8
Uma Arquitetura Distribuída para Sistemas de Workflow

Este capítulo apresenta uma arquitetura distribuída para um sistema de workflow incluindo o mecanismo de tratamento de exceção proposto nesta tese. A Seção 8.1 apresenta uma introdução à arquitetura distribuída para a execução de processos. A Seção 8.2 descreve sucintamente as características principais da linguagem ACME. A Seção 8.3 discute a arquitetura proposta. Por fim, a Seção 8.4 apresenta uma discussão sobre a execução de instâncias de processo no ambiente distribuído. A Seção 8.5 apresenta um resumo deste capítulo.

8.1
Introdução à Arquitetura Distribuída

O capítulo anterior apresentou a máquina abstrata para OWL-S com o propósito de especificar a semântica dos construtores básicos e adicionais e esclarecer alguns aspectos de OWL-S que a proposta de padrão deixava em aberto.

Este capítulo representa um passo adicional ao sugerir uma arquitetura para nortear uma implementação para um sistema de workflow incluindo o mecanismo de tratamento de exceção para a flexibilização proposto. Ao mesmo tempo, o capítulo estende a discussão ao considerar a questão da execução distribuída de processos.

Brevemente, a arquitetura distribuída para o sistema de workflow considera várias máquinas de execução, chamadas aqui de *gerentes de processos*, onde cada uma delas é responsável por executar instâncias de processos.

Por simplicidade, assumiremos um ambiente distribuído perfeito em que:

– todas as mensagens enviadas são recebidas corretamente, sem perda, troca de ordem ou conteúdo corrompido;

– os gerentes de processo executam corretamente, sem interrupção indesejada e sem nunca perderem dados.

Em mais detalhe, sejam p (a definição de) um processo e p' um subprocesso de p. Seja M um *gerente de processos* controlando a instância P de p. Se, na sua definição, p' pode ser delegado para ser executado em outro *gerente de processos*, M pode distribuir p' para, digamos, N. Este gerente N então cria uma instância P' de p'. Quando isso ocorre, dizemos que M *delegou* p' para N, que M *coordena* N, que N é *diretamente subordinado* a M, e que P' é *diretamente subordinado* a P. Além disso, definimos o relacionamento *é subordinado a* entre *gerentes de processos* como o fecho transitivo do relacionamento *é diretamente subordinado a* e, do mesmo modo, para as instâncias de processos, gerando uma hierarquia de gerentes de processos e de transações aninhadas, conforme discutido no Capítulo 2.

Observe que um *gerente de processos* pode estar subordinado a zero ou mais *gerentes de processos*. Se um gerente N recebeu de dois outros gerentes dois processos para execução, então N é subordinado destes dois gerentes. A subordinação de cada gerente N a outro *gerente de processos* diz respeito exclusivamente à instância gerada a partir do processo recebido deste gerente para execução.

Essa delegação de processos, comum em um ambiente distribuído, não tinha semântica na máquina abstrata, na qual todas as informações de execução estavam contidas. Agora, no ambiente distribuído, faz-se necessária a propriedade booleana de processo que chamamos de *canBeDelegated* (veja Apêndice *A*), pertencente ao *namespace p-ext*, que indica se um processo pode ser transferido para um outro *gerente de processos*. Quando a instância subordinada termina, o *gerente de processos* no qual ela estava sendo executada é responsável por sinalizar o término da execução da instância ao gerente coordenador.

A propriedade *p-ext:canBeDelegated*, da mesma forma que a propriedade *p-ext:flexibilize* descrita nos capítulos anteriores, pode ser aplicada a instâncias de processo ou a instância da classe *process:Perform*. Isso significa que, se um processo possui esta propriedade definida, é porque a delegação deve respeitar o valor desta propriedade.

No entanto, se a propriedade *p-ext:canBeDelegated* não está definida para um processo, ela o pode estar para a chamada deste processo (por meio de uma instância da classe *process:Perform*) dentro de uma composição de processos. Neste caso, a propriedade indica se, naquele contexto, ou seja, quando executado naquela composição, o processo pode ser delegado a outro *gerente de processos*. Isso permite que a delegação seja dependente

do contexto no qual o processo está inserido, ou seja, do processo composto a partir do qual este processo está sendo invocado, e não apenas do processo em si.

A delegação de processos de um *gerente de processos* M para outro gerente N é realizada através da mensagem *runProcess*, conforme apresentado na página 245.

Mensagens também são necessárias para:

- o registro de componentes ativos no sistema;
- o registro e a atualização do registro de instâncias de processo em execução no sistema;
- a delegação e a terminação de processos;
- a execução do mecanismo de tratamento de exceção temporal, para uso de valor default, para substituição, por concretização (de processos ou recursos) e por cancelamento; e
- a consulta ao valor de parâmetros de uma determinada instância de processo.

Por analogia ao capítulo anterior, chamaremos uma instância coordenadora Q de uma instância P de sua superinstância.

É importante frisar aqui que no ambiente distribuído a diferença com relação à execução de uma instância de processo na máquina abstrata é puramente uma questão de onde estão disponíveis os dados e qual componente do sistema o pode verificar. A MAE, apresentada no Capítulo 7, tinha o controle completo sobre todas as instâncias. No entanto, no ambiente distribuído, as informações de execução estão espalhadas entre os *gerentes de processos* e o mecanismo de tratamento de exceção proposto está implementado fora do *gerente de processos*, de modo que fique acessível a todos os gerentes no sistema.

Além disso, no ambiente distribuído, existe a necessidade de uma gerência de componentes para que todas as informações necessárias sejam alcançáveis pelos componentes.

8.2 Preliminares: a Linguagem ACME

Para permitir uma descrição detalhada da arquitetura de implementação de um sistema de gerência de workflows, foi utilizada a linguagem ACME (*Architectural Description Interchange Language*). Em [48] é possível encontrar uma discussão abrangente sobre o papel da arquitetura de software, incluindo o de permitir o desenvolvimento de softwares a partir de descrições mais compreensíveis, que permitam o reuso, a evolução e a gerência do software. A definição da arquitetura proposta pode ser encontrada na Seção 8.3.

8.2.1 Características principais de ACME

Atualmente, existem na literatura diversas linguagens de descrição de arquitetura, as chamadas ADLs, do inglês *Architecture Description Language*. Uma ADL está focada na estrutura de alto nível das aplicações, e não em seus detalhes de implementação.

No entanto, normalmente cada ADL está voltada para atender um domínio específico de aplicação. Desta forma, as diferenças passam a ser maiores e a tornarem cada vez mais difícil que uma descrição de uma arquitetura em uma ADL seja processada por um ferramenta que compreende uma outra ADL.

Conforme bem colocado em [78], *ACME não é uma ADL*, mas sim foi projetada para permitir o mapeamento de descrições arquiteturais de uma ADL para outra, permitindo ainda a inclusão de anotações relativas a informações específicas de uma determinada ADL. Neste mesmo artigo pode ser encontrada uma ampla comparação entre ACME e diversas ADLs existentes.

ACME foi desenvolvida partindo da premissa de que existem características comuns às ADLs suficientes para descrever, de forma independente da linguagem, uma arquitetura de software [49].

O objetivo principal de ACME é, portanto, fornecer um formato de intercâmbio para ferramentas e ambientes de desenvolvimento de arquiteturas de software. Dentre os objetivos secundários, podem ser citados:

– fornecer um esquema representacional que permita o desenvolvimento de novas ferramentas para análise e visualização de descrições

arquiteturais;

- permitir o desenvolvimento de novas ADLs; e
- permitir a especificação de descrições arquiteturais de fácil entendimento e escrita.

Para alcançar esses objetivos, ACME basicamente fornece:

- um vocabulário para a representação da estrutura arquitetural;
- um framework semântico para a anotação de estruturas arquiteturais com propriedades específicas de cada ADL;
- um mecanismo para a especificação de famílias arquiteturais, para a abstração de arquiteturas ou elementos arquiteturais comuns e reusáveis;
- um framework semântico que permite a especificação de restrições impostas à arquitetura.

8.2.2 Vocabulário de ACME

O vocabulário de ACME é formado por 7 tipos de entidades para a representação de arquiteturas:

1. *components*: os componentes em ACME são os elementos mais importantes na composição de um sistema. Eles geralmente são os servidores, os clientes, os bancos de dados e os *blackboards* do sistema;

2. *connectors*: representam a interação (comunicação e coordenação) entre os componentes do sistema;

3. *systems*: representam uma particular configuração de componentes e conectores;

4. *ports*: são as interfaces de um componente. Cada porta identifica um ponto de interação entre o componente e o ambiente;

5. *roles*: são as interfaces de um conector. Cada papel (*role*) define um participante na interação que o conector representa. Cada conector pode ter dois ou mais papéis;

6. *representations*: permitem a descrição hierárquica de arquiteturas, possibilitando que componentes e conectores sejam especificados através de descrições em um nível mais detalhado. Tais descrições são conhecidas como representações;

7. *repmaps*: acrônimo para *representation-map*, os *repmaps* em ACME são necessários quando da utilização de representações. Um *repmap* indica a correspondência entre a representação interna do sistema e a interface externa do componente ou conector que está sendo representado.

