4
Middleware SCS-Collective

O SCS-Collective é uma extensão do *middleware* de componentes SCS que tem como objetivo oferecer mecanismos que melhorem o suporte ao desenvolvimento de aplicações paralelas baseadas em componentes. O SCS-Collective implementa o conceito de *Interfaces Coletivas* descrito no Capítulo 3. Essas interfaces são um caminho expressivo para a implementação de comportamento e padrões de programação paralela em aplicações componentizadas, permitindo de forma simples o uso de diversas arquiteturas de paralelização e políticas de redistribuição de dados.

Este capítulo abordará os principais aspectos relativos a arquitetura e implementação do SCS-Collective, descrevendo as decisões e padrões de programação que foram utilizados.

4.1
Arquitetura

Um componente de software implementado no modelo SCS, como descrito no Capítulo 3, deve implementar três facetas básicas (*IComponent*, *IReceptacles* e *IMetaInterface*) e mais as facetas definidas pelo programador. O SCS-Collective, estendeu o SCS com dois novos tipos de conectores (ver Figura 4.1):

1. **Conector *GatherFacet***: Foi idealizado como uma extensão do conceito de *facetas* do SCS para suportar a semântica da interface *Gathercast* do GCM. Dessa forma uma *GatherFacet* agrupa um conjunto de chamadas em uma única chamada e realiza as devidas operações de combinação e distribuição de dados. De forma análoga à interface *Gathercast*, esse conector possui semântica de uma barreira, pois somente invoca o método correspondente na interface interna do conector depois que todos os clientes conectados invocaram o método correspondente na interface externa. Os clientes permanecem bloqueados esperando a conclusão da chamada. Também é possivel a especificação de um tempo máximo de espera (*timeout*) para evitar problemas como o *deadlock*.

2. **Conector *MulticastReceptacle***: Foi definido como sendo uma especialização de um *receptáculo* SCS que implementa o comportamento da interface *Multicast* do GCM, possibilitando que uma única invocação seja transformada em um conjunto de invocações paralelas. Estratégias de redistribuição de dados como o *broadcast* ou *scattering* precisam ser definidas. Este conector possui duas *sub-interfaces*: uma interna, que permite o acesso do componente cliente às facetas conectadas no receptáculo, e uma externa, que define o tipo de faceta que pode ser conectada neste.

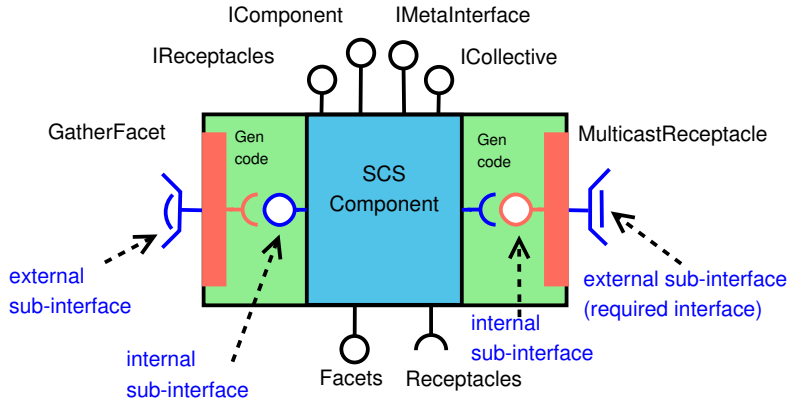

Figura 4.1: Modelo de componentes do SCS-Collective

Como descrito no Capitulo 3, as ICs possuem duas *sub-interfaces*, uma interna e uma externa, sendo essencialmente entre essas *sub-interfaces* que ocorre a redistribuição de dados. O ProActive faz uso da API de reflexão computacional da linguagem Java para configurar as políticas de redistribuição de dados.

Como a implementação do SCS-Collective utilizou a versão C++ do modelo de componentes SCS, e a mesma não dispõe de um mecanismo padronizado ou minimamente funcional para reflexão, é proposto um mecanismo de geração de código a partir das anotações em interfaces CORBA IDL, ou seja, além da geração convencional dos stubs pela ferramenta *idl* distribuída com os ORBs[1], tem-se também a geração do código de redistribuição de dados que faz a ligação entre as *sub-interfaces* externa e interna de cada conector paralelo, como pode-se ver nas regiões denominadas *Gen code* na Figura 4.1.

O conector *GatherFacet* exige que no momento que um *receptáculo* ou *MulticastReceptacle* se conecte nele, este realize uma chamada padrão para configuração dessa nova conexão. Nessa configuração é incrementado o número de

[1]ORBs (de *Object Request Broker*) são implementações do padrão de comunicação entre objetos distribuídos CORBA.

receptáculos conectados nessa *GatherFacet*, e os semáforos utilizados para a sincronização são reconfigurados. Para evitar a inserção de um método não funcional (método de configuração de conexão gather) na interface da *GatherFacet*, foi decidido pela criação da nova *faceta* básica chamada *ICollective* onde essas configurações pudessem ser agregadas (ver Figura 4.1).

