3
A Solução Proposta

Este capítulo apresenta a base da solução proposta que consiste no suporte a diferentes modelos de execução e de dados, de forma ortogonal.

3.1.
Suporte a Diferentes Modelos de Execução de Consultas

Tradicionalmente, uma MEC é construída para suportar um modelo de execução de consultas específico, presente num certo cenário. Neste contexto, os PECs submetidos à MEC contêm operadores (algébricos e de controle) e são executados sempre utilizando um único modelo de execução que, em geral, fica implícito e codificado na sua implementação. Por exemplo, a MEC tradicional executa PECs baseando-se no modelo tradicional.

Para permitir o suporte a diferentes modelos de execução apresentam-se, nesta seção, novos conceitos de execução, tais como: características e módulos de execução de consultas, e sua combinação com os demais conceitos apresentados no capítulo anterior. Todos os conceitos relativos à execução de consultas em uma MEC estão relacionados na Tabela 3 em níveis de composição (1 a 6) e em níveis de abstração (Físico e Lógico), onde os conceitos físicos implementam os conceitos lógicos.

Nível	Conceito Físico	Conceito Lógico
1	MEC	Cenário de aplicações
2	PEC	Modelo de execução
3	Modulo	Característica de execução
4	Operador de controle	Tipo de comunicação
5	Operador algébrico	Álgebra (modelo de dados)
6	Tupla	Estrutura de dados (modelo de dados)

Tabela 3 - Níveis de composição dos conceitos de execução de consultas

No nível de composição 1 (o maior), tem-se a implementação de uma MEC específica para cada cenário de aplicações de consultas. No nível 2, um PEC implementa um modelo de execução utilizando-se diferentes módulos. No nível 3,

um módulo implementa um determinado estado de uma característica de execução, combinando um ou mais operadores de controle. No nível 4, tem-se a implementação de um operador de controle para um tipo de comunicação entre dois operadores algébricos. No nível 5, têm-se a implementação dos operadores algébricos baseados nas operações semânticas do modelo de dados. No nível 6 (o menor) tem-se a implementação de uma tupla de dados baseada numa estrutura de dados do modelo de dados.

Este trabalho de tese fundamenta-se nestes diferentes níveis de composição e, em particular, no fato de um PEC combinar módulos de execução, para permitir o suporte a diferentes modelos de execução.

A seguir, definem-se: (i) as características de execução relacionadas aos cenários apresentados no capítulo anterior; (ii) os módulos de execução, baseados nestas características de execução e (iii) os modelos de execução, a partir da combinação de módulos de execução.

3.1.1. Características de Execução de Consultas

A partir da análise dos cenários descritos no capítulo anterior, observamos algumas técnicas de execução, que denominamos de características de execução de consultas, que podem estar presentes em um ou mais cenários e que podem assumir diferentes estados dependendo do modelo considerado. Algumas destas características foram introduzidas por Graefe (Graefe, 1993). Considera-se que esta lista de características não é finita e que novas características podem surgir a partir de novos cenários.

Característica de Sincronismo

Define o estado da execução de um operador (consumidor) quando ele requer serviços ou dados de um outro operador (produtor). Existem três possíveis estados: *wait* (*pipeline* síncrono), *wait-all* (seqüencial síncrono) e *no-wait* (assíncrono).

No estado *wait*, o consumidor aguarda o seu produtor produzir um dado para, então, ativar o seu processamento; no estado *wait-all*, o consumidor aguarda o seu produtor produzir todos os dados para ativar o seu processamento; no estado

no-wait o produtor produz dados independentemente de solicitação do consumidor.

Característica de Paralelismo

O paralelismo ocorre quando um operador processa dados de forma assíncrona, ou seja, o consumo do dado não está sincronizado com a sua produção e vice-versa. Dentro de um mesmo PEC a características de paralelismo pode apresentar dois estados:

- *inter-operator* que ocorre entre operadores distintos e independentes
- *intra-operator* que ocorre entre várias instâncias do mesmo operador.

A escolha das características de paralelismo são usadas num PEC para se obter boas estratégias no processamento paralelo de consultas.

Característica de Distribuição

Refere-se à troca de dados entre pares de operadores em processos independentes. Apresenta dois estados possíveis: *remote* e *local*. O estado *remote* indica que um operador consumirá dados de um operador que está sendo executado em outro *site* (ou nó da rede), havendo a necessidade de comunicação inter-processo com este produtor. O estado *local* indica que o operador consumirá dados de um operador no mesmo processo.

Característica de Fluxo de Controle

Refere-se ao controle da transferência de dados entre operadores formando um encadeamento de execução entre consumidores e produtores. As alternativas são: fluxo de controle direcionado à demanda *(Demand-Driven),* onde cada operador produz dados sob a demanda do seu consumidor, e fluxo de controle direcionado a Dados (*Data-Driven)* - também conhecido como *dataflow control flow* - onde cada operador recebe dados à medida que são produzidos pelo seu produtor.

Característica de Fluxo de Dados

Especifica a ordem na qual os dados são processados. Existem dois estados: *Fixed*, onde os dados percorrem o mesmo caminho através dos operadores, e *Adaptive*, onde os dados percorrem caminhos diferentes através dos operadores dependendo do desempenho desses operadores e de um critério de distribuição - por exemplo, um determinado dado poderá ser processado pelo primeiro operador disponível ou pelo operador que já processou mais dados - sendo que os operadores mais rápidos processam mais dados do que os demais. Esta abordagem é usada pelo *Eddies* (Avnur&Hellerstein, 2000).

Característica de Tempo de Resposta

Refere-se ao momento no qual os resultados da consulta estarão disponíveis para a aplicação do usuário. Existem dois estados: *First-Tuple First*, requer que os dados sejam enviados para o usuário imediatamente à medida que sejam produzidos, e *Last-Tuple First*, requer que os dados sejam enviados para o usuário somente quando todos tiverem sido produzidos.

3.1.2. Módulos de Execução de Consultas

Define-se um módulo como um operador de mais alto nível que, em geral, insere operadores de controle entre os operadores algébricos de um PEC para atender a um determinado estado de uma característica de execução. A seguir são apresentados os módulos de execução associados a cada característica de execução.

