# 7
# Semântica Operacional de OWL-S e suas Extensões

Este capítulo apresenta uma semântica operacional para OWL-S, incluindo as extensões propostas, definida sob forma de uma máquina abstrata (MA). Esta é uma opção atrativa por dois motivos:

1. a máquina abstrata é de fácil entendimento, utilizando apenas conceitos simples;

2. a definição pode ser feita de forma incremental, através de sucessivas extensões para acomodar construtores mais sofisticados.

A Seção 7.1 introduz os principais conceitos da máquina abstrata. A Seção 7.2 descreve como a máquina abstrata interpreta os processos atômicos e os principais construtores básicos de OWL-S. A Seção 7.3 apresenta a máquina abstrata modificada para atender às extensões de OWL-S propostas. A Seção 7.4 apresenta uma descrição das transições de estado de uma instância de processo na MAE. A Seção 7.5 descreve como a máquina abstrata estendida (MAE) executa as extensões básicas propostas à linguagem OWL-S, enquanto que a Seção 7.6 discute o modo como a MAE acomoda a flexibilização da execução. A Seção 7.7 aborda o log de execução de uma instância de processo. Por fim, a Seção 7.8 discute o teste de consistência realizado sobre uma instância de processo e a Seção 7.9 apresenta algumas situações de execução de instâncias de processos na MAE. A Seção 7.10 apresenta um resumo deste capítulo.

A organização deste capítulo reflete a organização do Capítulo 6, tomando como ponto de partida os construtores básicos de OWL-S e, posteriormente, as extensões propostas.

PUC-Rio - Certificação Digital Nº 0115612/CA

PUC-Rio - Certificação Digital Nº 0115612/CA

## 7.1 Definição da Máquina Abstrata para OWL-S

Esta seção descreve os conceitos básicos da máquina abstrata para OWL-S, incluindo alguns conceitos preliminares.

### 7.1.1 Componentes da Máquina Abstrata

A máquina abstrata, ou abreviadamente MA, possui a arquitetura apresentada na Figura 7.1.

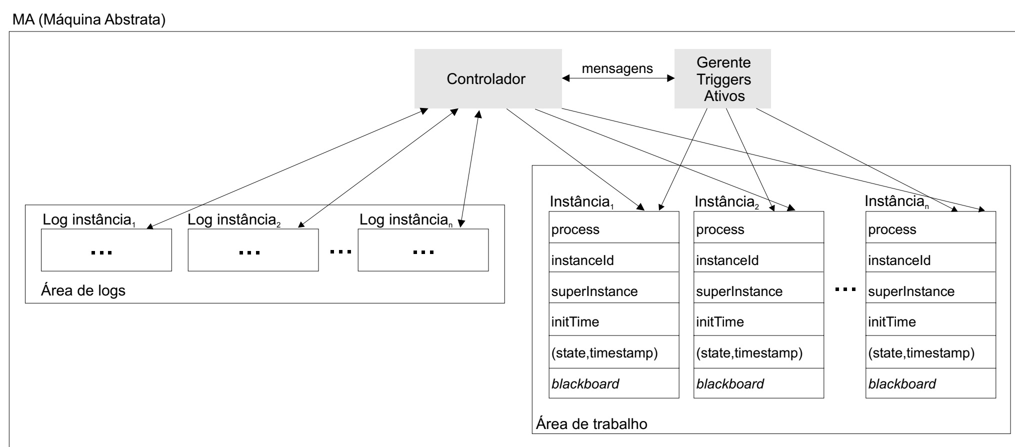


Figura 7.1: Máquina abstrata (MA).