8.2.3 Framework Semântico para Anotação

Além do vocabulário fixo, ACME fornece um framework semântico que provê uma forma de anotação para elementos arquiteturais específicos de uma determinada ADL. Estas anotações são feitas na forma de propriedades, sendo que cada uma possui um nome, um tipo (opcional) e um valor.

No entanto, propriedades não são interpretadas por ACME, que apenas pré-define tipos simples tais como inteiros, strings e booleanos. Algumas ferramentas de análise e manipulação de descrições de ADL, por outro lado, muitas vezes as interpretam e, nestes casos, o tipo de cada propriedade é útil porque indica a sublinguagem na qual a propriedade está especificada. Geralmente, a ferramenta que não consegue interpretar uma propriedade específica ou um tipo específico deve deixá-lo sem interpretação, porém disponível para outra ferramenta.

8.2.4 Famílias Arquiteturais em ACME

Uma característica bastante importante de ACME é o mecanismo de especificação de famílias arquiteturais. Uma família define um vocabulário específico para um domínio de aplicação, em conjunto com restrições de como utilizar este vocabulário. Neste vocabulário são definidos os padrões de elementos de arquitetura que ocorrem com freqüência em determinados sistemas e que podem, assim, ser reusados. Estes padrões criados nas famílias são instanciados aplicando a eles os tipos de parâmetros apropriados. Estes padrões são, portanto, estruturas sintáticas que podem ser estendidas para a produção de novas declarações arquiteturais. Padrões podem ser: tipos de componentes, tipos de conectores, tipos de portas, tipos de papéis e tipos de propriedades.

Se um elemento e em um sistema satisfaz um tipo T de uma família, então e contém a estrutura requerida e as propriedades especificadas em T, e e satisfaz necessariamente todas as restrições do tipo *invariant* definidas em

T (veja Seção 8.2.5). Assim, as especificações de uma família são utilizadas tanto para criar instâncias de elementos daquela família quanto para verificar se a instância criada do elemento satisfaz as restrições especificadas para aquele tipo de elemento.

Se um sistema necessita utilizar algum dos padrões definidos em uma família, basta especificar que este sistema utiliza aquela família e, então, torna-se possível o uso de qualquer um dos padrões desta família o número de vezes que for necessário para a construção do sistema, sem a necessidade de redefinição de cada elemento.

A utilização de famílias arquiteturais, ainda, aumenta a legibilidade e a capacidade de abstração de uma descrição arquitetural em ACME.

Para uma família ACME, pode ser definida uma estrutura mínima requerida, composta pelo conjunto mínimo de componentes, conectores, portas, papéis e propriedades necessários para a formação de um sistema nesta família.

8.2.5 Framework Semântico para a Especificação de Restrições

Esse framework semântico de ACME possui um papel importante na especificação de uma arquitetura de software, pois permite o mapeamento de aspectos estruturais da linguagem em um conjunto de relações e restrições. Além de permitir o mapeamento, ele permite que novas restrições sejam impostas sobre a arquitetura, em um nível lógico de descrição não possível puramente com o vocabulário de ACME.

Esse framework semântico é baseado em Lógica de Primeira Ordem. A partir desta linguagem, são especificadas as restrições impostas sobre a arquitetura. Cada restrição pode ser de dois tipos:

1. *invariant*, quando ela é uma regra que não pode ser violada; ou
2. *heuristic*, quando é uma regra que precisa ser observada, mas não necessariamente cumprida.

A violação de uma *invariant* torna a especificação arquitetural inválida, ao passo que a de uma heurística é tratada como um *warning* apenas [50].

Tanto *invariants* quanto *heuristics* podem ser impostas sobre a família de um modo geral ou sobre um tipo específico definido na família. As declarações de *invariant* de um tipo T, por exemplo, definem um conjunto de predicados que devem ser verdadeiros para todas as instâncias deste tipo.

Armani [83] é a linguagem de projeto de arquiteturas utilizada pela ferramenta ACMEStudio para a especificação das restrições. ACMEStudio (`http://www-2.cs.cmu.edu/~acme/AcmeStudio/AcmeStudio.html`) foi desenvolvida como um plugin do framework Eclipse (`www.eclipse.org`) pelo grupo *ABLE* da Carnegie Mellon University, para a criação e manipulação de descrições arquiteturais em ACME.

8.3 Definição da Arquitetura

A arquitetura proposta para um sistema de gerência de workflows é definida em ACME através do uso do conceito de *família*. A família *FamiliaExecucaoProcessos* criada possui tipos de componentes, conectores, portas e papéis, além das regras de restrições lógicas, conforme descrito nas subseções seguintes.

Basicamente, essa arquitetura consiste de três tipos de componentes:

- o tipo *GerenteCentral*, ou abreviadamente *GC*, responsável por controlar todo o fluxo de mensagens e oferecer os serviços de gerência de componentes e de instâncias;
- o tipo *GerenteOntologias*, ou abreviadamente *GO*, responsável por armazenar e manipular a ontologia de processos e recursos *pr*, a biblioteca *lib* e as ontologias de aplicação disponíveis, oferecendo um serviço de flexibilização como implementação do mecanismo de tratamento de exceção proposto neste trabalho; e
- o tipo *GerenteProcessos*, ou *GP*, responsável por executar instâncias de processos.

No que se segue, utilizaremos as palavras *gerente central*, *gerente de ontologias* e *gerente de processos* para designar, respectivamente, uma instância de componente do tipo *GerenteCentral*, *GerenteOntologias* e *GerenteProcessos*.

Como mostra a Figura 8.1, existe apenas um *gerente central* e um *gerente de ontologias* na arquitetura proposta, ao passo que podem ser vários os *gerentes de processos* existentes.

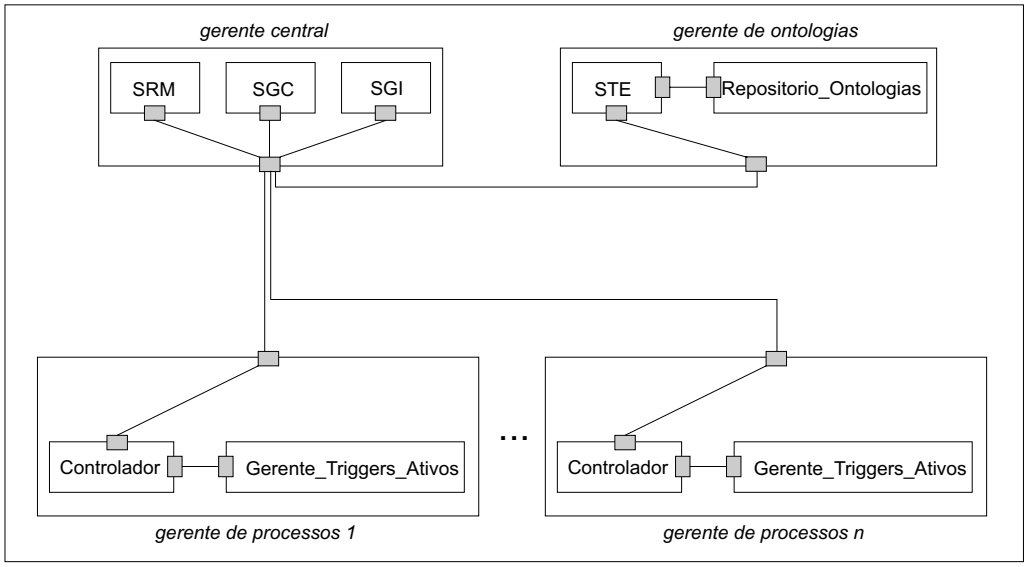


Figura 8.1: Representação simplificada da arquitetura distribuída proposta.

8.3.1
Tipos de papéis, conectores e portas

A família *FamiliaExecucaoProcessos* possui dois tipos de papéis, um tipo de conector e um tipo de porta.

Os tipos de papéis definidos são *rComponentT* e *rManagerT*. O primeiro deles é o papel assumido por cada componente do sistema na interação com o *gerente central*. O outro papel é o assumido por este gerente durante a interação com os demais componentes.

O tipo de conector *conCom* representa as interações entre cada um dos componentes e o *gerente central*. Neste tipo de conector, são dois os papéis: um do tipo *rComponentT* e outro do tipo *rManagerT*.

A família *FamiliaExecucaoProcessos* possui ainda um tipo de porta, o tipo *pCom*, que possibilita a interação entre os componentes. Através de uma porta deste tipo, cada componente recebe e envia mensagens aos outros componentes com os quais estão interligados.

Dessa forma, cada componente está conectado com outro componente por meio de uma porta desse tipo, através de um conector do tipo *conCom* que possui os dois tipos de papéis já definidos.

O Quadro 40 apresenta a descrição em ACME desses tipos descritos.

```
Role Type rComponentT = {  }

Role Type rManagerT = { }
```

```
Port Type pCom = { }

Connector Type conCom = {
   Role rComponent : rComponentT =  new rComponentT;
   Role rManager : rManagerT =  new rManagerT;
}
```

Quadro 40 − Descrição em ACME dos tipos de papel, porta e conector.