Módulos de Sincronismo

A Figura 11 mostra os módulos de execução *Wait*, *Nowait* e *Waitall*, os quais implementam os estados da característica de sincronismo através dos operadores de controle homônimos. Nesta figura, as setas indicam o fluxo dos dados através dos operadores. Observa-se que nos dois últimos módulos, há necessidade de materialização de dados parcial e total, respectivamente. A seguir detalharemos as funcionalidades de cada módulo de sincronismo baseando-se nos

operadores algébricos op1 e op2, onde op2 consome dados produzidos a partir de op1, e num operador de controle, que encapsula os tipos de sincronismo entre eles.

No módulo *Waitall*, quando op2 solicita uma instância de dados para o operador *waitall* este, por sua vez, consome totalmente as tuplas produzidas por op1 materializando-as em disco, e a seguir, envia uma instância para op2. A cada novo pedido de op2, o operador *waitall* produzirá uma nova instância a partir do disco, até o seu final. Portanto, neste módulo o consumidor só produzirá uma instância quando forem processados todos os sub-planos de suas entradas. Embora esta técnica possa ter importância na execução de parte do PEC (como discutido no módulo de tempo de resposta), ela não é eficiente como um todo, pois o tempo gasto para entrada/saída pode ser custoso.

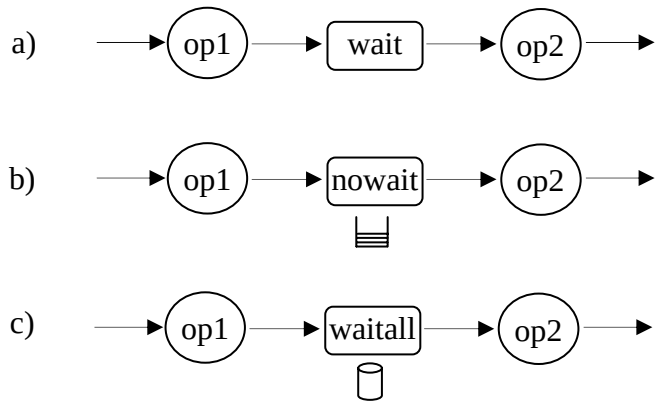

Figura 11 - Módulos de Sincronismo *Wait* (a), *Nowait* (b) e *Waitall* (c)

Nos módulos *Wait* e *Nowait*, a transferência de dados entre os operadores é feita em linha (*pipeline*), ou seja, os dados (tuplas) permanecem em memória durante o seu processamento, a não ser para o caso de insuficiência de memória, necessitando de uma gerência de memória virtual. A principal diferença entre os dois módulos está no fato de que no primeiro, o operador *wait* faz com que o processamento do consumidor (op2) esteja sincronizado com o processamento do produtor (op1), ou seja, o consumidor sempre fica esperando uma instância ser produzido pelo produtor. Já no segundo módulo, o processamento do consumidor e do produtor são assíncronos ou paralelos (como discutido na próxima seção), ou seja, o consumo de dados é independente de sua produção.

No módulo *Nowait*, ocorrem as duas situações distintas:

- Quando a taxa de consumo de Op2 é menor que a taxa de produção de Op1 os dados que chegam no operador *nowait* são armazenados temporariamente num *buffer* de memória (possivelmente virtual) e ficam aguardando para serem consumidos por Op2. O limitação no tamanho do buffer de memória pode conter a taxa de produção do nowait.
- Quando a taxa de consumo de Op2 é maior que a taxa de produção de Op1 o operador *nowait* passa a se comportar como um operador *wait*, não necessitando de um *buffer* de armazenamento.

No módulo *Wait*, o uso do operador *wait* pode ser descartado caso ele esteja consumindo dados de um operador algébrico, devido ao fato de que a linguagem de programação subjacente geralmente implementa este sincronismo de forma nativa (por exemplo, a linguagem *Java*). Porém, o uso operador *wait* é necessário neste módulo, caso ele consuma dados de outro operador de controle.

A figura 12 ilustra casos onde um operador binário (união ou junção, por exemplo), que consome dados de dois ou mais operadores, pode possuir sincronismos iguais (a e b) ou diferentes (c). Neste caso, os três PECs abaixo são equivalentes em termos algébricos.

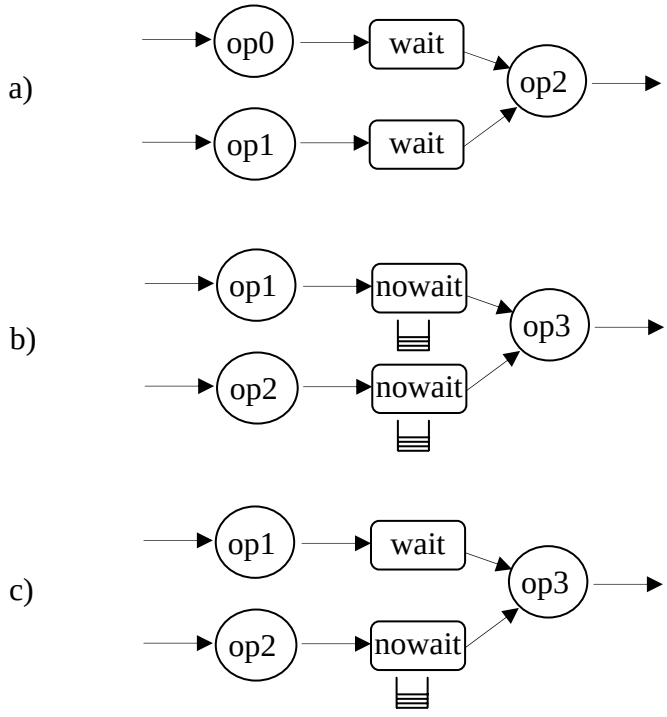

Figura 12 - Combinando Módulos de Sincronismo

Módulos de Paralelismo

O módulo de paralelismo *Inter-operator* é mostrado na Figura 13, através de um PEC de topologia *bushy*, onde um operador binário (por exemplo, união ou junção) consome, em paralelo, dados de dois (ou mais) operadores. O fato deste PEC ser de topologia *bushy* não implica, necessariamente, no uso do módulo de paralelismo. Por exemplo, os PECs da Figura 12 (a) e (c) não implementam o módulo de paralelismo. Observa-se que, embora os dois PECs acima implementem módulos de execução distintos, eles são equivalentes em termos algébricos, ou seja, o sincronismo e/ou paralelismo não interfere no resultado do processamento dos operadores.