Para a MA, uma *instância* $P$ de um processo $p$ compõe-se de:

- uma representação da definição do processo $p$ correspondente;
- um identificador que a identifica unicamente dentro da máquina abstrata – entrada $instanceId$ na Figura 7.1;
- um identificador da instância de seu superprocesso (veja definição a seguir) – entrada $superInstance$ da Figura 7.1;
- o *timestamp* indicando o momento em que a instância foi criada – entrada $initTime$ na Figura 7.1.
- um estado corrente, denotado por $\sigma[P]$, juntamente com o *timestamp* a ele associado – entrada $(state, timestamp)$ na Figura 7.1;
- um conjunto de atribuição de valores para os parâmetros definidos em $p$, chamado de *blackboard* de $P$ e denotado por $\beta[P]$ – entrada *blackboard* na Figura 7.1.

PUC-Rio - Certificação Digital Nº 0115612/CA

Cada instância $P$ de um processo $p$ é mantida na *área de trabalho* da MA, esteja $P$ aberta ou fechada (veja Figura 7.2). Note que a representação de uma instância fechada poderia ser simplificada através de um mecanismo de otimização que eliminasse todos os dados não mais necessários como, por exemplo, parâmetros de saída já consumidos por outras instâncias. Esta discussão está fora do escopo deste capítulo, que trata apenas da especificação da semântica operacional de OWL-S.

A MA também possui uma *área de logs*, onde armazena o log de execução de cada uma das instâncias de processo. Uma explicação sobre o log pode ser encontrada na Seção 7.7.

A MA consiste, ainda, de um *Controlador*, responsável pela gerência das instâncias, e de um *Gerente de Triggers Ativos*, que controla os *triggers*.

Os *triggers*[1] capturam a semântica dos processos atômicos e dos compostos, conforme seus construtores de controle, e guiam as transições de estado durante o processamento de uma instância de processo. A ativação de cada *trigger* provoca o envio, a partir do *Gerente de Triggers Ativos* ao *Controlador*, de um conjunto de mensagens que determinam o processamento da instância. Por fim, a MA não inclui explicitamente um componente para gerenciar a alocação e liberação de recursos. Esta questão será tratada em detalhe na Seção 7.3, no contexto das estratégias de flexibilização.

### 7.1.2
### Execução de uma Instância de Processo na MA

A execução de uma instância $P$ de um processo $p$ atravessa uma seqüência de estados, conforme o diagrama de transição de estados da Figura 7.2. Uma instância de processo está *fechada* (*closed*) se, e somente se, o seu estado é *Completed* ou *Aborted*, e ela está *aberta* (*open*) se, e somente se, o seu estado é *Initiated*, *Running* ou *PreparedToComplete*.

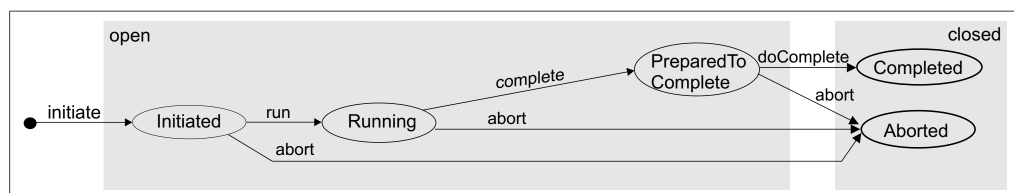


Figura 7.2: Diagrama de transição de estados definido para uma instância $P$ de um processo $p$ (OWL-S não estendido) na MA.

[1]O termo *trigger* será mantido em inglês. Gatilho é uma possível tradução.

As transições de estado nesse diagrama não são definidas por funções, como nos autômatos tradicionais, pois ocorrem estimuladas por: (i) eventos externos, tais como a interrupção de uma instância de processo ou a finalização de uma instância de um subprocesso, ou (ii) decisões de controle inerentes à estrutura do processo, como a "navegação" para o próximo subprocesso dentro de um processo composto. As mudanças de estado de $P$ são definidas por *triggers* que caracterizam a semântica dos processos atômicos e dos processos compostos. Os tipos de *triggers* definidos são os seguintes:

- *InstanceInit*["*T*"]($p$): *trigger* que é ativado quando existe um processo $p$ ainda não inicializado na lista de processos da MA. Esta lista contém os processos que devem ter sua execução disparada explicitamente pela MA;
- *InstanceInitiated*[$n$][$T$]($P$): *trigger* associado ao estado *Initiated* de $P$;
- *InstanceRunning*[$n$][$T$]($P$): *trigger* associado ao estado *Running* de $P$.