8.3.2 Gerente Central

O *gerente central* do sistema é um componente do tipo *GerenteCentral* (*GC*), responsável por manter todo o sistema em funcionamento, provendo os seguintes serviços, conforme apresentado na Figura 8.2:

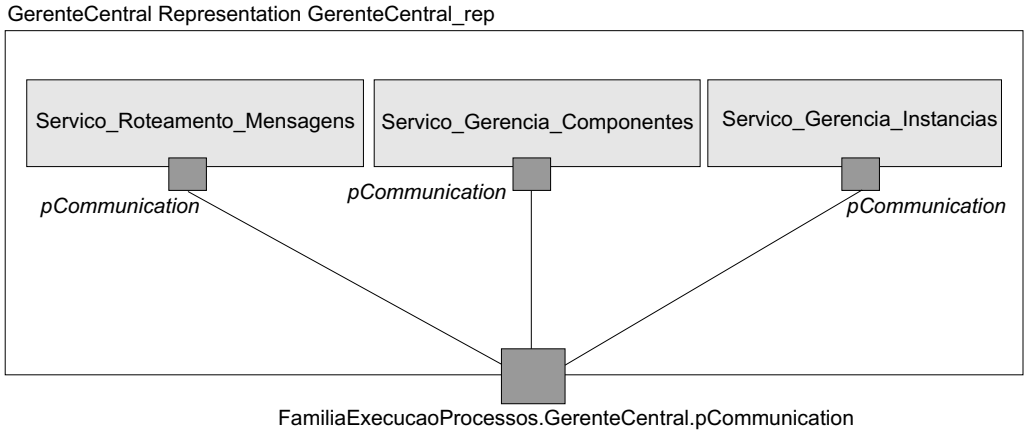


Figura 8.2: Representação interna (*repmap*) do tipo de componente *GerenteCentral* (*GC*).

serviço de roteamento de mensagens (*SRM*) representado na figura pelo subcomponente *Servico_Roteamento_Mensagens*;

serviço de gerência de componentes (*SGC*) representado na figura pelo subcomponente *Servico_Gerencia_Componentes*;

serviço de gerência de instâncias (*SGI*) representado na figura pelo subcomponente *Servico_Gerencia_Instancias*.

Vale ressaltar que cada subcomponente que representa os serviços é definido como parte do tipo de componente *GerenteCentral*.

O Quadro 41 apresenta a descrição em ACME do tipo de componente *GerenteCentral*.

```
Component Type GerenteCentral = {
  Port pCommunication : pCom =  new pCom;

  Representation GerenteCentral_rep = {
    System GerenteCentral_rep = {
      Component Servico_Roteamento_Mensagens = {
        Port pCommunication;
      };

      Component Servico_Gerencia_Componentes = {
        Port pCommunication;
      };

      Component Servico_Gerencia_Instancias = {
        Port pCommunication;
      };
    };

    Bindings {
       Servico_Roteamento_Mensagens.pCommunication
            to pCommunication;
       Servico_Gerencia_Componentes.pCommunication
            to pCommunication;
       Servico_Gerencia_Instancias.pCommunication
            to pCommunication;
    }
  };
}
```

Quadro 41 − Descrição em ACME do tipo de componente *GerenteCentral*.

Observe que a porta de comunicação de cada um dos componentes internos do tipo *Gerente_Central* (*Servico_Roteamento_Mensagens*, *Servico_Gerencia_Componentes* e *Servico_Gerencia_Instancias*) com o ambiente externo é chamada *pCommunication*. Observe ainda que a porta de comunicação do *Gerente_Central* com o ambiente externo também é chamada *pCommunication* e é do tipo *pCom* definido.

O *SRM* é o serviço do *gerente central* responsável por capturar cada uma das mensagens enviadas pelos demais componentes do sistema e enviá-las ao destinatário especificado. Este tipo de serviço evita que sejam necessárias conexões *peer-to-peer* entre os componentes do sistema que precisem se comunicar.

No entanto, o serviço *SRM* não representa a única solução para a interação entre os componentes do sistema, mas é com certeza uma abordagem bastante simples. Porém, com ele, o *gerente central* passa a representar um ponto de falha no sistema. Se ele vir a falhar, todo o sistema fica inoperante, porque as mensagens não podem ser entregues a seus destinatários.

Uma arquitetura *peer-to-peer* seria uma alternativa a considerar. Note, porém, que se o número de componentes cresce no sistema, uma quantidade considerável de conexões abertas pode ser subutilizada.

Uma outra alternativa possível seria adotar um esquema de *broadcast*. Neste caso, cada componente receberia as mensagens de todos os demais, processando só as que lhe fossem endereçadas. Esta abordagem tem a grande desvantagem de acarretar um fluxo desnecessário de mensagens entre os componentes, o que muitas vezes prejudica o desempenho do sistema.

Todas as mensagens trocadas no sistema possuem o formato:

nome_da_mensagem(<*componente_que_envia*>, <*componente_que_vai_processar*>, <*serviço*>, <*parâmetros*>)

onde:

- *componente_que_envia* pode ser um componente do tipo *GerenteCentral* ou do tipo *GerenteOntologias* ou do tipo *GerenteProcessos*;
- *componente_que_vai_processar* pode ser um componente do tipo *GerenteCentral* ou do tipo *GerenteOntologias* ou do tipo *GerenteProcessos*;
- *serviço* é um dos serviços oferecidos pelo *componente_que_vai_processar*;
- {*parâmetros*} é a lista de parâmetros do serviço.

O *SGC*, por sua vez, é o serviço responsável por manter controle sobre quais componentes estão ativos no sistema. Toda vez que um componente deseja participar do sistema, ele precisa enviar ao *SGC* a sua identificação (seu endereço e o identificador de sua porta de comunicação), através da mensagem *registerComponent*. No processamento desta mensagem, o *SGC* armazena localmente um registro do componente com as descrições enviadas.

O Quadro 42 resume como o *SGC* processa a mensagem *registerComponent*.

registerComponent(*idComponente*, *idGerenteCentral*, *SGC*, *endereçoComponente*, *id_pCom*) - crie um registro interno para o componente *idComponente*, cujo endereço é *endereçoComponente* e cuja porta para envio/recebimento de mensagens é *id_pCom*

Quadro 42 – Mensagem *registerComponent* processada pelo *SGC*.

O *SGC* também pode ser utilizado para consulta sobre componentes ativos, como poderá ser visto na Seção 8.4.

O *SGI* é o serviço responsável por manter controle local de todas as instâncias de processo em execução no sistema e, temporariamente, das já terminadas.

O registro de uma instância P mantido pelo *SGI* armazena os seguintes dados, recebidos do *Controlador* de cada *gerente de processos* (veja exemplo da Figura 8.3, página 230):

- o identificador do processo p (entrada *process* na figura);
- o identificador da instância P (entrada $instanceId$ na figura);
- o identificador do *gerente de processos* M no qual a instância P está sendo executada (entrada *idGerenteProcessos* na figura);
- o identificador da instância coordenadora (entrada $superInstance$ na figura);
- o identificador do *gerente de processos* N no qual a superinstância está sendo executada (entrada $idGerenteProcessosSuperInstance$ na figura);
- o identificador de cada uma de suas subinstâncias (entrada $\{subInstances\}$);
- o identificador (cadeia de caracteres) do tipo da instância, como aludido no Capítulo 7. Este tipo identifica a instância como sendo uma instância de um processo previsto no fluxo de controle (valor "*null*"), uma instância de tratamento de exceção temporal de um outro processo (valor "*temporal*"), uma instância de um processo concreto relativo a um processo abstrato (valor "*concretização*"), uma instância relativa a um processo substituto de outro processo (valor

"*substituição*") ou uma instância para desfazer os efeitos de outra instância (valor "*undo*"). Este tipo é identificado na figura pela entrada $instanceType$;

- o momento de início da instância P (entrada $initTime$ na figura);
- o estado corrente e o momento em que este estado foi alcançado (entrada $(state, timestamp)$ na figura);
- a substituição de recursos, através da entrada $\{(original\_r, substitute\_r)\}$ apresentada na figura;
- a concretização de recursos, através da entrada $\{(abstract\_r, concrete\_r)\}$ apresentada na figura;
- o identificador da instância original (entrada *pointer_original_p*), caso a instância em questão seja uma instância de flexibilização (que tenha sido criada para tratar alguma exceção levantada);
- o identificador da instância criada para tratar de alguma exceção levantada por P (entrada $pointer\_flex\_p$);
- o identificador do processo abstrato que a instância está representando, através da execução de um processo concreto associado (entrada *abstract_p* na figura);
- a tabela de mapeamento de parâmetros entre o processo original e o processo substituto ou entre o processo abstrato e o processo concreto escolhido (conforme pares de parâmetros informados na mensagem de atualização), se for o caso (entrada *parameters_mapping* na figura).

Esse conjunto de registros de instâncias mantido pelo *SGI* é chamado aqui de *banco de dados de instâncias*. Observe que cada registro de instância contém as mesmas informações mantidas pelas instâncias na MAE e, adicionalmente, informação sobre qual *gerente de processos* a está executando.