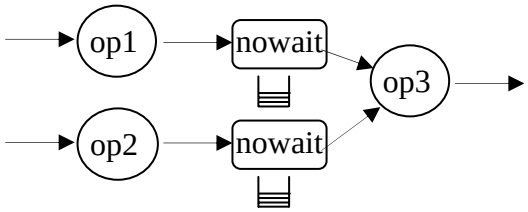

Figura 13 - Módulo de Paralelismo *Inter-operador*

O módulo de paralelismo *Intra-operator* é ilustrado na Figura 14 através de duas instâncias do operador op1, denominadas de $op1^a$ e $op1^b$, que produzem dados em paralelo para serem processados pelo operador op2. Os operadores de controle *split* e *merge* são inseridos no PEC para prover o paralelismo sobre as instâncias de op1. Além disso, esse módulo deve atender às seguintes funcionalidades:

i. o operador op2 não precisa conhecer qual das instâncias de op1 lhe fornecerá os dados, visto que estes são coletados pelo seu produtor;
ii. cada instância de op1 processa os dados como se fosse a única instância de op1;
iii. o operador *split* fornece dados para instâncias de op1 numa certa ordem de acordo com a política de particionamento de op1 tais como: *hashing*, *round-robin*, etc.
iv. o operador *split* fornece dados distintos ou idênticos para as instâncias de op1 dependendo da implementação de op1. No caso do

processamento paralelo de dados idênticos, o operador *merge* deverá unir as correspondências dos dados processados.

v. o paralelismo entre as instâncias de op1 independe do fluxo de controle dos dados. Se o fluxo for baseado em demanda, as instâncias de op1 solicitam dados ao seu produtor. Se o fluxo for baseado em dados, o produtor é quem envia dados para as instancias de op1

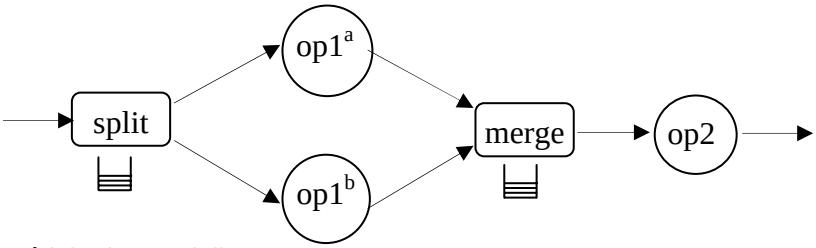

Figura 14 - Módulo de Paralelismo *Intra-operator*

Módulo de Distribuição

A Figura 15 mostra o módulo de execução distribuída entre os operadores op1 e op2, os quais estão sendo executados em *sites* distintos através das máquinas MEC1 e MEC2, respectivamente. Os operadores de controle *send* e *receive* implementam as atividades de comunicação necessárias para a transferência de dados entre processos (de *sites* distintos), o que torna a distribuição transparente aos demais operadores de um PEC. Esses operadores de controle possuem uma área temporária de memória para permitir a transferência de blocos de dados (*row blocking*) quando necessário.

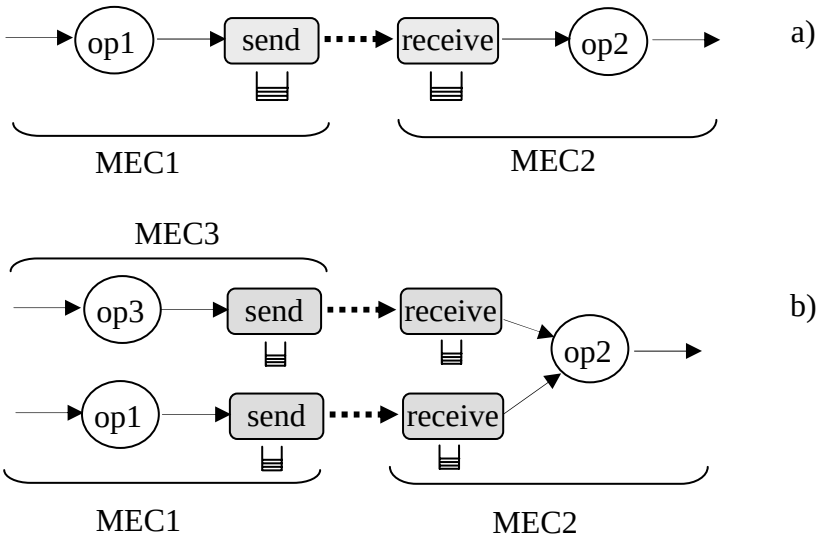

Figura 15 - Módulo de Distribuição sobre PEC linear (a) e PEC *bushy* (b)

Módulos de Fluxo de Controle

No módulo *Demand-Driven,* cada consumidor solicita dados do seu produtor através de um método *get* e no módulo *Data-Driven*, cada operador envia dados para o seu produtor através de um método *put*, como mostra a Figura 16. Para implementar estes módulos é necessário que a MEC implemente os dois métodos *get* e *put* na interface de seus operadores. A interface iterador (apresentada na seção de MECs) pode ser usada para esta finalidade, necessitando lhe adicionar o método *put*, inexistente.

Embora a maioria dos modelos de execução necessitem de apenas um desses fluxos em um mesmo PEC, é possível combiná-los através dos escalonadores de fluxo de controle, implementados pelos operadores de controle *passive* e *active,* ilustrados na Figura 17. Neste caso, o operador *active* solicita dados de seu produtor (via *get*) e os entrega ao seu consumidor (via *put*). O operador *Passive* recebe dados de seu produtor (via *put*) e que são enviados (via *get*) para o seu consumidor. Esses dois operadores de controle formam uma barreira de controle dentro de um PEC.