PUC-Rio - Certificação Digital Nº 0115612/CA

onde $n \geq 0$ e $T$, nos *triggers InstanceInitiated* e *InstanceRunning*, é o nome do construtor de $p$, se $p$ for composto, e $T$ = "*Atomic*", se $p$ for atômico, onde $p$ é o processo do qual $P$ é uma instância. No *trigger InstanceInit*, "*T*" determina que o *trigger* é válido para qualquer processo/instância, seja ele composto(a), independentemente de qual for o construtor de controle correspondente, ou atômico(a). As notações *InstanceInitiated*[$n$] e *InstanceRunning*[$n$] são apenas convenções para permitir a definição de mais de um *trigger* associado aos estados *Initiated* e *Running*.

Para caracterizar um comportamento transacional de uma instância de processo, os *triggers* devem ser construídos de tal forma a garantir as seguintes propriedades, onde $P$ é uma instância de processo composto:

**Propriedade 1:** se $P$ terminou no estado *Completed*, então todas as subinstâncias diretas de $P$ terminaram no estado *Completed*;

**Propriedade 2:** se $P$ terminou no estado *Aborted*, então todas as subinstâncias diretas de $P$ terminaram no estado *Aborted*.

Estas propriedades serão redefinidas para a máquina abstrata estendida (MAE), incorporando os mecanismos de tratamento de exceção para flexibilização. Por hora, observe que a Propriedade 1 implica imediatamente que, se $P$ terminou no estado *Completed*, então todas as subinstâncias diretas e indiretas de $P$ terminaram no estado *Completed*, pela característica transacional da execução de instâncias de processos.

Da mesma forma, a Propriedade 2 implica que, se $P$ terminou no estado *Aborted*, então todas as subinstâncias diretas e indiretas de $P$ terminaram no estado *Aborted*.

Vale ressaltar novamente que, por comportamento transacional, entendemos a propriedade garantindo que ou todas as ações de uma instância terminam corretamente, ou todas são abandonadas e identificadas como tal.

Naturalmente, como a máquina abstrata não tem controle estrito sobre as ações atômicas, ela não desfaz automaticamente os efeitos das ações abandonadas, como em um sistema de gerência de banco de dados. Apenas nos casos em que o projetista previu de antemão, a máquina abstrata estendida poderá de fato desfazer os efeitos de uma ação atômica, conforme veremos na Seção 7.3.

A definição do *trigger* *InstanceInit*[“*T*”]($p$), que garante a inicialização de instâncias de processos que não são componentes de processos em execução, é apresentada no Quadro 28.

*InstanceInit*[“*T*”]($p$)

entrada:

- um processo $p$, atômico ou composto, independentemente de construtor de controle

condição de disparo:

- $p$ pertence à lista de processos da MA ainda não inicializados
- pré-condições de $p$ são verdadeiras

corpo:

- envie a mensagem $initiate(p, null)$ ao *Controlador*

Quadro 28 – *Trigger InstanceInit*[“*T*”]($p$).

A definição dos *triggers* *InstanceInitiated*[$n$][$T$]($P$) e *InstanceRunning*[$n$][$T$]($P$) reflete a semântica de $T$ e será detalhada nas seções seguintes. De maneira geral, temos que:

- os *triggers*, através de suas condições de disparo, podem observar os estados correntes, os *blackboards* e os tempos de início ($initTime$) das instâncias de processos que estão na MA;

PUC-Rio - Certificação Digital Nº 0115612/CA

– os *triggers* comunicam-se com o *Controlador* através de 5 tipos de mensagens: *initiate*, *run*, *complete*, *doComplete* e *abort*. A mensagem *initiate* possui como parâmetros uma definição de processo $p$ e uma referência para a superinstância $S$ da instância que será criada a partir de $p$. As demais mensagens possuem como parâmetro apenas uma instância $P$ de um processo $p$.

