de adaptação, os quais devem ser instanciadas de acordo com os modelos de dados e de execução desejados. A seguir detalharemos esta arquitetura.

A classe *QEEF* implementa o padrão de projeto *facade* (Gamma et al., 1995) responsável pela interface da máquina de execução com a aplicação usuária que submete um Plano de Execução de Consulta descrito em XML (PEC-XML), cuja especificação será apresentada no próximo capítulo.. Esta classe possui métodos para inicialização, submissão de um PEC-XML, obtenção dos resultados e finalização.

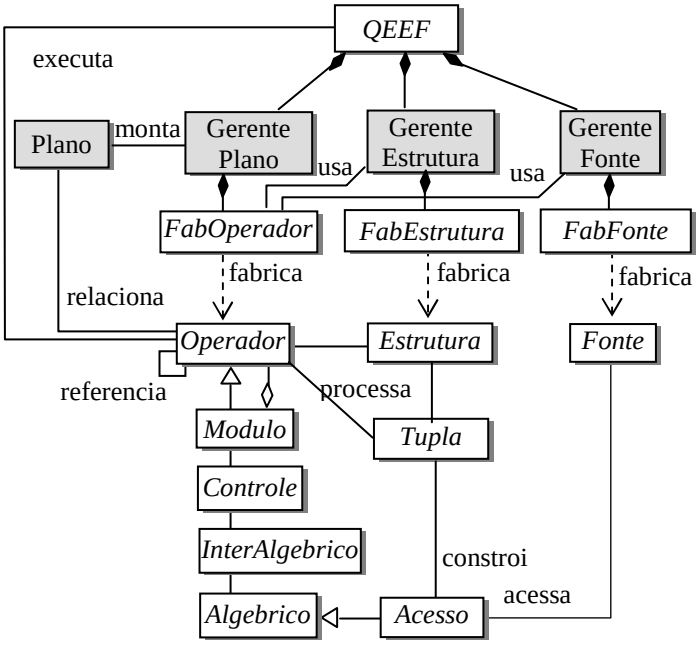

Figura 37 - O Framework QEEF

A classe GerentePlano (padrão de projeto *singleton*) interpreta um PEC-XML e gerencia a montagem do plano (classe Plano), utilizando uma ou mais fábricas de operadores para fabricar os operadores, de acordo como suas classes (*Algebrico, InterAlgébrico,* de *Controle e Módulo),* inserindo-os no plano na forma de produtor e/ou consumidor, resultando numa árvore de operadores. Existe uma fábrica para cada classe de operadores e todas são sub-classes da classe *FabricaOperador* e devem ser fornecidas na instanciação do QEEF (ver próxima seção). A classe GerentePlano pode utilizar mais de uma fabrica de operadores algébricos, uma para cada álgebra, permitindo a criação de operadores de diferentes álgebras.

A classe *Algebrico* generaliza todos os operadores algébricos e a classe *Acesso* representa os operadores algébricos que acessam uma fonte de dados para criar as tuplas de dados (classe *Tupla*), as quais serão processadas pelos demais

operadores do PEC, podendo sofrer atualizações ou descarte. A classe *InterAlgebrico* representa os operadores de conversão e deve ser especializada quando há a utilização de mais de uma álgebra.

A classe *Controle* generaliza todos os operadores de controle e tem a importante função de desacoplar os operadores algébricos. O QEEF instancia vários tipos de controle tais como os mostrados na Figura 38.

A classe *Modulo* generaliza todos os módulos de execução. A implementação de um módulo requer a utilização de operadores de controle, os quais são inseridos no PEC de acordo com a lógica de implementação do módulo. A Figura 38 mostra exemplos de módulos e os operadores de controles que eles utilizam. Os módulos podem ser combinados com outros operadores (incluindo módulos) segundo o modelo de execução descrito no PEC-XML, como será visto no próximo capítulo. Para permitir esta combinação implementa-se o padrão de projeto *Decorator*.