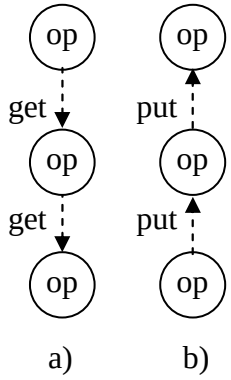

Figura 16 - Módulos de Fluxo de Controle *Demand-driven*(a) e *Data-driven*(b)

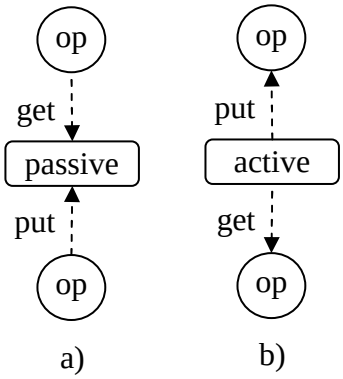

Figura 17 - Operadores de controle *passive*(a) e *active*(b)

Quando combinamos o módulo de sincronismo *Nowait* com o módulo de fluxo de controle *Data-driven*, como mostrado na Figura 18, pode ocorrer o problema conhecido como *back pressure* (Graefe, 1990), onde a taxa de recebimento de dados do *nowait* é maior do que a taxa de consumo Neste caso, considerando-se que a produção de dados de op1 independente do consumo de dados de op2 e que a taxa de produção de op1 é maior do que a taxa de consumo de op2, o operador *nowait* poderá ter sua capacidade de processamento comprometida devido ao grande volume de dados. Neste caso, o operador *nowait* deve detectar esta situação fazendo que o seu consumidor fique aguardando até que os dados do *buffer* sejam consumidos por op2 (através do uso de semáforo).

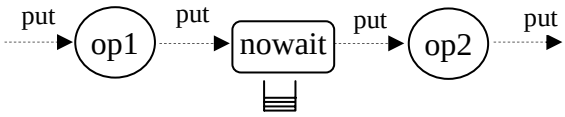

Figura 18 - O problema de *back pressure*

Módulos de Fluxo de Dados

A Figura 19 mostra os módulos de fluxos de dados. No módulo *Fixed* (Figura 19-a) o caminho percorrido pelos dados entre os operadores é sempre o mesmo, ou seja, de op1 para op2, e isto independe da existência de outros módulos entre eles (tais como sincronismo ou distribuição). No módulo *Adaptive* (Figura 19-b) a adaptabilidade ocorre devido ao operador de controle *distributor*, o qual escalona a execução dos operadores segundo um certa política.

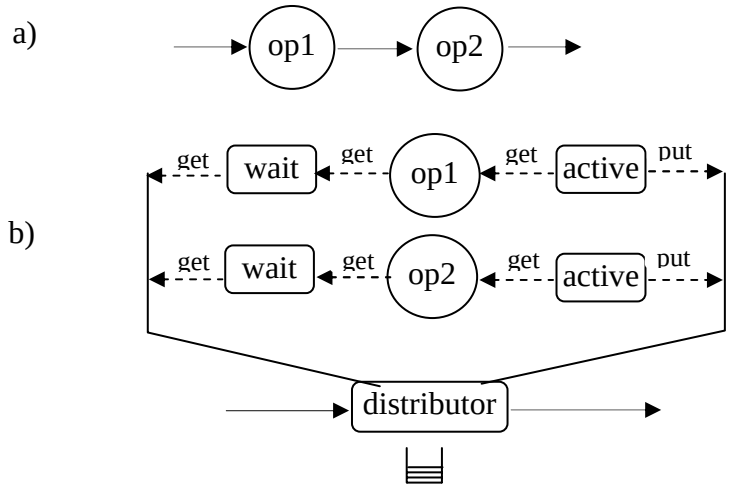

Figura 19 - Módulos de Fluxo de Dados *Fixed* (a) e *Adaptive* (b)

O módulo *Adaptive* implementa o estado *Adaptive*, da característica de execução de fluxo de dados, da seguinte forma: para cada operador adaptado (no exemplo da figura são: op1 e op2), o módulo insere os operadores *wait* e *active* (descritos em seções anteriores) para controlar o fluxo de dados daqueles operadores além de utilizar os fluxos de controle *Demand-driven* e *Data-Driven*. Cada operador *wait* instanciado consome uma instância de dados do operador *distributor* que por sua vez a fornece, a partir de sua fila de entrada. Em seguida, cada operador *wait* repassa a instância recebida, para o respectivo operador adaptado que a processa e, caso não a elimine, a repassada para o respectivo operador *active* o qual a envia de volta para a fila do *distributor* para ser processada por um outro operador ou, caso tenha completado o seu processamento por todos os operadores adaptados, para ser fornecida, quando possível, ao seu consumidor externo. De acordo com a Figura 19-b, as tuplas consumidas pelo *distributor*, podem ser processadas pelos operadores op1 e op2, segundo os seguintes caminhos:

- op1 -> op2
- op2 -> op1

À medida que aumenta a quantidade de operadores adaptados pelo módulo, aumentam também os possíveis caminhos que as tuplas podem percorrer. Por exemplo, considerando-se três operadores, teríamos os seguintes caminhos possíveis dentro do módulo:

- op1 -> op2 -> op3
- op1 -> op3 -> op2
- op2 -> op1 -> op3
- op2 -> op3 -> op1
- op3 -> op1 -> op2
- op3 -> op2 -> op1

No módulo *Adaptive* as tuplas percorrem o caminho que oferece maior rapidez de processamento, num certo intervalo de tempo, ao contrário do módulo *Fixed* onde as tuplas conhecem previamente o caminho a ser percorrido. Portanto

podemos afirmar que o primeiro módulo se baseia na otimização dinâmica e o segundo, se baseia na otimização estática, exercida pelo otimizador.

Módulo de Tempo de Resposta

A Figura 20 - mostra os módulos *First-tuple* e *Last-tuple*. A diferença entre eles encontra-se no segundo módulo, com a inclusão do operador de controle *waitall* antes do operador de resultado (raiz do PEC), promovendo a materialização dos resultados antes de serem consumidos pela aplicação usuária.