O *Controlador* processa as mensagens recebidas dos *triggers* conforme indicado no Quadro 29. Observe que o *Controlador* também atualiza o log associado a uma instância à medida em que processa a instância. A descrição do log pode ser encontrada na Seção 7.7.

$initiate(p, S)$
- crie uma instância $P$ para $p$
- inicialize $superInstance$ de $P$ com $S$
- inicialize *initTime* com o *timestamp* em que a instância foi criada
- inicialize o *blackboard* $\beta[P]$
- coloque $P$ no estado *Initiated*
- aloque os recursos necessários a $P$
- registre o novo estado de $P$, atualizando o par $(state, timestamp)$
- registre o novo estado no log de $P$ (veja Seção 7.7), através do registro $newState(state, timestamp)$

$run(P)$
- coloque $P$ no estado *Running*
- registre o novo estado de $P$, atualizando o par $(state, timestamp)$
- registre o novo estado no log de $P$, através do registro $newState(state, timestamp)$
- registre os valores dos parâmetros de entrada no log de $P$, através do registro $inputParameters(\{parameter, value\})$

$complete(P)$
- coloque $P$ no estado *PreparedToComplete*
- desaloque os recursos usados por $P$
- registre o novo estado de $P$, atualizando o par $(state, timestamp)$
- registre o novo estado no log de $P$, através do registro $newState(state, timestamp)$
- registre os valores dos parâmetros de saída no log de $P$, através do registro $outputParameters(\{parameter, value\})$

PUC-Rio - Certificação Digital Nº 0115612/CA

$doComplete(P)$
- coloque $P$ no estado *Completed*
- registre o novo estado de $P$, atualizando o par $(state, timestamp)$
- registre o novo estado no log de $P$, através do registro $newState(state, timestamp)$
- para cada subinstância $P_i$ de $P$, processe $doComplete(P_i)$

$abort(P)$
- coloque $P$ no estado *Aborted*
- desaloque os recursos usados por $P$
- registre o novo estado de $P$, atualizando o par $(state, timestamp)$
- registre o novo estado no log de $P$, através do registro $newState(state, timestamp)$
- para cada subinstância $P_i$ de $P$, processe $abort(P_i)$

Quadro 29 – Mensagens recebidas dos *triggers* e processadas pelo *Controlador* da MA.

PUC-Rio - Certificação Digital Nº 0115612/CA

Observe o comportamento transacional, por exemplo, na mensagem *doComplete*. Quando esta mensagem é executada para uma determinada instância $P$ de processo composto $p$, é porque esta instância encontra-se no estado *PreparedToComplete*.

Vale ressaltar que, quando dizemos que o *Controlador* processa uma mensagem, queremos dizer que o *Controlador* executa todos os passos associados à mensagem, como se ele executasse uma subrotina definida a priori. Por exemplo, no processamento da mensagem $abort(P)$, o *Controlador* processa $abort(P_i)$, para cada $P_i$ subinstância de $P$. Isso significa que, para cada subinstância, ele executa os passos associados à mensagem *abort*.

Ainda, se $P$ é uma instância de um processo $p$, dizemos que uma instância $P'$ de *p'* *foi criada* a partir de $P$ se, e somente se, o *Controlador* criou $P'$ ao processar a mensagem $initiate(p', P')$, gerada por um *trigger* associado a $P$.

O Quadro 30 apresenta os *triggers* necessários para a correta terminação de uma instância de processo, seguindo o comportamento transacional.