O *SGI* adota uma política de limpeza do banco de dados de instâncias, descrita a seguir, e exemplificada na Figura 8.3. Trata-se, basicamente, de um algoritmo recursivo simples:

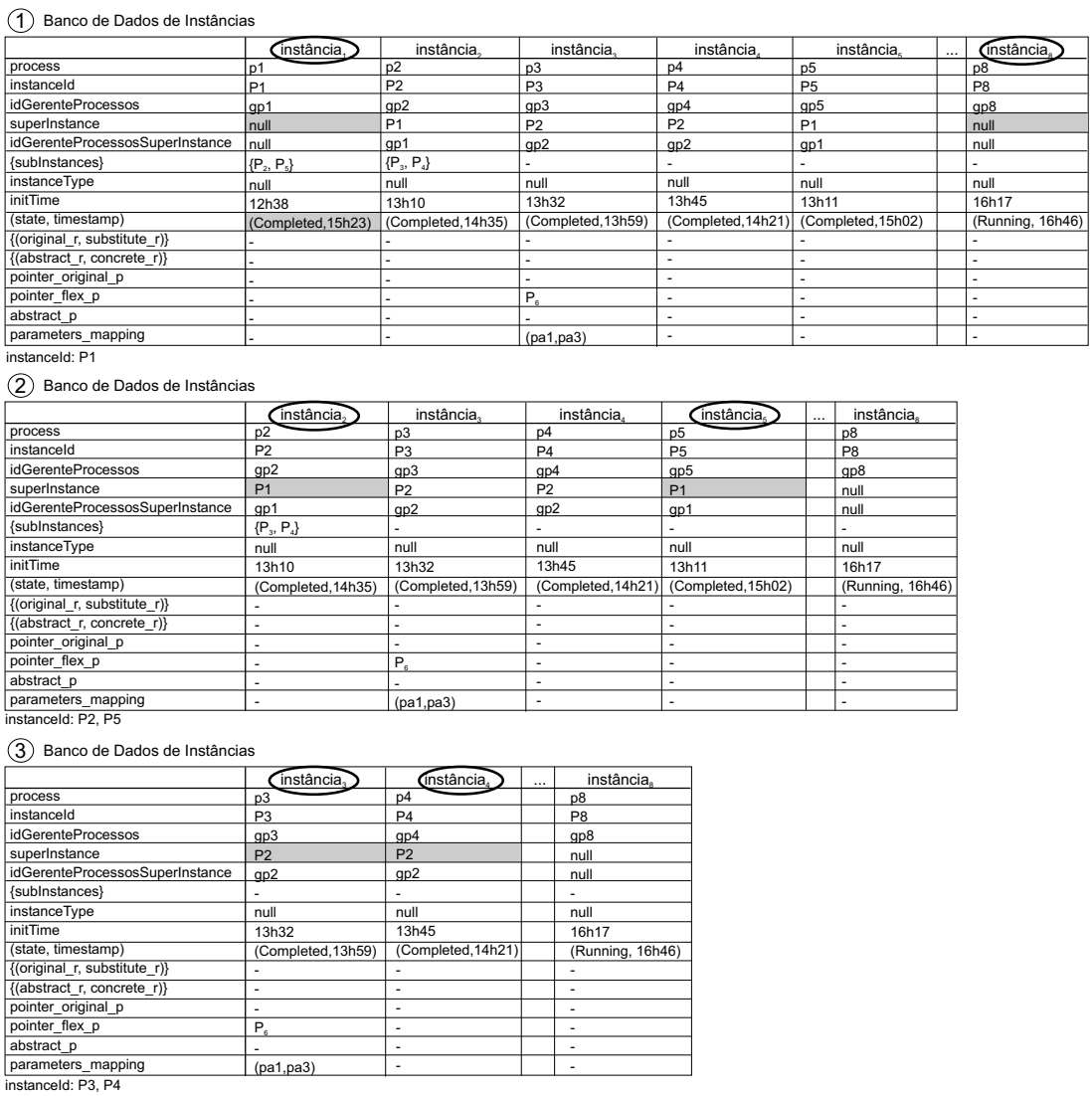
1. A intervalos definidos internamente no *SGI*, o *SGI* realiza uma varredura para determinar as instâncias que não possuem superinstância.

2. Dentre estas instâncias que não possuem superinstância, o *SGI* determina aquelas que já terminaram, ou seja, que alcançaram o

estado *Completed*, *Aborted*, *Undone* ou *Forced* (instâncias no estado *Skipped* não são consideradas porque podem ter provocado a execução de outras instâncias substitutas). O *SGI* copia os identificadores destas instâncias para uma área temporária e apaga as entradas correspondentes.

3. O *SGI* determina todas instâncias cujos identificadores das superinstâncias estão na área temporária. O *SGI* copia os identificadores destas instâncias para a área temporária e apaga a entrada correspondente.

4. O *SGI* repete a etapa 3 até que não existam mais instâncias cujos identificadores das superinstâncias estejam na área temporária.



① Banco de Dados de Instâncias

	instância$_1$	instância$_2$	instância$_3$	instância$_4$	instância$_5$	...	instância$_8$
process	p1	p2	p3	p4	p5		p8
instanceId	P1	P2	P3	P4	P5		P8
idGerenteProcessos	gp1	gp2	gp3	gp4	gp5		gp8
superInstance	null	P1	P2	P2	P1		null
idGerenteProcessosSuperInstance	null	gp1	gp2	gp2	gp1		null
{subInstances}	{P_2, P_5}	{P_3, P_4}	-	-	-		-
instanceType	null	null	null	null	null		null
initTime	12h38	13h10	13h32	13h45	13h11		16h17
(state, timestamp)	(Completed,15h23)	(Completed,14h35)	(Completed,13h59)	(Completed,14h21)	(Completed,15h02)		(Running, 16h46)
{(original_r, substitute_r)}	-	-	-	-	-		-
{(abstract_r, concrete_r)}	-	-	-	-	-		-
pointer_original_p	-	-	-	-	-		-
pointer_flex_p	-	-	P_6	-	-		-
abstract_p	-	-	-	-	-		-
parameters_mapping	-	-	(pa1,pa3)	-	-		-

instanceId: P1

② Banco de Dados de Instâncias

	instância$_2$	instância$_3$	instância$_4$	instância$_5$	...	instância$_8$
process	p2	p3	p4	p5		p8
instanceId	P2	P3	P4	P5		P8
idGerenteProcessos	gp2	gp3	gp4	gp5		gp8
superInstance	P1	P2	P2	P1		null
idGerenteProcessosSuperInstance	gp1	gp2	gp2	gp1		null
{subInstances}	{P_3, P_4}	-	-	-		-
instanceType	null	null	null	null		null
initTime	13h10	13h32	13h45	13h11		16h17
(state, timestamp)	(Completed,14h35)	(Completed,13h59)	(Completed,14h21)	(Completed,15h02)		(Running, 16h46)
{(original_r, substitute_r)}	-	-	-	-		-
{(abstract_r, concrete_r)}	-	-	-	-		-
pointer_original_p	-	-	-	-		-
pointer_flex_p	-	P_6	-	-		-
abstract_p	-	-	-	-		-
parameters_mapping	-	(pa1,pa3)	-	-		-

instanceId: P2, P5

③ Banco de Dados de Instâncias

	instância$_3$	instância$_4$	...	instância$_8$
process	p3	p4		p8
instanceId	P3	P4		P8
idGerenteProcessos	gp3	gp4		gp8
superInstance	P2	P2		null
idGerenteProcessosSuperInstance	gp2	gp2		null
{subInstances}	-	-		-
instanceType	null	null		null
initTime	13h32	13h45		16h17
(state, timestamp)	(Completed,13h59)	(Completed,14h21)		(Running, 16h46)
{(original_r, substitute_r)}	-	-		-
{(abstract_r, concrete_r)}	-	-		-
pointer_original_p	-	-		-
pointer_flex_p	P_6	-		-
abstract_p	-	-		-
parameters_mapping	(pa1,pa3)	-		-

instanceId: P3, P4

Figura 8.3: Processo de limpeza do banco de dados de instâncias.

Para executar uma determinada instância P de um processo p, o *Controlador* muitas vezes necessita de valores de parâmetros sobre os quais não tem conhecimento.

Conforme anteriormente discutido, em OWL-S os valores dos parâmetros de entrada de um processo podem ser provenientes basicamente de quatro fontes distintas, como apresentado na Seção 4.1:

- de uma constante;
- do resultado de aplicar uma função, tendo como argumentos constantes e parâmetros;
- de uma instância de uma classe OWL;
- de valores de parâmetros de outros processos OWL-S.

Se os valores são provenientes de uma das três primeiras fontes, então a busca destes valores é trivial. No entanto, se estes valores são provenientes de outros processos OWL-S, o *gerente de processos* que está executando a instância P que necessita destes valores precisa encontrar a instância que os gerou.

O controle de instâncias mantido pelo *SGI* permite a cada *gerente de processos* encontrar valores de parâmetros dos quais depende a execução de uma determinada instância, quando estes valores são provenientes de processos executados em outros *gerentes de processos*.

Essa busca pela instância que gerou os valores dos parâmetros passa, primeiro, por identificar qual instância considerar. Isso porque podem ser várias as instâncias em execução relativas a um mesmo processo p no sistema.

OWL-S tenta resolver o problema de determinar a instância ativa de um determinado processo p através da variável *process:TheParentPerform*, que aponta, em tempo de execução, para a instância corrente do processo p em uma determinada composição. No entanto, esta variável não resolve o problema se são várias as instâncias de um mesmo processo no sistema, conforme discutido no capítulo anterior.

Dessa forma, todas as vezes que um *gerente de processos* precisa do valor de um ou mais parâmetros de um processo p', ele o solicita ao *SGI*, através da mensagem *findParametersValue*.