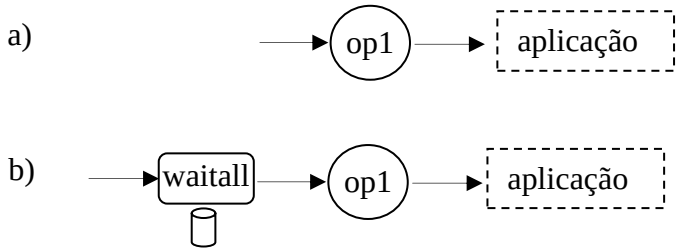

Figura 20 - Módulo de Tempo de Resposta *First-tuple* (a) e *Last-tuple* (b)

3.1.3. Modelos de Execução de Consultas

Para permitir o suporte a diferentes modelos de execução, define-se um modelo de execução como uma combinação de módulos de execução que, por sua vez, implementam as características de execução presentes em um ou mais cenários. A Tabela 4 relaciona os módulos de execução aos modelos de execução e determina, para cada modelo de execução de consultas, os respectivos módulos que devem ser implementados (ou oferecidos) pela MEC no suporte ao respectivo modelo. Observa-se que cada característica de execução apresenta diferentes módulos, em um ou mais modelos, e os modelos permitem módulos alternativos de uma mesma característica.

A seguir, são apresentados os modelos de execução tradicional, adaptável, contínuo e baseado em streams. Em cada modelo, é exemplificado a combinação de módulos em um PEC. A escolha dos exemplos visa demonstrar a flexibilidade necessária para a execução de PECs que suportem diferentes modelos de execução.

Características de Execução	*Módulos*	Modelos			
		Tradicional	Adaptável	Contínuo	Streams
Sincronismo	*Wait*	*	*		X
	Wait-all		X		X
	Nowait	X	X	*	*
Distribuição	*Remote*	X	X	X	X
	Local	*	*	X	X
Paralelismo	*Inter*	*	*	X	X
	Intra	X	X	X	*
Fluxo de Dados	*Fixed*	*		X	*
	Adaptive		*	*	X
Fluxo de Controle	*Demand-driven*	*	*	X	X
	Data-driven		X	*	*
Tempo de Resposta	*First-tuple*	*	*	*	*
	Last-tuple		X		X

Legenda: (*)=configuração mais relevante, (X)=configuração possível

Tabela 4 - Classificação dos Modelos de Execução de Consultas

Modelo de Execução Tradicional

Nesta seção, exemplifica-se a execução de um PEC segundo o modelo de execução tradicional. A Figura 21 mostra o PEC do Exemplo 1 que combina os módulos *Fixed*, *First-tuple*, *Demand-Driven* e *Wait*. As setas sólidas da figura indicam o sentido do fluxo dos dados. As fontes de dados foram omitidas por simplicidade. A Figura 22 ilustra a execução deste PEC onde cada operador consome uma instância de cada vez de seu produtor. As setas tracejadas da figura indicam o sentido do fluxo de controle. Como conseqüência deste processo de execução, os quatro operadores, op1, op2, op3 e op4 processam tuplas de maneira síncrona. À medida que a aplicação solicita novas tuplas, estas são produzidas por op4. O caminho percorrido pelas tuplas é fixo, isto é: op1 -> op2 -> op3 -> op4.

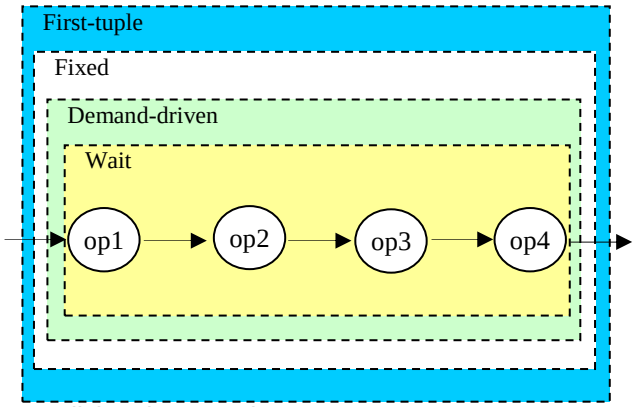

Figura 21 - Modelo Tradicional: Exemplo 1

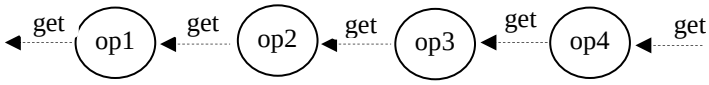

Figura 22 - Execução do Exemplo 1

A Figura 23 mostra o PEC do Exemplo 2 que combina os módulos *Fixed*, *First-tuple*, *Demand-Driven*, *Wait e Intra-operator*. A Figura 24 ilustra a execução deste PEC. A única diferença para o exemplo 1 consiste no paralelismo dos operadores op1, op2 e op3 que antecipam a chegada de tuplas para op4.

Neste ponto, é importante observar que o exemplo 2 é equivalente ao exemplo 1, em termos algébricos, embora apresentem variações nos seus modelos de execução.

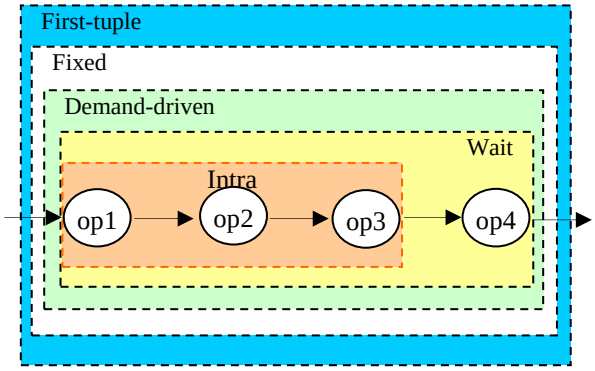

Figura 23 - Modelo Tradicional: Exemplo 2

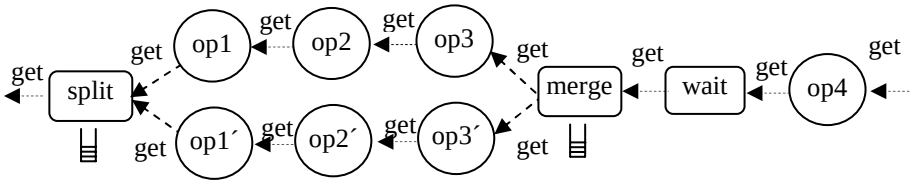

Figura 24 - Execução do Exemplo 2

A Figura 25 mostra o PEC do Exemplo 3 que combina os módulos *Fixed*, *First-tuple*, *Demand-Driven*, *Wait e Inte-operator*. A Figura 26 ilustra a execução deste PEC de forma paralela (paralelismo inter-operador vertical e horizontal). Considerando-se que op3 é um operador de junção não simétrico (por exemplo, *NeestedLoop*), as tuplas recebidas de op2 podem ser consumidas, por ele, N vezes. Para se evitar este custo na execução, deve-se configurar o operador nowait para guardar (em memória e/ou disco) as tuplas recebidas de op2 na primeira vez que

são consumidas. Neste caso, é importante notar que a execução do operador *nowait* não o caracteriza como sendo um *waitall*, pois ele não necessita consumir todas as tuplas de op2 para enviar uma instância a op3.