$InstancePreparedToComplete[1][T](P)$

entrada:
- uma instância $P$ de um processo $p$

condição de disparo:
- $superInstance$ de $P$ é $null$
- $\sigma[P] = PreparedToComplete$
- para todas as subinstâncias $P_i$, diretas ou indiretas de $P$, $\sigma[P_i] = PreparedToComplete$

corpo:
- envie a mensagem $doComplete(P)$ ao $Controlador$

---

$InstancePreparedToComplete[2][T](P)$

entrada:
- uma instância $P$ de um processo $p$

condição de disparo:
- $superInstance$ de $P$ é $null$
- $\sigma[P] = PreparedToComplete$
- nem todas as subinstâncias $P_i$, diretas ou indiretas de $P$, são tais que $\sigma[P_i] = PreparedToComplete$

corpo:
- envie a mensagem $abort(P)$ ao $Controlador$

Quadro 30 − *Triggers* que controlam a terminação de instâncias na MA.

Os documentos de OWL-S não definem claramente qual a semântica de um processo com múltiplas pré-condições. Neste caso, assumimos que, se existem diversas pré-condições definidas, todas elas devem ser verdadeiras para que o processo possa ser executado.

As pré-condições de um processo podem ser definidas através de: (1) valores constantes; (2) funções; (3) referências a instâncias de classes OWL da mesma hierarquia de tipos do parâmetro correspondente; ou (4) parâmetros de outros processos.

Quando as pré-condições são definidas através de construções como

PUC-Rio - Certificação Digital Nº 0115612/CA

nos três primeiros casos, a sua análise é trivial. Quando são definidas sobre valores de parâmetros de outros processos, OWL-S utiliza a variável *TheParentPerform*, que aponta para uma particular execução de um processo definido, via *Perform*, como componente de um processo composto. No entanto, *TheParentPerform* por si só não é suficiente porque podem existir simultaneamente, na máquina abstrata, mais do que uma instância de um mesmo processo, sobre uma mesma hierarquia de instâncias e, por isso, é preciso que o *Controlador* saiba em qual delas buscar o valor dos parâmetros dos quais necessita para a execução de uma instância. Este é o caso que ocorre quando se utiliza o construtor *ForAll*, que dispara simultaneamente mais de uma instância de um mesmo processo.

Dessa forma, assumimos que é necessário que a MA capture, no caso de haver mais do que uma instância do mesmo processo, a instância que terminou mais recentemente sua execução.

Para isso, cada instância guarda, além do estado corrente, o *timestamp* em que ele foi alcançado. Desta forma, o *Controlador* é capaz de encontrar a instância de um determinado processo $p'$ que alcançou o estado *PreparedToComplete* mais recentemente, e recuperar do seu *blackboard* os valores dos parâmetros dos quais necessita. Conforme mencionado na Seção 4.1.3, como a questão de fluxo de dados em OWL-S não é bem definida, assumimos que, se um processo $p$ recebe valores de entrada provenientes de parâmetros de um outro processo $p'$, então $p$ e $p'$ possuem um superprocesso em comum, ou seja, pertencem à mesma hierarquia de processos. Assim, esta é uma característica importante do modelo de execução de workflow adotado, uma vez que determina que, se parâmetros são passados de uma instância de processo para outra, então estas duas instâncias fazem parte de uma mesma instância.

Dessa forma, os seguintes passos são executados pelo *Controlador* para poder encontrar valores de parâmetros para uma determinada instância $P$:

– encontrar a instância $P'$, relativa ao processo $p'$ a partir do qual é obtido o valor de um parâmetro de entrada em $P$, que mais recentemente alcançou o estado *PreparedToComplete*;
– buscar no *blackboard* de $P'$ o valor deste parâmetro.

Em resumo, as hipóteses feitas são:

– pré-condições de processos podem ser explícita ou implicitamente definidas. São pré-condições implícitas para a execução de um determinado processo a disponibilidade dos recursos dos quais

PUC-Rio - Certificação Digital Nº 0115612/CA

necessita para executar e o conhecimento dos valores dos parâmetros de entrada;

- todas as pré-condições para que um processo possa ser inicializado devem ser avaliadas como verdadeiras;
- se são várias as instâncias de um mesmo processo $p$, é considerada a instância que mais recentemente alcançou o estado *PreparedToComplete* para a recuperação de valores de parâmetros;
- se o valor de entrada de um processo $p$ é oriundo de um processo $p'$, então $p$ e $p'$ possuem um superprocesso $p''$ em comum;
- a execução de qualquer instância é caracterizada por um comportamento transacional, de acordo com as propriedades apresentadas na página 150.