Para oferecer suporte às *sub-interfaces* do *MulticastReceptacle*, foram realizadas algumas extensões nas estruturas que definem um receptáculo SCS, tais como a adição dos campos **`external_interface_name`** e **`internal_interface_name`**, que explicitam respectivamente o tipo de interface que pode ser conectada neste receptáculo *multicast* (interface externa) e o tipo da interface que será usado pelo componente para acessar as conexões no seu receptáculo *multicast* (interface interna).

Código 4.1: Estrutura que define um *MulticastReceptacle*

```
1  struct ReceptacleDescription {
2
3    /** O nome identificador. **/
4    string name;
5    /** O nome da interface que se conecta ao receptáculo.   **/
6    string interface_name;
7    /** Indica se o receptáculo suporta múltiplas facetas conectadas. **/
8    boolean is_multiplex;
9    /** As descrições das conexões que foram estabelecidas.  **/
10   ConnectionDescriptions connections;
11
12   /** ========== MulticastReceptacle fields ========== **/
13   /**  Nome da interface que se conecta ao receptaculo     **/
14   string external_interface_name;
15   /** Nome da interface do objeto que é fornecido no       *
16    *  momento que o receptáculo é usado                    **/
17   string internal_interface_name;
18 };
19
20 /** O conjunto de descritores de receptáculos.            **/
21 typedef sequence<ReceptacleDescription> ReceptacleDescriptions;
```

Como pode ser visto no Código 4.1, o *MulticastReceptacle* foi idealizado como uma extensão do receptáculo do SCS (linhas 12-17) e, para isso, na implementação foi utilizada a técnica de polimorfismo para a instanciação do receptáculo *multicast*. O processo de instanciação foi estruturado seguindo o padrão de projeto *Strategy* [34]. Como se pode verificar na Figura 4.2, a criação do *MulticastReceptacle* é realizado pela classe *ComponentBuilder* (representado no padrão por *Context*) que possui o conhecimento para a instanciação das estratégias concretas *Receptacle* (estratégia padrão) e *MulticastReceptacle* (representados no padrão por *ConcreteStrategy A e B*).

O conector *GatherFacet* possui uma implementação semelhante a de uma faceta convencional do SCS, com a diferença que o mecanismo de geração de código cria a implementação da *sub-interface* externa, sendo papel do

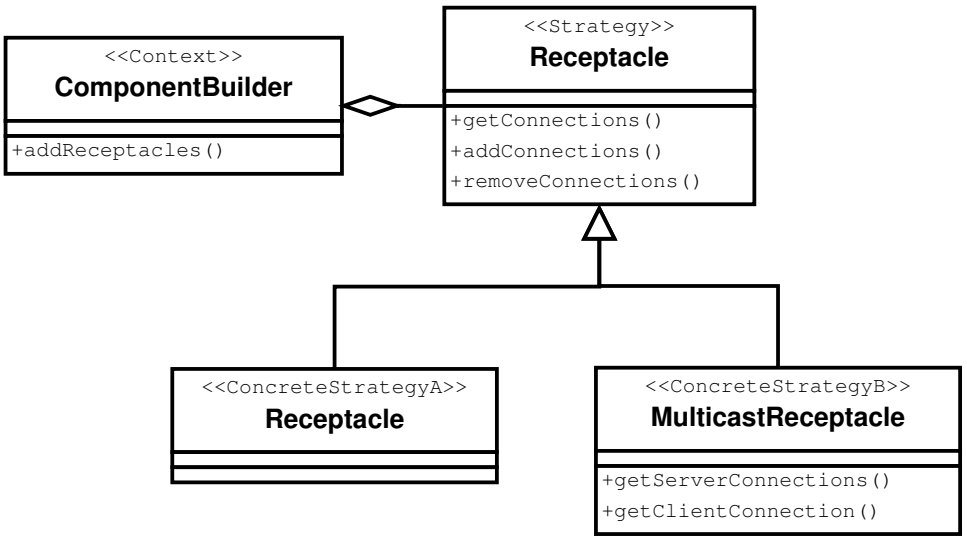

Figura 4.2: Padrão *Strategy* na instanciação do conector *MulticastReceptacle*

desenvolvedor completar a implementação do comportamento do conector no esqueleto de implementação gerado para a *sub-interface* interna.

As chamadas na *sub-interface* externa de uma *GatherFacet* são sincronizadas através de semáforos, que somente permitem a invocação da implementação do serviço (*sub-interface* interna) quando todos os clientes conectados realizaram a chamada.