O Quadro 43 resume como o *SGI* processa essa mensagem.

findParametersValue($idGerenteProcessos, idGerenteCentral, SGI, P, \{(p', \{parameter\})\}$)

- busque no banco de dados de instâncias as entradas relativas às instâncias P' de cada processo p' que mais recentemente alcançou o estado *PreparedToComplete* ou *Forced* ou, a partir de uma instância de p' que alcançou os estados *PreparedToAbort*, *Aborted*, *Undone* ou *Skipped*, a última instância na lista a partir de $pointer\_flex\_p$, que tenha alcançado o estado *PreparedToComplete* ou *Forced*, observando o mapeamento de parâmetros
- verifique, para cada p', a instância encontrada que mais recentemente alcançou o estado *PreparedToComplete* ou *Forced*. Extraia o identificador N do *gerente de processos* que a executou
- para cada instância P' de cada p' encontrada, envie a mensagem *findParametersValue*($idGerenteCentral, N, Controlador, P', \{param\}$), onde cada *param* é o parâmetro especificado em *parameter*, se P' não é uma instância de flexibilização, ou *parameter*$'$, de acordo com o mapeamento de parâmetros, caso P' tenha sofrido flexibilização

Quadro 43 − Mensagem *findParametersValue* processada pelo SGI.

No processamento da mensagem *findParametersValue*, o SGI primeiramente busca no banco de dados de instâncias pelo registro da instância deste processo p' ou da instância que está direta ou indiretamente tratando de uma exceção levantada para p', que mais recentemente alcançou o estado *PreparedToComplete* ou *Forced*, sob a mesma superinstância em execução (considerando sempre que instâncias distintas não interferem nas execuções umas das outras).

Tendo encontrado a instância que terminou mais recentemente com sucesso a execução de p', o SGI solicita ao *gerente de processos* N onde esta instância foi executada o valor dos parâmetros relacionados, segundo especificado na mensagem *findParametersValue*, processada pelo *Controlador* de N.

O Quadro 44 resume como o *Controlador* de um *gerente de processos* processa a mensagem *findParametersValue*.

findParametersValue($idGerenteCentral, idGerenteProcessos, Controlador, P', \{parameter\}$)
- busque no blackboard $\beta[P']$ (veja definição na Seção 7.1) relativo à instância P' o valor *value* de cada parâmetro em $\{parameter\}$ e envie a mensagem *parametersValueFound*($idGerenteProcessos$, $idGerenteCentral, SGI, \{(parameter, value)\}$). Se *value* é desconhecido, envie como valor vazio

Quadro 44 – Mensagem *findParametersValue* processada pelo *Controlador*.

No processamento da mensagem *findParametersValue*, o *Controlador* busca no *blackboard* da instância correspondente o valor de cada parâmetro especificado. Os valores encontrados são enviados ao *gerente central*, através da mensagem *parametersValueFound*.

O Quadro 45 resume como o *gerente central* processa essa mensagem.

parametersValueFound($idGerenteProcessos, idGerenteCentral, SGI$, $\{(parameter, value)\}$)
- envie a mensagem *parametersValueFound*($idGerenteCentral$, $idGerenteProcessos, Controlador, P, \{(parameter, value)\}$)

Quadro 45 – Mensagem *parametersValueFound* processada pelo *gerente central*.

No processamento de *parametersValueFound*, o SGI do *gerente central* apenas repassa os valores recebidos para o *gerente de processos* de direito.

O Quadro 46 resume como o *Controlador* processa essa mensagem.

parametersValueFound($idGerenteCentral, idGerenteProcessos$, $Controlador, P, \{(parameter, value)\}$)
- armazene no *blackboard* $\beta[P]$ os valores *value* de cada parâmetro *parameter*

Quadro 46 – Mensagem *parametersValueFound* processada pelo *Controlador*.

8.3.3
Gerente de Ontologias

O *gerente de ontologias* do sistema é um componente do tipo *GerenteOntologias* que tem por responsabilidade armazenar a ontologia *pr*, as bibliotecas *lib* associadas aos processos e recursos definidos no sistema e as ontologias de aplicação, e implementar o mecanismo proposto de tratamento de exceção para a flexibilização, oferecendo-o como um serviço aos *gerentes de processos* no sistema.

A Figura 8.4 apresenta a representação interna de um componente do tipo *GerenteOntologias*.

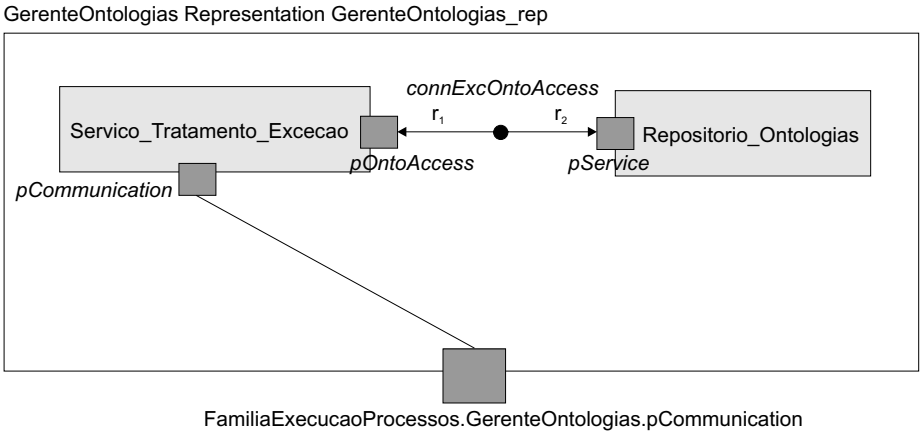


Figura 8.4: Representação interna (*repmap*) do tipo de componente *GerenteOntologias* (*GO*).

O Quadro 47 apresenta a descrição em ACME do tipo de componente *GerenteOntologias*.

```
Component Type GerenteOntologias = {
  Port pCommunication : pCom =  new pCom;

  Representation GerenteOntologias_rep = {
    System GerenteOntologias_rep = {

      Component Repositorio_Ontologias = {
        Port pService;
      };
      Component Servico_Tratamento_Excecao = {
        Port pOntoAccess;
        Port pCommunication;
      };
```

```
      Connector connExcOntoAccess = {
         Role r1;
         Role r2;
      };
      Attachment Repositorio_Ontologias.pService to
           connExcOntoAccess.r2;
      Attachment Servico_Tratamento_Excecao.pOntoAccess to
           connExcOntoAccess.r1;

      Bindings {
        Servico_Tratamento_Excecao.pCommunication
            to pCommunication;
      }
  };
}
```

Quadro 47 – Descrição em ACME do tipo de componente *GerenteOntologias*.

Observe ainda que o *Servico_Tratamento_Excecao* oferecido pelo *GO* possui exatamente duas portas: uma para a ligação com a porta externa do tipo *GerenteOntologias*, a porta *pCommunication*, do tipo *pCom*, e outra para a comunicação com o *Repositorio_Ontologias*, a porta *pOntoAccess*. Este subcomponente, por sua vez, não realiza comunicação direta com nenhum componente externo do sistema e é de acesso exclusivo do serviço de tratamento de exceção, através da porta *pService*. A ligação entre os dois componentes internos *Servico_Tratamento_Excecao* e *Repositorio_Ontologias* no componente do tipo *GO* é feito por meio do conector *connExcOntoAccess*, cujos papéis são denominados *r1* e *r2*.

O serviço de tratamento de exceção oferecido pelo *gerente de ontologias* através do componente *Servico_Tratamento_Excecao*, ou abreviadamente *STE*, é o responsável por:

– dada uma requisição de um *gerente de processos* para uma determinada instância de processo p que sofreu exceção temporal, acessar o *Repositorio_Ontologias* e criar uma lista de processos que tratem a exceção levantada pela instância de p, ordenada de acordo com o modelo de custo empregado. Esta lista, que contém os processos, seus recursos necessários, a informação de mapeamento de parâmetros

e o valor encontrado pelo modelo de custo para cada um deles, é então devolvida como resposta do serviço à requisição efetuada pelo *gerente de processos*;

- dada uma requisição de um *gerente de processos* para um determinado processo abstrato p, acessar o *Repositorio_Ontologias* e criar uma lista com os processos concretos encontrados para p, ordenada de acordo com o modelo de custo empregado. Esta lista, que contém os processos, seus recursos necessários, o mapeamento de parâmetros e recursos entre o processo abstrato e cada um dos concretos e o valor encontrado para cada um deles pelo modelo de custo é então devolvida como resposta do serviço à requisição efetuada pelo *gerente de processos*;
- dada uma requisição de um *gerente de processos* para encontrar recursos concretos de um determinado recurso abstrato encontrado na definição de um processo em execução, acessar o *Repositorio_Ontologias* e criar uma lista com os recursos concretos encontrados. Esta lista, ordenada de acordo com o modelo de custo empregado, é devolvida como resposta do serviço à requisição efetuada pelo *gerente de processos*;
- dada uma requisição de um *gerente de processos* para encontrar alternativas a um determinado processo p ou recurso r, acessar o *Repositorio_Ontologias* e criar uma lista com os processos ou recursos, conforme a requisição, ordenada de acordo com o modelo de custo empregado. Esta lista, que contém os processos, seus recursos necessários, o mapeamento de parâmetros e o valor encontrado para cada um deles pelo modelo de custo, no caso de processos alternativos, e recursos e seus valores de acordo com o modelo de custo, no caso de recursos alternativos, é então devolvida como resposta do serviço à requisição efetuada pelo *gerente de processos*;
- dada uma requisição de um *gerente de processos* para encontrar o valor default de um determinado parâmetro de processo, seja o parâmetro de entrada ou de saída, acessar o *Repositorio_Ontologias* e devolver como resposta à requisição o valor default encontrado. Vale lembrar que no máximo um valor default é definido por parâmetro e que parâmetros de processos distintos que fazem parte de um mesmo fluxo de dados devem ter valores default iguais, por questão de consistência. Ainda, estes valores default podem ser obtidos através da propriedade *p-ext:defaultParameterValue* ou de regras de suposição previamente definidas;

– dada uma requisição de um *gerente de processos* para encontrar um processo para desfazer os efeitos de uma instância de um determinado processo atômico *p*, acessar o *Repositorio_Ontologias* e criar uma lista dos processos encontrados, ordenada de acordo com o modelo de custo empregado. Esta lista, que contém os processos, seus recursos necessários e o valor encontrado para cada um deles pelo modelo de custo, é devolvida como resposta do serviço à requisição efetuada pelo *gerente de processos*.