## 7.2 Sintaxe Abstrata e Semântica de Processos Atômicos e de Construtores Básicos de OWL-S

Esta seção apresenta uma proposta para a semântica de processos atômicos e de alguns dos construtores da linguagem OWL-S, de acordo com a máquina abstrata descrita. Para facilitar a discussão, a seção introduz ainda uma sintaxe abstrata. Vale ressaltar que a descrição não deve ser interpretada como uma sugestão de implementação.

No que se segue, a variável sintática $T$ varre os valores "*Atomic*", "*SEQUENCE*", "*SPLIT*", "*SPLIT-JOIN*", "*CHOICE*", "*ANY-ORDER*", "*IF-THEN-ELSE*".

### 7.2.1 Processos Atômicos

A execução de processos atômicos procede de acordo com o diagrama de estados apresentado na Figura 7.2 (página 149).

Um processo atômico $p$ pode ser inicializado a partir do *trigger* *InstanceInit*["*T*"]($p$), quando $p$ encontra-se na lista de processos não inicializados da MA, ou a partir da execução de seu superprocesso. Neste último caso, o momento em que um processo é inicializado depende do construtor de controle, como será visto adiante.

A instância $P$ do processo $p$ alcança o estado *Initiated* quando o *Controlador* processa a mensagem *initiate* para $p$.

PUC-Rio - Certificação Digital Nº 0115612/CA

Do estado *Initiated*, $P$ passa para o estado *Running* quando todas as suas pré-condições são verdadeiras, conforme definido pelo *trigger* *InstanceInitiated*[1][“*Atomic*”]($P$) apresentado no Quadro 31.

Do estado *Running*, $P$ pode alcançar dois estados: *PreparedToComplete* e *Aborted*. $P$ alcança o estado *PreparedToComplete* quando seu processamento é encerrado com sucesso, conforme definido pelo *trigger* *InstanceRunning*[1][“*Atomic*”]($P$), apresentado no Quadro 31.

Existem, por outro lado, duas formas distintas de $P$ alcançar o estado *Aborted*. Na primeira, o *trigger* *InstanceRunning*[2][“*Atomic*”]($P$) é ativado, pela ocorrência de alguma falha em $P$, provocando o envio da mensagem *abort*($P$) ao *Controlador*. Na segunda, $P$ é colocada pelo *Controlador* no estado *Aborted* quando a instância $Q$ do seu superprocesso é abortada, por exemplo, pelo fato de uma das subinstâncias de $Q$ ter abortado.

Do estado *PreparedToComplete* uma instância $P$ de processo atômico pode alcançar dois estados: *Aborted* e *Completed*. $P$ alcança o estado *Completed* se as superinstâncias diretas ou indiretas também alcançaram o estado *Completed*. Por outro lado, $P$ alcança o estado *Aborted*, a partir do estado *PreparedToComplete*, se sua superinstância também alcançou este estado.

O Quadro 31 apresenta os *triggers* definidos para processo atômico e instância de processo atômico.

*InstanceInitiated*[1][“*Atomic*”]($P$)

entrada:
- uma instância $P$ de um processo atômico $p$

condição de disparo:
- $\sigma[P] = Initiated$
- todas as pré-condições de $P$ são verdadeiras

corpo:
- envie a mensagem *run*($P$) ao *Controlador*

---

*InstanceRunning*[1][“*Atomic*”]($P$)

entrada:
- uma instância $P$ de um processo atômico $p$

PUC-Rio - Certificação Digital Nº 0115612/CA

condição de disparo:
- $\sigma[P] = Running$
- processamento de $P$ terminou com sucesso

corpo:
- envie a mensagem $complete(P)$ ao *Controlador*

---

$InstanceRunning[2][\text{“}Atomic\text{”}](P)$

entrada:
- uma instância $P$ de um processo atômico $p$

condição de disparo:
- $\sigma[P] = Running$
- ocorreu algum problema na execução de $P$

corpo:
- envie a mensagem $abort(P)$ ao *Controlador*

Quadro 31 – *Triggers* para processo atômico e instância de processo atômico.