4.1.1 Conectores Coletivos

Visando um entendimento geral da arquitetura de funcionamento do SCS-Collective, estão descritos nas Figuras 4.3 e 4.4 os principais elementos presentes na arquitetura de cada conector. Estão descritas as iterações entre esses elementos para que uma chamada se complete, destacando pontos como o *marshaling* e *unmarshaling* de dados, barreiras para sincronização, objetos cuja implementação é gerada pelo utilitário de geração e outros detalhes.

O uso do conector *MulticastReceptacle* se dá em três partes principais:

1. criação das conexões com as *facetas* que o *MulticastReceptacle* aceita (*sub-interface* externa),
2. realização da configuração do *MulticastReceptacle*, e
3. invocação através da *sub-interface* interna (local), que gerará as invocações paralelas para as *sub-interfaces* externas conectadas.

Em relação às conexões, a Figura 4.3 mostra os componentes conectados *PI_DartboardWorkerImpl* e a especialização do receptáculo padrão *IReceptacles*

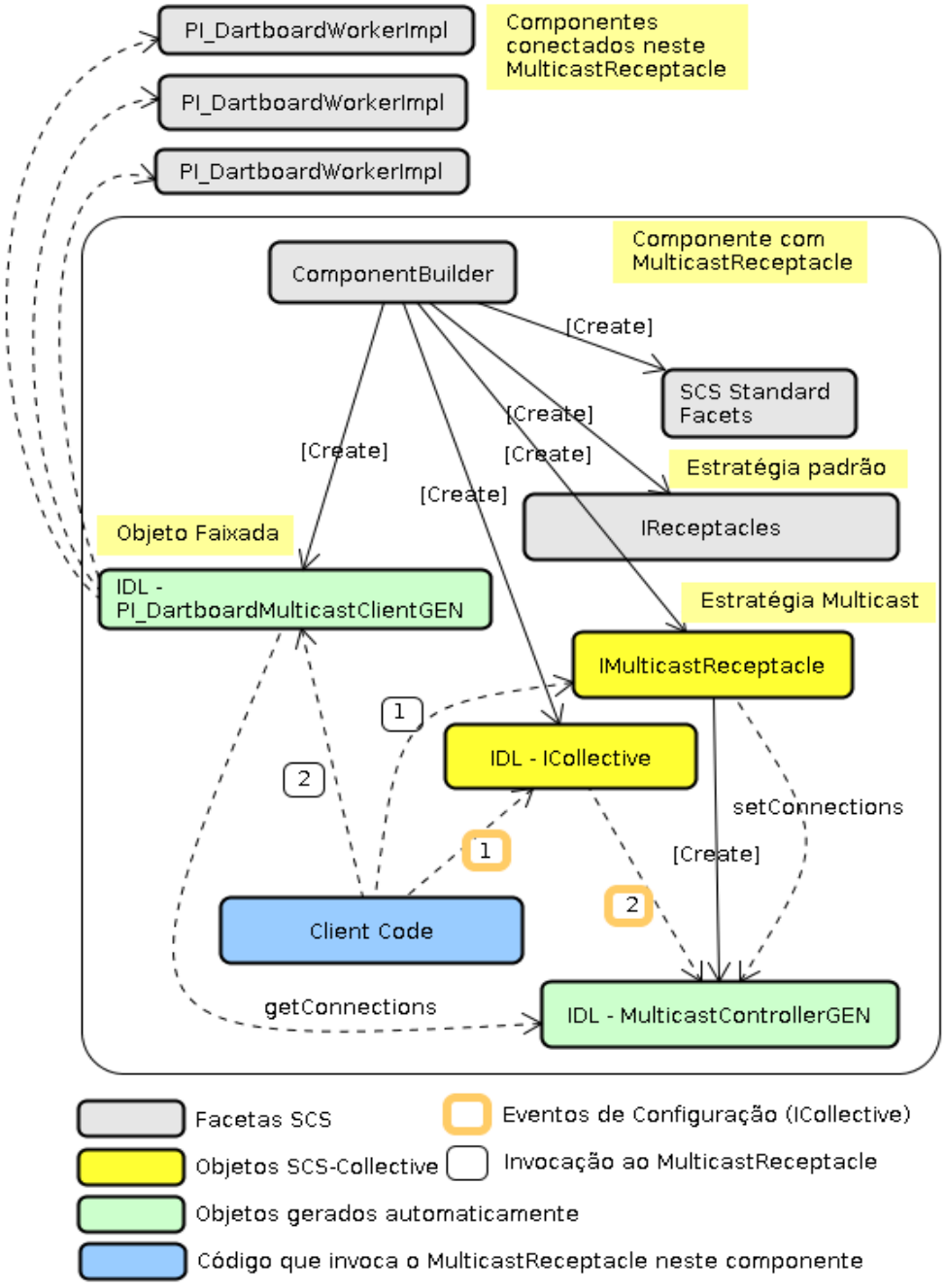

Figura 4.3: Interações para um componente com *MulticastReceptacle*

em *IMulticastReceptacle* (implementação do receptáculo com estratégia *Multicast*) que realiza as configurações de conexão de facetas através do método *setConnections* no controlador *MulticastControllerGEN*.