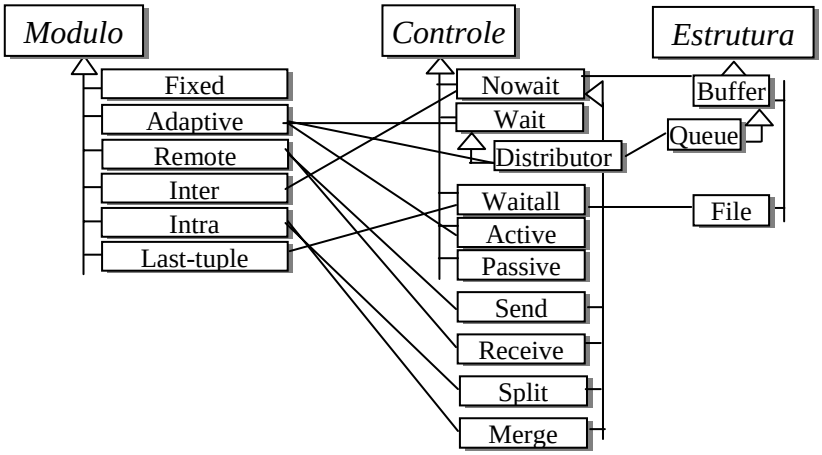

Figura 38 - Relacionamentos entre Módulos, Controles e Estruturas

Os operadores podem conter uma ou mais estruturas de dados (classe *Estrutura*), tais como listas (Buffer), filas (Queue) e arquivos (File), para armazenamento de tuplas de dados durante a execução (ver Figura 38). Essas estruturas são criadas através da classe GerenteEstrutura (padrão de projeto *singleton*), que gerencia a memória consumida pelo plano e a persistência dos seus dados, quando necessária.

A classe *Fonte* encapsula o acesso dos operadores às fontes de dados e, por isso, deve ser especializada de acordo com o modelo de dados das fontes e deve ser adaptada, através do padrão de projeto *adapter*, às interfaces das fontes, tais

como: a interface *ResultSet*, para acesso a fontes do padrão JDBC-ODBC, e a interface *ContentHandler*, para uso de um analisador XML sobre fontes de dados XML. As fontes são fabricadas pela classe FabricaFonte e gerenciadas pela classe GerenteFonte (padrão de projeto *singleton*), a partir da descrição de seus metadados.

A execução de um plano dá-se através da invocação de métodos dos seus operadores. Para isso, a classe Operador implementa uma interface comum a todos os operadores, baseada na classe iterador (descrita no Capítulo 2) estendida para suportar o fluxo de controle data-driven, contendo os seguintes métodos*:*

open: prepara o operador para produzir dados

hasnext: verifica se o operador produziu algum dado

getnext: recebe um dado sob demanda do consumidor

putnext: produz um dado e envia para o consumidor

close: finaliza o operador

Alguns desses métodos são mostrados no Quadro 4. Note que a implementação dos métodos *getnext* e *putnext* é genérica para todos os operadores e que a implementação do método *hasnext* é específica para cada operador (neste caso, o Project).

```
public abstract class Operador{
...
  public Tupla getnext(){
            if (hasnext) {hasnext=false; return(thenext);}
            else       return(null) ;
  }
  public boolean putnext(Tupla t)      {
            if (consumidor.putnext(t)) return(true);
            else       return(false);
  }
}
public class Project extends Algebrico{
...
  public boolean hasnext()   {
            hasnext = produtor.hasnext();
            if (hasnext) {thenext = produtor.getnext();projetaratributos();}
            else       thenext = null;
            return(hasnext);
}}
```

Quadro 4 - Implementação dos métodos getnext, putnext e hasnext

A seguir, descreve-se o processo de execução de uma consulta. O Quadro 5 exemplifica uma aplicação que utiliza uma MEC instanciada a partir do QEEF.