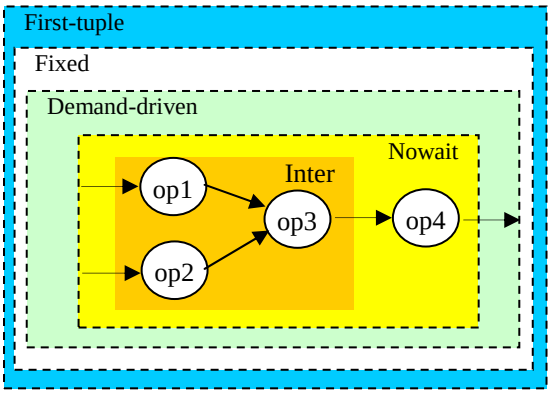

Figura 25 - Modelo Tradicional: Exemplo 3

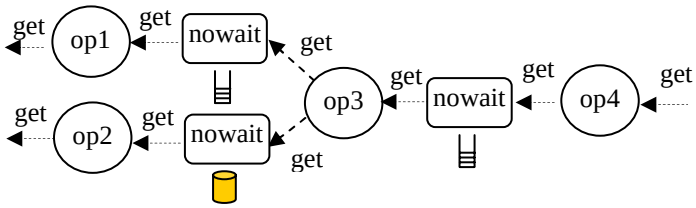

Figura 26 - Execução do Exemplo 3

Modelo de Execução Contínuo

Nesta seção, exemplifica-se a execução de um PEC segundo o modelo de execução contínuo. A Figura 27 mostra o PEC do Exemplo 4 que combina os módulos *Fixed*, *First-tuple*, *Data-driven*, *Nowait e Inter-operator*. A Figura 28 ilustra a execução deste PEC que contém os mesmos operadores do Exemplo 3, porém, com o fluxo de controle direcionado a dados (via *put*), ou seja, com a chegada de novos dados, novos resultados são produzidos.

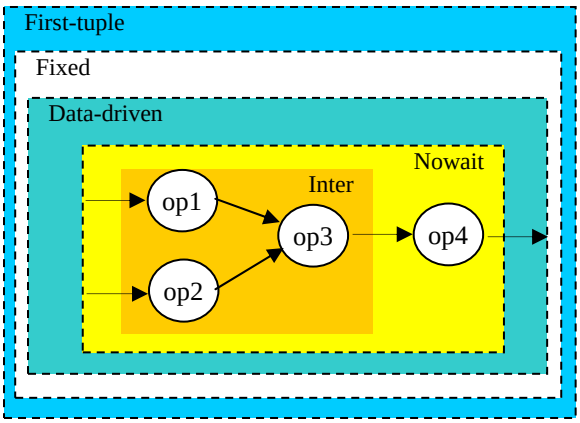

Figura 27 - Modelo Contínuo: Exemplo 4

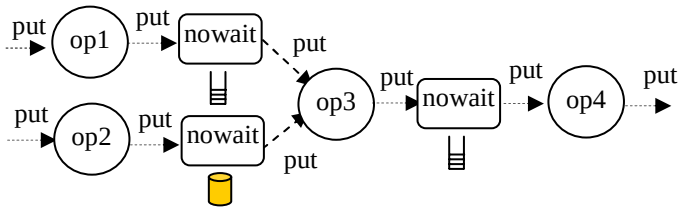

Figura 28 - Execução do Exemplo 4

Modelo de Execução Adaptável

Nesta seção, exemplifica-se a execução de um PEC segundo o modelo de execução adaptável. A Figura 29 mostra o PEC do Exemplo 5 que combina os módulos *Adaptive*, *Fixed*, *First-tuple*, *Demand-Driven* e *Wait*. A Figura 30 ilustra a execução deste PEC que ocorre da seguinte forma: o operador op4 solicita uma instância ao *distributor* que consome tuplas de op1 e as envia para a execução adaptável dos operadores op2 e op3, até que um destes produza uma instância para op4. Como conseqüência deste processo de execução, os quatro operadores, op1, op2, op3 e op4 processam tuplas de maneira síncrona. À medida que a aplicação solicita novas tuplas, estas são produzidas por op4. Os caminhos percorridos pelas tuplas são fixos de op1 para op4, mas variam entre op2 e op3 de acordo com a adaptabilidade destes dois últimos, como indicado a seguir:

- op1 -> op2 -> op3 -> op4
- op1 -> op3 -> op2 -> op4

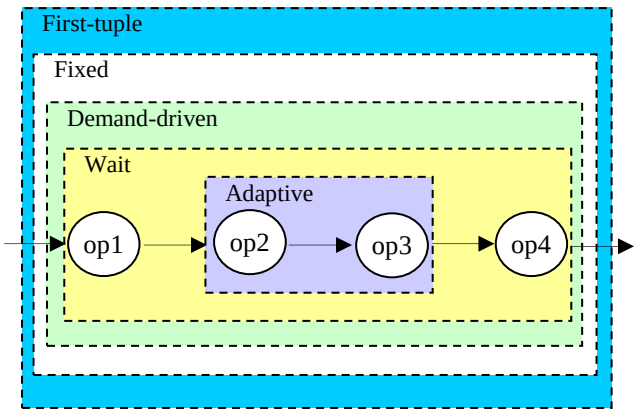

Figura 29 - Modelo Adaptável: Exemplo 5

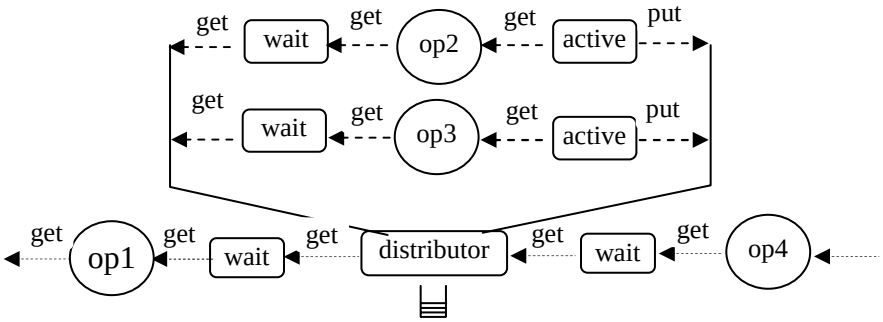

Figura 30 - Execução do Exemplo 5

A Figura 31 mostra um exemplo de PEC que combina os módulos *Adaptive*, *Nowait*, *Demand-Driven*,*Fixed*, *e Last-tuple*. A diferença deste exemplo para o anterior, é a o módulo de sincronismo Nowait. A Figura 32 ilustra a execução deste PEC. Como resultado, os quatro operadores, op1, op2, op3 e op4 processam tuplas de maneira assíncrona. A adaptabilidade, bem como a produção de tuplas do resultado, ocorre como descrito no modelo anterior

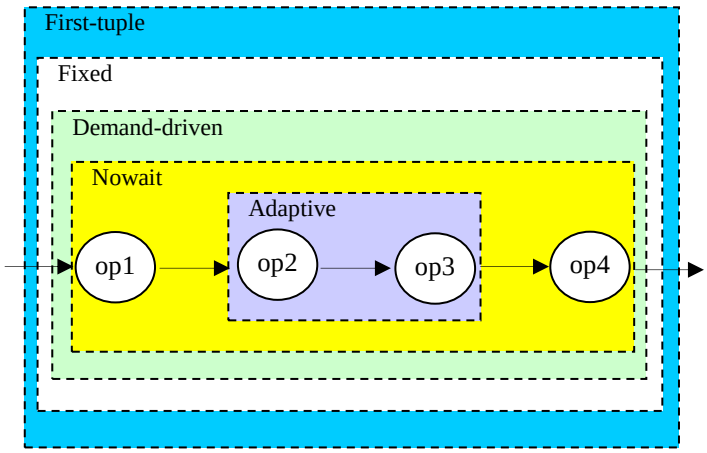

Figura 31 - Modelo Adaptável: Exemplo 6

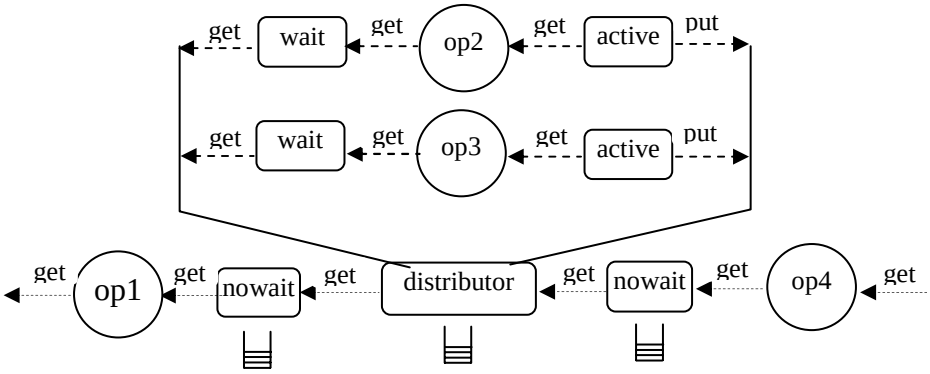

Figura 32 - Execução do Exemplo 6

Modelo de Execução baseado em Streams

Nesta seção, exemplifica-se a execução de um PEC segundo o modelo de execução adaptável. A Figura 33 mostra o PEC do Exemplo 7 que combina os módulos *Fixed*, *First-tuple*, *Demand-driven*, *Data-driven e Inter-operator*.

A Figura 34 ilustra a execução deste PEC. Os operadores op1 e op2 consomem tuplas enviadas pelas respectivas *streams* que são confrontadas em op3. Por outro lado, a aplicação deseja consumir tuplas sob demanda, ou seja, não necessita processar dados de forma contínua. Neste caso, há necessidade da combinação dos módulos *Demand-driven* e *Data-driven* e isso produz uma barreira de controle sobre o operador op3 (escalonamento *get* vs. *put*) que deve ser eliminada com a inclusão do operador *passive*.

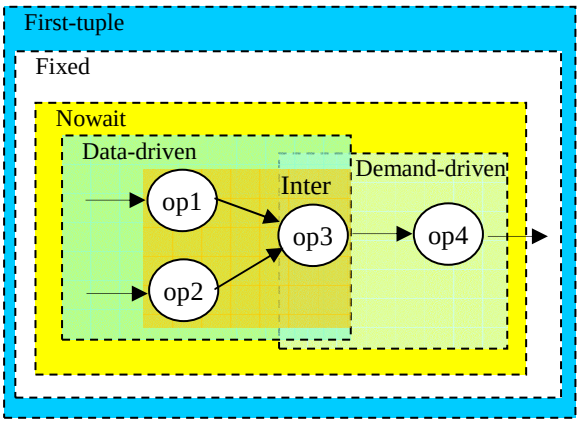

Figura 33 - Modelo baseado em Streams: Exemplo 7

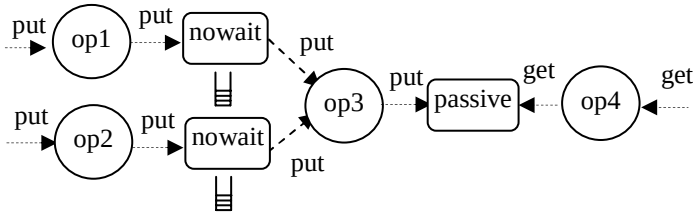

Figura 34 - Execução do Exemplo 7

Outros Modelos de Execução

Esta seção e as anteriores exemplificaram os modelos de execução de PECs dos cenários estudados neste trabalho que demonstram o potencial para a geração de novos modelos de execução tal como o modelo do PEC da Figura 35 o qual combina os módulos *Adaptive*, *Wait, Data-Driven, Fixed, e Last-tuple*.