A configuração do *MulticastReceptacle* também é mostrada na Figura 4.3, onde o código executando no componente (*Client Code* na Figura 4.3) realiza uma chamada à *Faceta ICollective*, esta realiza a configuração final no controlador *MuticastControllerGEN*.

A invocação procede da forma convencional do SCS para invocações

a facetas conectadas em um receptáculo, onde o *Client Code* (código em execução de função de alguma faceta do componente) obtém as conexões de *IReceptacle*, nesse caso, uma única conexão para o objeto de faixada *PI_DartboardMulticastClientGEN*, o qual mantém as referências para as interfaces conectadas e realiza as invocações paralelas.

Resumindo, os elementos ilustrados na Figura 4.3, que são importantes para o funcionamento do *MulticastReceptacle* possuem os seguintes papéis:

- **ICollective**: Configura o receptáculo *multicast* através do controlador.
- **IMulticastReceptacle**: O receptáculo *multicast* em si (especialização do receptáculo padrão do SCS).
- **MulticastControllerGEN**: Controlador usado pelo *ICollective* para configurar o receptáculo *multicast*.
- **PI_DartboardMulticastClientGEN**: Objeto que implementa a *sub-interface* interna responsável por manter as conexões com as facetas remotas *PI_DartboardWorkerImpl* (*sub-interface* externa).
- **Client Code**: Código de alguma faceta do componente que usa o receptáculo *multicast*.
- **PI_DartboardWorkerImpl**: Faceta requerida pelo receptáculo *multicast* (*sub-interface* externa).
- **IReceptacles**: Gerencia os receptáculos no SCS.
- **ComponentBuilder**: Instancia todas as facetas presentes na descrição do componente, incluindo as facetas providas pelo SCS.
- **SCS Standard Facets**: Representa as facetas padrão do SCS como a *IComponent* e *IMetaInterface*.

Um facilitador para a realização dessas invocações paralelas foi a utilização da API de invocação dinâmica do CORBA (*Dynamic Invocation Interface* - DII), que permite a montagem de requisições para objetos remotos. A requisição é construída utilizando-se informações como nome da interface, nome do método e informação sobre os parâmetros e valores de retorno. Uma vez que a invocação tenha sido devidamente configurada, podem ser realizadas invocações síncronas (**invoke**), assíncronas sem retorno (**send_oneway**) e assíncronas com retorno (**send_deferred**). A implementação do *MulticastReceptacle* fez uso dessa API do CORBA para a construção das requisições e, através da rotina **invoke** e da biblioteca de suporte a *threads pThreads*, implementou o suporte às invocações paralelas.