Note que a MEC pode ser manipulada através de uma interface similar a dos operadores, descrita anteriormente. Após a criação de um objeto MEC, é feita a preparação das fontes de dados através do método *open*, onde é fornecido o arquivo Metabase.txt contendo a descrição das fontes de dados a serem criadas e conectadas antes da execução da consulta. Um PEC é submetido à MEC através do método *execute*, sendo fornecido um arquivo Plano.xml com a sua descrição (PEC-XML). Neste momento, é criado o plano de execução conforme descrito acima. Inicia-se a execução do plano através da invocação do método *open* da máquina que, por sua vez, invoca o método *open* do operador raiz do plano, de forma iteradora, até atingir os operadores folhas, onde cada operador consumidor invoca o método *open* do seu operador produtor. De forma análoga, para fornecer cada tupla de dados para a aplicação usuária, invoca-se os métodos *hasnext* e *getnext* da máquina e de cada operador, produzindo tuplas ao longo do plano até que elas sejam consumidas pela aplicação usuária. O conteúdo das tuplas pode ser visualizado por um método de serialização (p.ex: *toString*). A execução do plano termina quando uma tupla vazia é produzida, e neste caso, é invocado o método *close* de cada operador do plano, para finalizar o seu processamento sem a sua destruição, permitindo a re-execução do plano. Caso a MEC utilize um modelo de execução baseado em fluxo de controle direcionado a dados (*data-driven*), a aplicação deverá implementar a interface Putnext contendo o método *putnext*, o qual permitirá o envio de tuplas, de forma iteradora, a partir do método *putnext* de cada operador. As mensagens produzidas são registradas num arquivo (Log*)*.

```
public class Aplicacao{
  public static void main (String[] args)   {
            Log.open("Log.txt");
            try {     MEC maquina = new MEC();
                      maquina.open("Metabase.txt");
                      maquina.execute("Plano.xml");
                      while (maquina.hasnext()) {
                                Tupla t = maquina.getnext();
                                System.out.println("resposta=>"+ t.toString() );
                      }
                      maquina.close();
            }
            catch (Exception e)  { Log.println(e); }
            Log.close();
 }}
```

Quadro 5 - Exemplo de uso de uma MEC

4.2.3. Instanciação do QEEF

O *framework* QEEF deve ser instanciado para um ou mais modelos de dados e para um modelo de execução, produzindo uma MEC específica. Para permitir isso, faz-se necessária a instanciação dos seguintes pontos de adaptação (*hot-spots*):

- **tupla de dados:** instanciadas de acordo com o modelo de dados desejado (exemplos: relacional, XML)
- **fonte de dados:** instanciadas de acordo com o modelo de dados e do tipo de fonte (arquivo, SGBD, fonte *web*, serviço *web*, etc);
- **operadores algébricos:** instanciados de acordo com a semântica do modelo de dados e de acordo com o tipo do operador (acesso, processo ou resultado);
- **operadores de controle:** instanciados de acordo com a necessidade de implementação dos módulos de execução (exemplos: leitor, coletor, distribuidor);
- **módulos de execução:** instanciado de acordo com o modelo de execução do cenário escolhido;
- **estruturas de dados:** instanciadas a partir das necessidades de armazenamento de tuplas pelos operadores (exemplos: filas com ou sem prioridade, buffer de páginas);

Além da instanciação das classes referentes a esses pontos de adaptação, faz-se necessária a instanciação das fábricas de operadores, de acordo com o modelo de dados, como mostrado no Quadro 6 para dois modelos de dados hipotéticos (M1 e M2). Logo, deve-se fornecer, para cada modelo de dados, as suas fábricas de fontes de dados e de operadores algébricos. Além disso, caso haja mais de um modelo de dados, é necessária a inclusão da fábrica de operadores inter-algébricos, para conversão de dados entre esses modelos. Adicionalmente, é opcional a inclusão de uma fábrica de módulos e de operadores de controle. A instanciação do QEEF para um modelo de execução será discutida no próximo capítulo. Para completar a instanciação do QEEF, pode-se definir uma fábrica de estruturas de dados, as quais podem ser utilizadas por todos os operadores.

```
public class MEC extends QEEF{
  public MEC ()   {
      super ("MEC");
      this.inserirFabricaAlgebrico  (new FabricaAlgebricoM1() );
      this.inserirFabricaFonte      (new FabricaFonteM1());
      this.inserirFabricaAlgebrico  (new FabricaAlgebricoM2() );
      this.inserirFabricaFonte      (new FabricaFonteM2());
      this.inserirFabricaInterAlgebrico (new FabricaConversao());
      this.inserirFabricaEstrutura  (new FabricaEstrutura());
      this.inserirFabricaControle   (new FabricaControle());
      this.inserirFabricaModulo     (new FabricaModulo());
}}
```

Quadro 6 - Instanciação do QEEF

4.3. Síntese do Capítulo

Neste capítulo, foi apresentada a MEC Extensível, denominada de QEEF (*Query Execution Engine Framework*), construída através da técnica de *framework* de *software* que tem sido usada para a construção de sistemas flexíveis e reutilizáveis. Para obtenção do QEEF, foi realizado um estudo sobre arquiteturas de MECs. A especificação do QEEF foi feita em UML e sua implementação foi feita em Java. Foi mostrado um exemplo de uso e de instanciação para diferentes modelos de dados. A seguir, será apresentado o meta-modelo QUEM, o qual permite a instanciação do QEEF para diferentes modelos de execução.