PUC-Rio - Certificação Digital Nº 0115612/CA

### 7.2.2 Sequence

Uma composição por *sequence* possui a sintaxe abstrata $p_c = p_1; p_2; ...; p_n$ , onde $p_i$ são processos, para $i \in [1, n]$. Dizemos que $p_i$ são *componentes* de $p_c$.

A composição $p_c$ determina que:

- o processamento de $p_1$ inicie logo que o processamento de $p_c$ iniciar;
- o processamento de $p_i$ inicie após o processamento de $p_{i-1}$ terminar, para $i \in [2, n]$;
- o processamento de $p_c$ termine quando o processamento de $p_n$ terminar.

Assim, o processamento de uma instância $P_c$ de uma composição seqüencial $p_c = p_1; p_2; ...; p_n$ de $n$ processos ocorre da seguinte forma:

– quando $P_c$ alcança o estado *Running*, uma instância $P_1$ de $p_1$ é criada e colocada no estado *Initiated*;

– quando $P_{i-1}$ alcança o estado *PreparedToComplete*, uma instância $P_i$ de $p_i$ é criada e colocada no estado *Initiated*, para $i \in [2, n]$;

– quando $P_n$ alcança o estado *PreparedToComplete*, a instância $P_c$ é colocada no estado *PreparedToComplete*;

– quando alguma $P_i$ alcança o estado *Aborted*, $P_c$ é colocada no estado *Aborted*, para $i \in [1, n]$.

Observe que quando $P_c$ alcança o estado *Aborted* porque uma instância $P_i$, para $i \in [1, n]$, alcançou o estado *Aborted*, todas as demais instâncias $P_i$ também serão colocadas neste estado, conforme as propriedades que determinam o comportamento transacional das instâncias de processo.

Esse comportamento é especificado através da definição dos *triggers* apresentados no Quadro 32.

*InstanceInitiated*[1][“*SEQUENCE*”]($P_c$)

entrada:

- uma instância $P_c$ de uma composição seqüencial $p_c = p_1; p_2; ...; p_n$

condição de disparo:

- $\sigma[P_c] =$ *Initiated*
- todas as pré-condições de $P_c$ são verdadeiras

corpo:

- envie a mensagem *run*($P_c$) ao *Controlador*
- envie a mensagem *initiate*($p_1, P_c$) ao *Controlador*
- crie no *blackboard* $\beta[P_c]$ de $P_c$ a variável interna *last*($P_c$) para representar o índice do último subprocesso inicializado, e inicialize-a com o valor 1

---

*InstanceRunning*[1][“*SEQUENCE*”]($P_c$)

entrada:

- uma instância $P_c$ de uma composição seqüencial $p_c = p_1; p_2; ...; p_n$

PUC-Rio - Certificação Digital Nº 0115612/CA

condição de disparo:

- $\sigma[P_c] = Running$
- $\sigma[P_{last(P_c)}] = PreparedToComplete$, onde $last(P_c) < n$

corpo:

- incremente a variável interna $last(P_c)$ de 1
- envie a mensagem $initiate(p_{last(P_c)}, P_c)$ ao *Controlador*

---

$InstanceRunning[2][\text{“}SEQUENCE\text{”}](P_c)$

entrada:

- uma instância $P_c$ de uma composição seqüencial $p_c = p_1; p_2; ...; p_n$

condição de disparo:

- $\sigma[P_c] = Running$
- $\sigma[P_n] = PreparedToComplete$

corpo:

- envie a mensagem $complete(P_c)$ ao *Controlador*

---

$InstanceRunning[3][\text{“}SEQUENCE\text{”}](P_c)$

entrada:

- uma instância $P_c$ de uma composição seqüencial $p_c = p_1; p_2; ...; p_n$

condição de disparo:

- $\sigma[P_c] = Running$
- $\sigma[P_{last(P_c)}] = Aborted$

corpo:

- envie a mensagem $abort(P_c)$ ao *Controlador*

Quadro 32 – *Triggers* que definem o comportamento do construtor *SEQUENCE*.