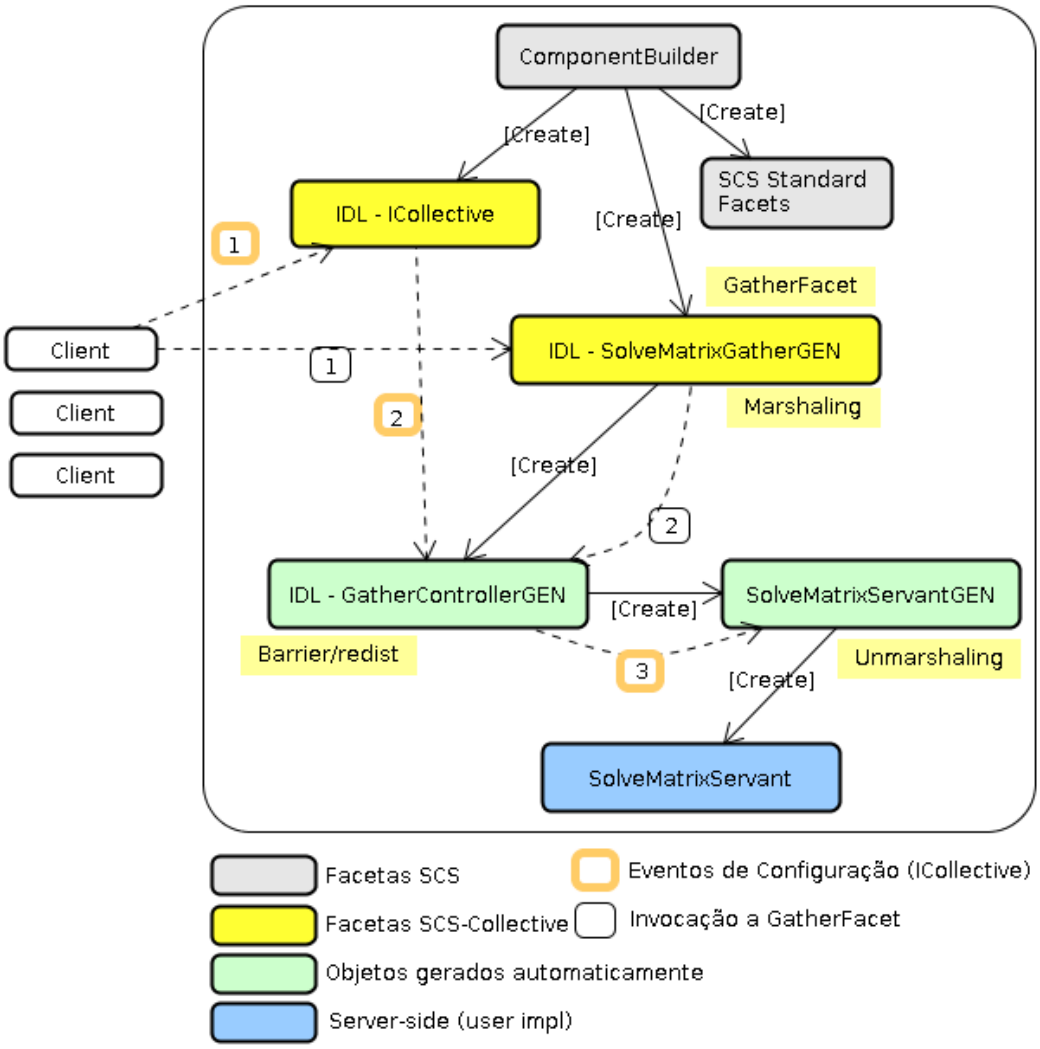

Figura 4.4: Interações para um componente com conector *GatherFacet*

O uso do conector *GatherFacet* segue o mesmo esquema do conector *MulticastReceptacle*, com algumas modificações:

1. a *GatherFacet* (*sub-interface* externa) é conectada a vários receptáculos de outros componentes,

2. cada receptáculo no qual a *GatherFacet* foi conectada precisa configurá-la pela faceta *ICollective*. Um dos itens dessa configuração é o incremento do contador de clientes conectados.

3. agora todos os receptáculos podem invocar a *GatherFacet*.

Como se pode observar na Figura 4.4, os componentes *Client* conectados a *gatherfacet* precisam, antes de realizar a invocação a algum método da *gatherfacet SolveMatrixGather*, fazer a configuração da mesma utilizando a *faceta ICollective*.

Durante a invocação, a *gatherfacet SolveMatrixGather* (*sub-interface* externa) mostrada na Figura 4.4 realiza o agrupamento dos parâmetros e repassa a chamada para *GatherControllerGEN* que bloqueia (não retorna) as primeiras invocações até que todos os componentes conectados tenham realizado a invocação. Uma vez que a sincronização tenha sido feita, uma invocação unificada (que leva os parâmetros das invocações realizadas pelos componentes conectados) é iniciada para o objeto *SolvematrixServantGEN* (implementa a *sub-interface* interna), que, finalmente, executa a implementação funcional da *GatherFacet* e retorna.

Resumindo, o papel de cada elemento necessário para o funcionamento da *GatherFacet* ilustrado na Figura 4.4 é o seguinte:

- **`ICollective`**: Permite a configuração da *GatherFacet*. Essa configuração é realizado através do controlador da *GatherFacet*.
- **`SolveMatrixGatherGEN`**: Objeto que implementa a *sub-interface* externa da *GatherFacet*. Faceta exposta, que recebe as requisições remotas, realiza o agrupamento dos dados e as repassa para o controlador, onde é feita a sincronização das chamadas.
- **`GatherControllerGEN`**: Controlador da *GatherFacet*. Bloqueia as invocações dos clientes até que todos tenham invocado, iniciando, assim, a invocação para o objeto *SolveMatrixServantGEN* que repassa para o objeto que implementa a *sub-interface* interna da *GatherFacet*.
- **`SolveMatrixServantGEN`**: Objeto que realiza o *unmarshaling* ou desagrupa os dados agrupados em *SolveMatrixGatherGEN*, colocando-os no formato para a execução no objeto *SolveMatrixServant* que implementa a *sub-interface* interna.
- **`SolveMatrixServant`**: Objeto que implementa a *sub-interface* interna da *GatherFacet* e possui a implementação funcional da aplicação.
- **`ComponentBuilder`**: Instancia todas as facetas presentes na descrição do componente, incluindo as facetas providas pelo SCS.
- **`SCS Standard Facets`**: Representa as facetas padrão do SCS como a *IComponent* e *IMetaInterface*.