A Figura 36 ilustra a execução deste PEC que ocorre da seguinte forma: as tuplas produzidas por op1 são repassadas para op2 (via *wait*), e assim sucessivamente, pelos demais os operadores do modulo *Fixed* (sempre utilizando o método *put)*, até chegar em *waitall* que as materializa. Quando a produção de op1 terminar, op4 consumirá as tuplas armazenadas por *waitall*. O módulo *Adaptive* procederá da mesma maneira dos exemplos anteriores.

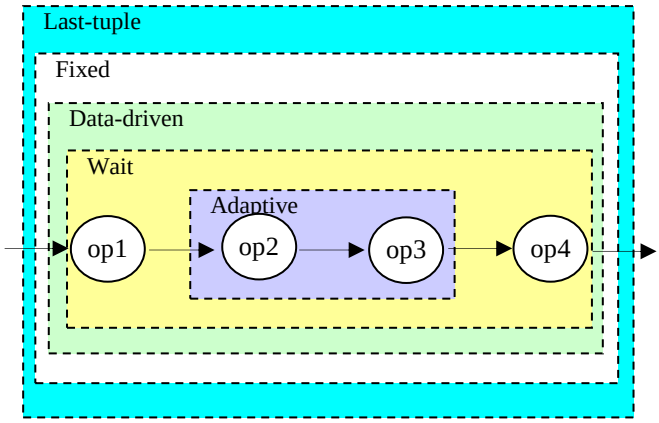

Figura 35 - Novo Modelo: Exemplo 8

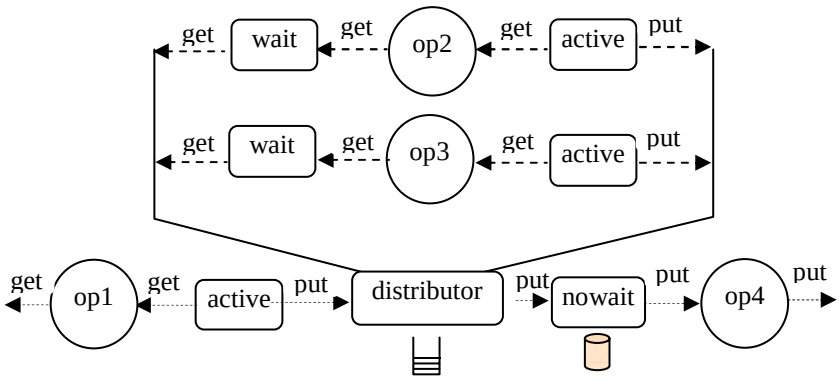

Figura 36 - Execução do Exemplo 8

3.2. Suporte a Diferentes Modelos de Dados

Tradicionalmente, uma MEC é construída para suportar um modelo de dados específico, presente num certo cenário. Neste contexto, a MEC utiliza uma tupla de dados que, durante a execução de um PEC, trafegará entre os operadores da MEC, sendo criada pelos operadores mais próximos das fontes de dados e sendo processada pelos demais operadores do PEC até atingir o operador mais próximo do resultado da consulta, podendo ainda ser descartada por um dos operadores. Cada tupla de dados possui uma estrutura de dados definida a partir do modelo de dados da MEC e que é conhecida por todos os operadores que a manipulam. Por exemplo, a estrutura de uma tupla no modelo relacional pode ser modelada como uma lista de atributos contendo cada um, um nome, um tipo e um valor.

A abordagem proposta nesta tese permite construir uma MEC que combine tuplas de dados em diferentes modelos (p.ex: no modelo Relacional e XML) desde que processadas pelos seus respectivos operadores algébricos através de fragmentos de PEC distintos, um para cada modelo de dados – o uso combinado dos modelos XML e Relacional é particularmente útil em aplicações de integração de dados na *web*, tais como, *Agora* (Manolescu et al., 2000) e *Silkroute* (Fernandez et al., 2002). Desta forma poderíamos ter dentro de um PEC um fragmento de execução para processar tuplas de dados relacionais e um fragmento para processar tuplas de dados XML. Além disso, faz necessário o uso de operadores inter-algébricos para conversão de tuplas de dados entre os modelos. Estes operadores são inseridos no PEC num ponto[3] de conversão entre os dois fragmentos.

Portanto, para permitir o suporte a mais de um modelo de dados, uma MEC deve implementar, além dos operadores algébricos de cada modelo, operadores de conversão (inter-algébricos) entre os modelos.

[3] Neste trabalho, não será discutido como o otimizador determinará o ponto de conversão entre modelos.

3.3. Ortogonalidade entre Modelos de Execução e de Dados

Considerando-se que: (i) os operadores algébricos implementam os algoritmos de manipulação de dados baseados na álgebra de um modelo de dados, (ii) os operadores de controle são inseridos entre os operadores algébricos de forma a implementar um determinado módulo de execução (como parte de um modelo de execução) e (iii) a independência entre operadores algébricos e operadores de controle, que permite modificar os primeiros sem afetar os demais (e vice-versa); obtemos a ortogonalidade entre os modelos de execução e de dados, permitindo a combinação entre diferentes modelos de dados com diferentes modelos de execução. Um exemplo disso pode ser observado na Seção 3.1.3, onde o exemplo 2 é equivalente ao exemplo 1, em termos algébricos, embora apresentem diferentes modelos de execução.

3.4. Síntese do Capítulo

Neste capítulo, foi apresentada a solução proposta desta tese para o suporte a diferentes modelos de execução e de dados, de forma ortogonal. O suporte a diferentes modelos de execução dá-se através da combinação de módulos de execução, os quais inserem operadores de controle apropriados dentro do PEC para implementar uma determinada característica de execução presente num certo cenário. O suporte a diferentes modelos de dados dá-se através da combinação de diferentes operadores algébricos dentro de um PEC através do uso de operadores inter-algébricos. A independência entre operadores algébricos e de controle, e as associações <operadores algébricos, modelo de dados> e <operadores de controle, modelo de execução>, se refletem na ortogonalidade entre os modelos de dados e de execução.