O acesso ao *Repositorio_Ontologias* é realizado por meio de consultas efetuadas em linguagens de consultas para ontologias. Por exemplo, consultas podem ser formuladas em linguagem RDQL [106] para que, por exemplo, um processo alternativo seja encontrado. Note que não somente é necessário encontrar os processos que são semanticamente próximos a um dado processo *p*, mas também é necessário verificar, por exemplo, a disponibilidade dos recursos necessários para a execução de cada um deles.

8.3.4 Gerente de Processos

Um *gerente de processos* é um componente do tipo *GerenteProcessos*, responsável por controlar a execução de uma ou mais instâncias de processos, e delegar um subprocesso para outro *gerente de processos*, caso esta delegação seja permitida (propriedade *p-ext:canBeDelegated*, explicada na página 217).

Internamente, conforme descrito no Capítulo 7, segundo o conceito de máquina abstrata, e apresentado na Figura 8.5, um *gerente de processos* é composto por dois subcomponentes: o *Controlador* e o *Gerente_Triggers_Ativos*. As funções destes dois subcomponentes já foram apresentadas em mais detalhe no capítulo anterior. A comunicação entre o *Controlador* e o componente *Gerente_Triggers_Ativos* é realizada por meio de trocas de mensagens, realizadas através das portas *pComGT* do *Controlador* e *pComControlador* do *Gerente_Triggers_Ativos*. O *Controlador* de um *gerente de processos* é o seu único subcomponente que processa mensagens vindas do meio externo e, por isso, é determinado o *binding* entre a porta *pCommunication* do *Controlador* com a porta *pCommunication* do componente *gerente de processos*.

O Quadro 48 contém a especificação em ACME do tipo de componente *GerenteProcessos*.

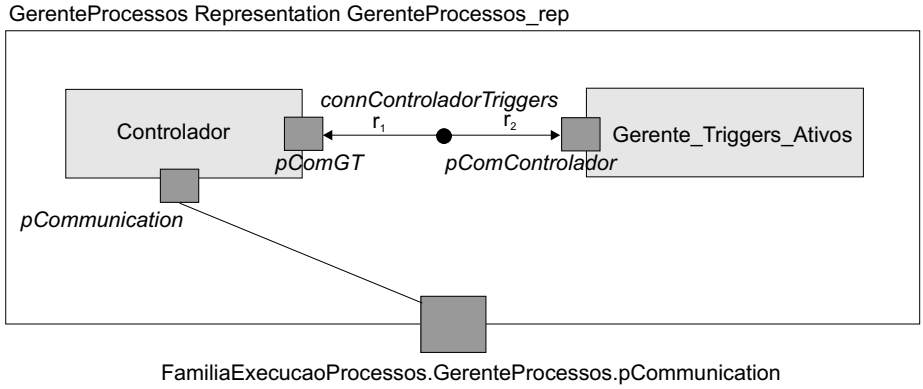


Figura 8.5: Representação interna (*repmap*) do tipo de componente *GerenteProcessos* (*GP*).

```
Component Type GerenteProcessos = {
  Port pCommunication : pCom =  new pCom;

  Representation GerenteProcessos_rep = {
    System GerenteProcessos_rep = {
      Component Controlador = {
        Port pCommunication;
        Port pComGT;
      };
      Component Gerente_Triggers_Ativos = {
        Port pComControlador;
      };
      Connector connControladorTriggers = {
        Role r1;
        Role r2;
      };
      Attachment Controlador.pComGT
          to connControladorTriggers.r1;
      Attachment Gerente_Triggers_Ativos.pComControlador
          to connControladorTriggers.r2;
    };
    Bindings {
       Controlador.pCommunication to pCommunication;
    }
  };
}
```

Quadro 48 – Descrição em ACME do tipo de componente *GerenteProcessos*.

Além disso, cada *gerente de processos* mantém na área de trabalho as instâncias que estão em execução e na área de log os logs relativos a cada uma delas.

Cada instância P mantida na área de trabalho de um *gerente de processos* armazena, além de todas as informações que uma instância na MAE armazena (Capítulo 7), o identificador do *gerente de processos* que a está executando. Além disso, armazena também, junto a cada subinstância, o identificador do *gerente de processos* correspondente. Estes identificadores permitem que o *Controlador* do gerente onde está sendo executada a instância P possa enviar mensagens para os gerentes da superinstância e das subinstâncias diretas de P.

8.3.5 Regras de Projeto

A definição completa da família *FamiliaExecucaoProcessos* inclui, além dos tipos, algumas regras que capturam a semântica adicional sobre os tipos de componentes, portas, conectores e papéis. As regras estão escritas na linguagem Armani [83], baseada em Lógica de Primeira Ordem, utilizada pela ferramenta ACMEStudio.

A Regra R_1 garante que apenas fazem parte de um sistema nessa família componentes do tipo *GerenteProcessos*, *GerenteOntologias* e *GerenteCentral*.

```
/* Regra R1 */

invariant Forall comp : component in self.Components |
   (satisfiesType(comp, GerenteProcessos) OR
    satisfiesType(comp, GerenteOntologias) OR
    satisfiesType(comp, GerenteCentral))
```

Quadro 49 – Regra R_1.

A Regra R_2 garante que, para todos os papéis de todos os conectores de um sistema na família, existe uma porta de um componente ao qual eles estão vinculados.

```
/* Regra R2 */

invariant Forall conn : connector in self.Connectors |
 Forall r : role in conn.Roles |
  Exists comp : component in self.Components |
   Exists p : port in comp.Ports |
    attached(p, r);
```

Quadro 50 − Regra R_2.

A Regra R_3 garante que não existem portas em componentes sem conexão.

```
/* Regra R3 */

invariant Forall comp : component in self.Components |
  Forall p : port in comp.Ports |
    Exists conn : connector in self.Connectors |
      Exists r : role in conn.Roles |
        attached(p, r);
```

Quadro 51 − Regra R_3.

A Regra R_4 garante que todos os componentes têm exatamente uma porta e ela é do tipo *pCom*.

```
/* Regra R4 */

invariant Forall comp : component in self.Components |
  Forall p : port in comp.Ports |
    (size(comp.Ports) == 1 AND satisfiesType(p, pCom));
```

Quadro 52 − Regra R_4.

A Regra R_5 garante que todos os conectores são do tipo *conCom*.

```
/* Regra R5 */

invariant Forall conn : connector in self.Connectors |
  (satisfiesType(conn, conCom));
```

Quadro 53 − Regra R_5.

A Regra R_6 garante que todo conector do tipo *conCom* tem exatamente dois papéis, sendo um do tipo *rComponentT* e outro do tipo *rManagerT*.

```
/* Regra R6 */

invariant Forall con : connector in self.Connectors |
  (satisfiesType(con, conCom) ->
   (size({Select r in con.roles |
    satisfiesType(r, rComponentT) }) == 1 AND
   size({Select r in con.roles |
   satisfiesType(r, rManagerT) }) == 1 AND
   size(con.roles) == 2));
```

Quadro 54 − Regra R_6.

A Regra R_7 garante que, se dois componentes estão conectados, um deles é do tipo *GerenteCentral*.

```
/* Regra R7 */

invariant Forall comp1 : component in self.Components |
  Forall comp2 : component in self.Components |
    Exists con : connector in self.Connectors |
      connected(comp1, comp2) ->
      satisfiesType(comp1, GerenteCentral) AND
      attached(con, comp1) AND attached(con, comp2);
```

Quadro 55 − Regra R_7.

A Regra R_8 garante que existe apenas um componente do tipo *GerenteCentral* e um do tipo *GerenteOntologias* em um sistema dessa família.

```
/* Regra R8 */

invariant
  size({Select compGC in self.Components |
  satisfiesType(compGC, GerenteCentral) }) == 1 AND
  size({Select compGO in self.Components |
  satisfiesType(compGO, GerenteOntologias) }) == 1;
```

Quadro 56 – Regra R_8.