4.2 Políticas de Redistribuição de Dados Suportadas

A configuração das políticas de redistribuição de dados é feita através de anotações na interface CORBA IDL que descreve o conector. Como é ilustrado no Código 4.2, foi criado um conjunto de anotações para identificação dos conectores paralelos, suas respectivas *sub-interfaces* interna e externa e políticas de redistribuição suportadas.

Código 4.2: Exemplo de definição dos conectores e políticas de redistribuição

```
1  /**
2   * JacobiSolver.idl
3   * Autor: Paulo da Silva
4   **/
5
6  module scs {
7    module demos {
8      module JacobiSolver {
9
10       typedef sequence<double> DoubleSeq;
11
12       //Defining GatherFacet and MulticastReceptacle
13       //=============================================
14
15       /**
16        * @CollectiveFacetConfig (
17        *    @GatherMode ( GathercastMode.INTERNAL )
18        *    @MulticastMode ( MulticastMode.INTERNAL ))
19        **/
20       interface SolveJacobiServant
21       {
22         /** //MulticastReceptacle data distribution configuration
23          * @MethodDispatchConfig (
24          *    @ParamDispatch ( Strategy.SCATTER ),
25          *    @ResultGathering ( Strategy.GATHER ) )
26          **/
27         DoubleSeq exchangeBorders ( in DoubleSeq borders );
28       };
29
30       /**
31        * @CollectiveFacetConfig (
32        *    @GatherMode ( GathercastMode.EXTERNAL ),
33        *    @MulticastMode ( MulticastMode.EXTERNAL ) )
34        **/
35       interface SolveJacobiReceiver
36       {
37         /** //GatherFacet data distribuition configuration
38          * @MethodDispatchConfig (
39          *    @ParamGathering ( Strategy.GATHER ),
40          *    @ResultDispatch ( Strategy.SCATTER ) )
41          **/
42         double exchangeBorder ( in double border ) ;
43
44       };
45     };
46   };
47 };
```

A configuração dos conectores paralelos é realizada com a anotação **`@CollectiveFacetConfig`**, onde é possível utilizar as seguintes opções:

– **`@MulticastMode ( MulticastMode.INTERNAL )`**: Configura essa interface como a *sub-interface* interna do *MulticastReceptacle*. Sua

implementação seria gerada de forma automática (no estado atual da implementação essa geração de código ainda não é automática).

- **`@MulticastMode ( MulticastMode.EXTERNAL )`**: Configura essa interface como a *sub-interface* externa do *MulticastReceptacle*.
- **`@GatherMode ( GathercastMode.EXTERNAL )`**: Configura essa interface como a *sub-interface* externa da *GatherFacet*. Será a interface exposta, ou, que poderá ser conectada em quaisquer receptáculos de outros componentes SCS. Sua implementação seria gerada de forma automática (essa geração de código ainda é feita de forma manual como já mencionado).
- **`@GatherMode ( GathercastMode.INTERNAL )`**: Configura essa interface como a *sub-interface* interna da *GatherFacet*. A implementação dessa interface é o código funcional da aplicação.

As políticas de redistribuição de dados (parâmetros e resultados) dos métodos são definidas pela anotação **`@MethodDispatchConfig`**, e podem assumir os seguintes valores:

- Para o conector *MulticastReceptacle*:
  - **`@ParamDispatch ( Strategy.BROADCAST )`**: Se aplica na configuração do envio de parâmetros de um método, onde o primeiro parâmetro é repetido em todas as chamadas paralelas geradas. No estado atual da implementação, as políticas de redistribuição de dados são aplicadas somente no primeiro parâmetros das funções.
  - **`@ParamDispatch ( Strategy.SCATTER )`**: Mesmo esquema da anotação anterior, todavia o primeiro parâmetro (normalmente um vetor/sequencia) é dividido entre as chamadas paralelas geradas. Se o tamanho do vetor não é múltiplo do número de chamadas, o número de elementos referente a resto da divisão é dividido entre as invocações.
  - **`@ResultGathering ( Strategy.GATHER )`**: Realiza uma junção do retorno de invocações paralelas geradas por um *MulticastReceptacle*. Se a sequencia retornada em cada uma de 8 chamadas paralelas possuem um tamanho X, então a sequencia de retorno final possuirá tamanho $8 * X$. A ordem de junção é determinada pela ordem dos objetos de requisições no vetor que os armazena, pois os resultados são capturados percorrendo esse vetor, onde é verificado se cada requisição já possui um resultado válido retornado. Se alguma requisição não retornou o procedimento é interrompido até que o resultado esteja disponível.

  - **`@ResultGathering ( Strategy.REDUCE )`**: Aplica uma determinada operação de redução nos valores de retorno de invocações paralelas geradas. A implementação atual do SCS-Collective possui a soma como operação de redução padrão.

- Para o conector *GatherFacet*:

  - **`@ParamGathering ( Strategy.GATHER )`**: É utilizado para agregação de parâmetros de invocações a *GatherFacet*. Nesse caso é realizada uma junção dos vetores de cada invocação, gerando um vetor resultante de tamanho igual a soma dos tamanhos dos vetores das invocações. Esse vetor será o parâmetro para a única chamada gerada. A ordem de junção é determinada pela ordem de invocação da *GatherFacet*.
  - **`@ParamGathering ( Strategy.REDUCE )`**: Se diferencia da anotação anterior por realizar uma operação de *reduce* (atualmente fixa como soma) nos vetores das invocações, resultando em um vetor de tamanho igual aos vetores de cada invocação.
  - **`@ResultDispatch ( Strategy.BROADCAST )`**: Copia o valor de retorno para todas as invocações realizadas.
  - **`@ResultDispatch ( Strategy.SCATTER )`**: Se o retorno de uma invocação a uma *GatherFacet* for uma sequência, a mesma é dividida em um número de partes igual ao número de invocações realizadas e essas partes são retornadas separadamente em cada uma dessas invocações. A ordem de envio das partes (retorno das partes) é definida pela ordem de chegada das invocações.

4.3 Geração de Código

Para tornar possível essa característica das ICs de realizar a comunicação paralela através de funções da semântica da própria aplicação, seria preciso que para cada conector definido pelo programador da aplicação (seja *GatherFacet* ou *MulticastReceptacle*) fosse gerado um código que fizesse a ponte entre os tipos de dados e assinaturas dos métodos (nome, quantidade de parâmetros, tipo dos parâmetros) definidos com o código específico de redistribuição de dados indicado pelas anotações. Para isso, foi proposta a implementação de uma um parser de IDL que gere os fontes necessários utilizando-se das informações contidas na interface IDL do conector. Esse parser deve utilizar informações como as assinaturas dos métodos e as anotações para montar o código de redistribuição.

Nesse sentido foram criadas algumas aplicações que exemplificam o uso dos conectores coletivos com diferentes políticas de redistribuição (o Capítulo 5 apresenta algumas destas aplicações), porém, com esse código de redistribuição sendo montado manualmente. Isso permitiu que fossem identificados os pontos de variação de código, tornando mais rápida a montagem desse código a cada nova implementação.

O próximo passo desse processo seria a implementação desse parser que, de posse desses modelos de código já implementados e da verificação sintática da interface IDL do conector, fosse capaz de gerar o código necessário para que a implementação do mesmo.

No estado atual de implementação do SCS-Collective a geração de código ainda não é feita de forma automática, pois realiza-se a montagem do código manualmente.

4.4 API

A API do SCS-Collective foi definida de forma a facilitar a utilização dos conectores paralelos. No Código 4.3 está disposta a IDL do SCS-Collective.

Código 4.3: Interfaces OMG IDL implementadas pelo SCS-Collective

```
1  /************************
2   * scs-collective.idl
3   * Autor: Paulo da Silva
4   ************************/
5  #include "scs.idl"
6
7  module scs {
8    module core {
9      module collective {
10       /** Facet for collective control operations */
11       interface ICollective{
12
13         /** Gathercast configuration methods */
14         void registerGatherFacet ( in string gatherFacetName ) ;
15         void unregisterGatherFacet ( in string gatherFacetName ) ;
16       };
17
18       };
19     };
20   };
21 };
```

O papel básico de cada interface e seus respectivos métodos do Código 4.3 são os seguintes:

- **`ICollective`**: Faceta responsável pela configuração das *GatherFacet* e *MulticastReceptacle* adicionadas ao componente.

– **registerGatherFacet**: Avisa ao controle dessa *GatherFacet* que existe mais um receptáculo conectado (*registerGatherFacet*) e que as devidas operações de adição de uma nova conexão deve ser executadas. Primeiro deve-se adquirir uma referência remota para a *sub-interface* externa de uma determinada *GatherFacet* para somente depois invocar `registerGatherFacet` passando o nome dessa *GatherFacet* por parâmetro.

– **unregisterGatherFacet** realiza o trabalho inverso, notificando que determinado receptáculo foi desconectado e que as operações relacionadas (reinício de semáforos, decremento de contadores, etc) devem ser executadas.

4.5 Considerações Finais

Utilizando-se do conceito de *Interfaces Coletivas* para a definição de dois novos conetores paralelos para o *middleware* de Componentes SCS, a ferramenta proposta neste trabalho oferece uma forma simples e flexível de comunicação coletiva.