Mesmo tendo definido esse conjunto de regras, ainda se faz necessário um outro conjunto de restrições para impor a natureza de algumas conexões. A definição de um número ainda maior de regras foi evitado pelo uso de *Connection Patterns* na ferramenta ACMEStudio.

Para cada uma das conexões permitidas, há padrões de conexão que a ferramenta interpreta no momento da criação de um sistema na família, evitando a criação de conexões indesejadas.

O Quadro 57 apresenta os padrões de conexão definidos.

```
[GerenteCentral-pCom] | rManagerT -> conCom ->
 rComponentT | [pCom-GerenteOntologias]

[GerenteOntologias-pCom] | rComponentT -> conCom ->
 rManagerT | [pCom-GerenteCentral]

[GerenteCentral-pCom] | rManagerT -> conCom ->
 rComponentT | [pCom-GerenteProcessos]

[GerenteProcessos-pCom] | rComponentT -> conCom ->
 rManagerT | [pCom-GerenteCentral]
```

Quadro 57 – Padrões de conexão definidos para a família *FamiliaExecucaoProcessos*.

O primeiro padrão de conexão garante que um conector do tipo *conCom* conecta um componente do tipo *GerenteOntologias* a um do tipo *GerenteCentral* através de portas do tipo *pCom* nos dois elementos, sendo

que o componente do tipo *GerenteOntologias* assume o papel do tipo *rComponentT*, enquanto que o componente do tipo *GerenteCentral* assume o papel do tipo *rManagerT* na interação. O segundo padrão é o inverso do primeiro. Os próximos dois são análogos, definindo o relacionamento possível entre um componente do tipo *GerenteCentral* com um componente do tipo *GerenteProcessos*.

Observe que as regras em conjunto com os padrões de conexão permitem a correta formação de um sistema nessa família.

8.4 Introdução à Execução de uma Instância de Processo no Ambiente Distribuído

Durante a execução de uma instância P de processo p, se a definição de p permitir, um *gerente de processos* pode delegar seus subprocessos para a execução por outros gerentes.

Como o ambiente é distribuído, antes de delegar um subprocesso a um outro *gerente de processos*, um gerente precisa saber quais são os *gerentes de processos* ativos no sistema. Para isso, ele deve enviar a mensagem *findActiveProcessManagers* para o *SGC*.

O processamento dessa mensagem poderia ser bastante sofisticado tentando balancear a carga em cada gerente. Nesta tese, por simplicidade, assumimos que o *SGC* do *gerente central* apenas busca pelos gerentes ativos no sistema, sem qualquer controle com relação ao número de instâncias em execução em cada *gerente de processos*.

O Quadro 58 resume como o *SGC* processa a mensagem *findActiveProcessManagers*.

findActiveProcessManagers(*idGerenteProcessos*, *idGerenteCentral*, *SGC*) - envie a mensagem *activeProcessManagersListFound*(*idGerenteCentral*, *idGerenteProcessos*, *Controlador*, *List*), onde *List* contém a lista dos *gerentes de processos* ativos no sistema e registrados no *gerente central*

Quadro 58 − Mensagem *findActiveProcessManagers* processada pelo *SGC*.

No processamento da mensagem *findActiveProcessManagers*, o *gerente central* apenas reúne em uma lista *List* os identificadores dos *gerentes de*

processos ativos no sistema e nele registrados, e envia esta lista ao *gerente de processos* que a solicitou.

O Quadro 59 resume como o *Controlador* de um *gerente de processos* processa a mensagem *activeProcessManagersListFound*.

$activeProcessManagersListFound(idGerenteCentral, idGerenteProcessos, Controlador, List)$ - armazene localmente a lista *List* dos *gerentes de processos* ativos no sistema, para posterior consulta para a delegação de processos

Quadro 59 − Mensagem *activeProcessManagersListFound* processada pelo *Controlador*.

Observe que a lista armazenada localmente por um *gerente de processos* pode se tornar desatualizada com o passar o tempo, sendo necessária uma nova consulta ao *SGC*. Para isso, assumimos que esta lista é atualizada em um período de tempo internamente definido pelo *SGC*.

Determinado o *gerente de processos B* para o qual a delegação vai ser realizada, o *gerente de processos A* envia para *B* a mensagem *runProcess*.

No entanto, a requisição da inicialização de um processo por parte do *Controlador* pode acontecer não apenas por meio de delegação, mas também pela ativação de um *trigger* relativo a um construtor de controle. Neste caso, o *Controlador* do *gerente de processos A*, onde o *trigger* foi ativado, recebe a partir deste *trigger* uma mensagem *initiateProcess* com o processo a ser executado e o identificador da sua superinstância. Observe que a mensagem enviada pelo *Gerente_Triggers_Ativos* em decorrência da ativação do *trigger* de inicialização não é mais *initiate*, mas sim *initiateProcess*, justamente para permitir a delegação de processos.

O Quadro 60 resume como o *Controlador* de um *gerente de processos* processa as mensagens *runProcess* e *initiateProcess*.

$runProcess(idGerenteProcessosSuperInstance, idGerenteProcessos, Controlador, p, superInstance)$ - processe $initiate(p, superInstance, idGerenteProcessosSuperInstance, \text{“}null\text{”}, null, null, null, null)$ $initiateProcess(p, superInstance)$ - se for delegar p (caso permitido), envie a mensagem $runProcess(idGerenteProcessosA, idGerenteProcessosB,$

$Controlador, p, superInstance$), onde $idGerenteProcessosA$ é o identificador do *gerente de processos* que está executando a mensagem e $idGerenteProcessosB$ é o identificador do *gerente de processos* para o qual a delegação será feita - caso contrário, processe $initiate(p, superInstance,$ $idGerenteProcessosSuperInstance$,"$null$"$, null, null, null, null)$

Quadro 60 − Mensagens *runProcess* e *initiateProcess* processadas pelo *Controlador*.

No processamento da mensagem *runProcess*, o *gerente de processos* inicia a tarefa de execução do processo indicado.

Ao receber uma mensagem *initiateProcess* para um processo p, vinda do *Gerente_Triggers_Ativos* pela ativação de um *trigger* de um determinado construtor de controle ou do *trigger* de inicialização de processos não-inicializados, o *Controlador* de um gerente A decide se deseja ou não delegar p para outro gerente B, caso o valor da propriedade *p-ext:canBeDelegated* indique que a delegação é permitida para p. Caso decida (e possa) delegar, A envia para B a mensagem *runProcess*. Caso contrário, A inicializa uma instância para p, executando a mensagem *initiate*.

A Figura 8.6 apresenta um diagrama de seqüência mostrando a troca de mensagens entre os componentes do sistema durante a delegação de um processo à execução por um determinado *gerente de processos* (o subordinado, na figura).

Observe que todas as mensagens enviadas pelos componentes no sistema são capturadas pelo SRM do *gerente central* e reencaminhadas para o destinatário correto.

Cada vez que o *Controlador* de um *gerente de processos* cria uma instância de processo P, durante o processamento da mensagem *initiate*, este subcomponente envia a mensagem *insertAndUpdateRecords* endereçada ao SGI, contendo:

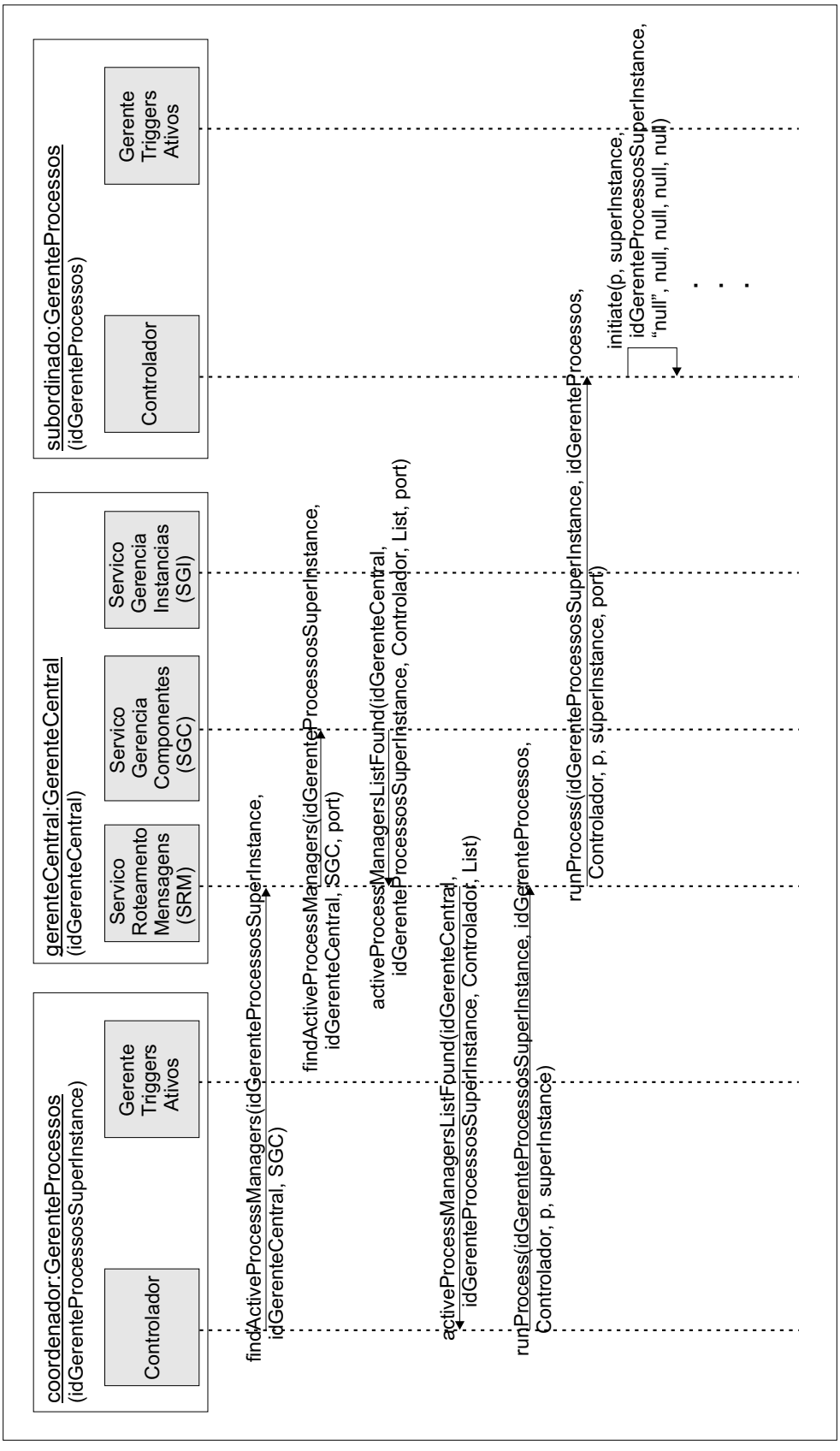


Figura 8.6: Diagrama de seqüência relativo à delegação de um processo.

- o processo p e a identificação da instância P correspondente criada, a serem colocados nas entradas *process* e $instanceId$ do registro, respectivamente;
- o momento em que P foi iniciada, a ser colocado na entrada $initTime$ do registro;
- a identificação de seu *gerente de processos*, a ser colocada na entrada $idGerenteProcessos$;
- a identificação da instância e do *gerente de processos* coordenadores de P, ou seja, da superinstância e de seu *gerente de processos*. Estas informações devem ser colocadas, respectivamente, nas entradas $superInstance$ e $idGerenteProcessosSuperInstance$ do registro;
- a identificação do tipo da instância, a ser colocada na entrada $instanceType$;
- a identificação da instância original, ou seja, da instância de processo que sofreu exceção e para a qual a instância corrente está sendo executada, se esta instância for de flexibilização. Esta informação deve ser colocada na entrada *pointer_original_p*;
- a identificação do processo abstrato, *abstract_p*, no caso da instância ser do tipo "*concretização*";
- informação de mapeamento de parâmetros, caso a instância correspondente seja uma instância de processo concreto encontrado a partir de um determinado processo abstrato ou uma instância de substituição (tanto no caso de substituição de processos propriamente dita quanto no caso de tratamento de exceção temporal). Esta informação deve ser colocada na entrada *parameters_mapping*;
- informação de mapeamento de recursos, caso a instância correspondente seja uma instância de processo concreto encontrado a partir de um determinado processo abstrato (informação $resources\_mapping$).

O Quadro 61 resume como o SGI processa a mensagem *insertAndUpdateRecords*.

$insertAndUpdateRecords(idGerenteProcessos, idGerenteCentral, SGI, p, P, initTime, superInstance, idGerenteProcessosSuperInstance, instanceType, pointer\_original\_p, abstract\_p, parameters\_mapping, resources\_mapping)$

- insira um registro no banco de dados de instâncias referente à instância P do processo p. Associe a ele os demais valores informados ($idGerenteProcessos, initTime, superInstance$, $idGerenteProcessosSuperInstance, instanceType, pointer\_original\_p$). As informações $abstract\_p$ e $resources\_mapping$ apenas são informadas quando a instância P é do tipo "concretização". A informação $parameters\_mapping$ é apenas informada quando a instância é de algum dos seguintes tipos: "*temporal*", "*substituição*" ou "*concretização*". Valores não informados são armazenados como *null*
- se $pointer\_original\_p \neq null$ e $instanceType \neq$ "*undo*", atualize a entrada $\{subInstances\}$ da superinstância $superInstance$ de P, colocando uma referência para P, junto à referência para P', referenciada em $pointer\_original\_p$, formando o par (P', P), indicando que a superinstância deve agora esperar pelo fim da instância P e não pelo fim de P'
- se $instanceType \neq$ "*undo*", envie a mensagem $updateSubInstances(idGerenteCentral$, $idGerenteProcessosSuperInstance, Controlador, superInstance, P$, $pointer\_original\_p)$
- registre na entrada $pointer\_flex\_p$ de P' uma referência para P

Quadro 61 − Mensagem *insertAndUpdateRecords* processada pelo SGI.

Além de inserir uma nova instância P no banco de dados de instâncias, ao processar a mensagem *insertAndUpdateRecords*, o SGI atualiza os registros das outras instâncias afetadas pela criação de P. No Capítulo 7, não era necessário um serviço de gerência de instâncias porque todas elas estavam sendo processadas na mesma MAE. No entanto, como no ambiente distribuído as instâncias estão espalhadas em vários *gerentes de processos*, o SGI muitas vezes precisa também solicitar a atualização de instâncias aos *gerentes de processos* competentes, devido a uma flexibilização sofrida por uma determinada instância.

O Quadro 62 resume como o *Controlador* processa a mensagem *updateSubInstances* enviada pelo SGI, para que uma instância $superInstance$ tenha atualizada a sua lista de subinstâncias, para o correto término de sua execução.

$updateSubInstances(idGerenteCentral, idGerenteProcessosSuperInstance, Controlador, superInstance, P, P')$ - registre na entrada $\{subInstances\}$ da $superInstance$ o par (P', P), substituindo o par existente em que P' era o primeiro elemento

Quadro 62 − Mensagem *updateSubInstances* processada pelo *Controlador*.

No processamento da mensagem *updateSubInstances*, o *Controlador* então registra da entrada $\{subInstances\}$ da instância $superInstance$ que P' não é mais considerada sua subinstância, mas sim o é a instância criada P.

Quando uma instância P muda de estado, o *Controlador* do *gerente de processos* onde P está sendo executada envia a mensagem *updateStateInRecord* ao SGI.

O Quadro 63 resume como o SGI processa essa mensagem.

$updateStateInRecord(idGerenteProcessos, idGerenteCentral, SGI, P, state, timestamp)$ - busque no banco de dados de instâncias o registro correspondente à instância P em $idGerenteProcessos$ e atualize neste registro a entrada $(state, timestamp)$

Quadro 63 − Mensagem *updateStateInRecord* processada pelo SGI.

Para atualizar os registros das instâncias de processo no SGI com relação à substituição ou concretização de recursos, o *Controlador* envia ao SGI a mensagem *updateRecordDueToResourcesFlexibilization*.

O Quadro 64 resume como o SGI processa essa mensagem.

$updateRecordDueToResourcesFlexibilization(idGerenteProcessos, idGerenteCentral, SGI, P, \{(original\_r, substitute\_r)\}, \{(abstract\_r, concrete\_r)\})$ - busque no banco de dados de instâncias o registro correspondente à instância P em $idGerenteProcessos$ e atualize neste registro as entradas $\{(original\_r, substitute\_r)\}$ e $\{(abstract\_r, concrete\_r)\}$, quando informadas não nulas

Quadro 64 − Mensagem *updateRecordDueToResourcesFlexibilization* processada pelo SGI.

Para a arquitetura distribuída, as mudanças de estados de uma instância ocorrem de acordo com o mesmo diagrama de transição de estados da Figura 7.6, apresentada no Capítulo 7.

O Apêndice A apresenta em detalhe as mensagens que precisaram ser alteradas para que uma instância de processo possa ser executada no ambiente distribuído descrito.

8.5 Resumo

Este capítulo apresentou uma arquitetura distribuída para um sistema de workflow, incluindo o mecanismo de tratamento de exceção proposto nesta tese, estendendo a discussão do capítulo anterior ao considerar a questão da execução distribuída de processos.

Basicamente, a arquitetura consiste de três tipos de componentes: *GerenteCentral*, *GerenteOntologias* e *GerenteProcessos*. O *gerente central* do sistema é responsável por manter todo o sistema em funcionamento, provendo os seguintes serviços: roteamento de mensagens, gerência de componentes e gerência de instâncias. O *gerente de ontologias* do sistema tem por responsabilidade armazenar a ontologia *pr*, as bibliotecas *lib* associadas aos processos e recursos definidos no sistema e as ontologias de aplicação, e implementar o mecanismo proposto de tratamento de exceção para a flexibilização, oferecendo-o como um serviço aos *gerentes de processos* no sistema. Um *gerente de processos* é responsável por controlar a execução de uma ou mais instâncias de processos, e delegar um subprocesso para outro *gerente de processos*, caso isso seja permitido. Essa delegação de processos, comum em um ambiente distribuído, não tinha semântica na máquina abstrata, na qual todas as informações de execução estavam contidas.

A arquitetura proposta foi definida em ACME, através do uso do conceito de *família*. A família *FamiliaExecucaoProcessos* criada possui tipos de componentes, conectores, portas e papéis, além das regras de restrições lógicas.