Através do uso de um conjunto de anotações na interface IDL é possível configurar as políticas de redistribuição de dados para um determinado conector coletivo. Para a geração automática do código de redistribuição de dados foi proposta a criação de um parser de IDL que identifica as configurações de redistribuição anotadas na interface e produz o código necessário.

Outra característica importante da ferramenta proposta neste documento é a sua implementação na linguagem C++, que é uma linguagem muito usada em aplicações científicas de processamento massivo, o que o torna mais adequado para computação paralela em ambiente de *clusters*.

Apesar dessas características, a implementação do SCS-Collective possui algumas limitações. A geração de código ainda não é automática, sendo feita através da montagem de blocos de código (referentes a cada política de redistribuição) manualmente. Uma importante continuação deste trabalho é a conclusão da ferramenta de geração de código, tendo como base os padrões de estrutura de código que já foram identificados a partir da montagem manual da implementação dos conectores coletivos.

Seria interessante também que a estrutura que define um *MulticastReceptacle* apresentada no Código 4.1 não fosse uma extensão da estrutura que define receptáculos do SCS. A adição dos campos **external_interface_name** e **internal_interface_name** na estrutura padrão obriga a se ter ciência deles, mesmo que o receptáculo não seja um *MulticastReceptacle*. A criação de

uma estrutura específica seria uma boa solução, pois desacoplaria a definição de um receptáculo padrão SCS da definição do *MulticastReceptacle*.

Ainda em relação ao *MulticastReceptacle*, um ponto que merece atenção é a realização das invocações paralelas que atualmente é feita através da criação de uma *thread* para cada conexão. Essa estratégia, com o aumento do número de conexões, pode degradar o desempenho global da aplicação, pois a criação de *threads* é um processo custoso. Uma alternativa interessante seria a utilização da rotina **`send_multiple_requests_deferred`** definida no padrão CORBA DII. Essa rotina possibilita a realização de invocação assíncrona de múltiplas requisições, o que se adequa muito bem a funcionalidade de invocações paralelas do *MulticastReceptacle* e evita a criação e gerenciamento explícitos de *threads*.

Outro aspecto passível de melhoria é o código que implementa as políticas de redistribuição, pois ainda existe muito espaço para refatoração e otimização. O conector *GatherFacet*, por exemplo, realiza 3 indireções de chamadas após receber a invocação de um determinado método de sua *sub-interface* externa (como se pode ver na Figura 4.4). Acreditamos que pelo menos uma dessas indireções poderia ser eliminada.

O SCS-Collective, assim como o SCS, não suporta assincronismo diretamente em seus conectores, sendo essa uma das características que podem ser adicionadas em trabalhos futuros.

Um aspecto importante para sistemas de paralelização que ainda não possui suporte no SCS-Collective é a tolerância a falhas. Algumas soluções simples nesse sentido podem ser aplicadas, como, por exemplo, a adição de um limite de tempo (*timeout*) no conector *GatherFacet*, como é feito na implementação padrão das ICs [33]. Caso algum cliente conectado não realize a chamada, o conector poderá retornar um aviso ou mesmo continuará a execução sem aquele cliente que falhou. Essa estratégia também é aplicável para o conector *MulticastReceptacle* no caso de alguma faceta (*sub-interface* externa) invocada não retornar, e, neste caso, a chamada deverá ser refeita assim como a redistriuição dos dados.

Na Tabela 4.1 é ilustrado o nível de adequação do SCS-Collective aos critérios aplicados na comparação dos trabalhos relacionados descritos no Capítulo 2.

Como se pode verificar nessa tabela, o *middleware* para Computação Paralela SCS-Collective apresenta características bem parecidas com os sistemas GCM e CCA. No quesito de dimensões de paralelismo pode-se ver que SCS-Collective é semelhante ao CCA, suportando paralelização SPMD, MPMD e sendo síncrono. A abstração de programação utilizada pelo

Critérios	**SCS-Collective**
Dimensões de paralelismo	SPMD (SCMD) e MPMD não suporta assincronismo
Abstração de programação	Componentes de Software
Plataformas de execução preferencial	*Clusters*
Arquitetura de comunicação coletiva	*Interfaces Coletivas*

Tabela 4.1: Adequação do SCS-Collective aos critérios elicitados no Cap. 2

SCS-Collective é a mesma do CCA e GCM: Componentes de Software. Em relação a plataforma de execução preferencial, o SCS-Collective se assemelha ao CCA, apesar de não oferecer suporte direto para ambientes multinúcleo como o CCA, podendo, com algumas restrições, ser utilizado também em grades computacionais, ambiente padrão para o GCM. O suporte a comunicação coletiva no SCS-Collective é alcançado através do mesmo mecanismo utilizado pelo GCM, as ICs.