PUC-Rio - Certificação Digital Nº 0115612/CA

### 7.2.3 Split

A composição por *split* possui a sintaxe abstrata $p_c = p_1 \backslash p_2 \backslash ... \backslash p_n$, onde $p_i$ são processos, para $i \in [1, n]$. Dizemos que $p_i$ são *componentes* de $p_c$.

A composição $p_c$ determina que o *thread* de controle de $p_c$ se divida em $n$ *threads* para execução paralela de $p_1, p_2, ..., p_n$. Mais precisamente:

- quando $P_c$ alcança o estado *Running*, as instâncias $P_1, P_2, ..., P_n$ de $p_1, p_2, ...p_n$ são criadas e colocadas no estado *Initiated*;
- logo após a inicialização de todas as instâncias, a própria instância $P_c$ é colocada no estado *PreparedToComplete*.

Repare que as instâncias $P_1$, $P_2$, ..., $P_n$ são processadas independentemente de $P_c$, sem alterar o processamento de $P_c$. Inclusive, se alguma abortar, $P_c$ não é abortado.

Esse comportamento não fere a Propriedade 2 apresentada na página 150, pois um componente de um processo $p$ definido por *Split* não é considerado um subprocesso de $p$, já que tem sua execução realizada independentemente.

O comportamento do construtor *Split* é especificado através da definição dos *triggers* mostrados no Quadro 33.

$InstanceInitiated[1][\text{“}SPLIT\text{”}](P_c)$

entrada:

- uma instância $P_c$ de uma composição por *split* $p_c = p_1 \backslash p_2 \backslash ... \backslash p_n$

condição de disparo:

- $\sigma[P_c] = Initiated$
- todas as pré-condições de $P_c$ são verdadeiras

corpo:

- envie a mensagem $run(P_c)$ ao *Controlador*
- envie mensagens $initiate(p_1, P_c)$ , $initiate(p_2, P_c)$ , ... , $initiate(p_n, P_c)$ ao *Controlador*

PUC-Rio - Certificação Digital Nº 0115612/CA

$InstanceRunning[1][\text{“}SPLIT\text{”}](P_c)$

entrada:

- uma instância $P_c$ de uma composição por *split* $p_c = p_1 \backslash p_2 \backslash ... \backslash p_n$

condição de disparo:

- $\sigma[P_c] = Running$
- $\sigma[P_i] = Initiated$, para todo $i \in [1, n]$

corpo:

- envie a mensagem $complete(P_c)$ ao *Controlador*

Quadro 33 – *Triggers* que definem o comportamento do construtor *SPLIT*.

PUC-Rio - Certificação Digital Nº 0115612/CA

### 7.2.4 Split-Join

A composição por *split-join* possui a sintaxe abstrata $p_c = p_1$ // $p_2$ // ... // $p_n$, onde $p_i$ são processos, para $i \in [1, n]$. Dizemos que $p_i$ são *componentes* de $p_c$.

A composição $p_c$ determina que o *thread* de controle de $p_c$ se divida em $n$ *threads* para execução paralela de $p_1, p_2, ..., p_n$. Ao término da execução paralela, o *thread* de controle volta a ser um só. Mais precisamente:

– quando $P_c$ alcança o estado *Running*, instâncias $P_1$ , $P_2$ , ... , $P_n$ de $p_1$, $p_2$, ... $p_n$ são criadas e colocadas no estado *Initiated*;

– quando $P_1$, $P_2$, ..., $P_n$ alcançam todas o estado *PreparedToComplete*, a instância $P_c$ é colocada no estado *PreparedToComplete*;

– se alguma $P_i$, $i \in [1, n]$ alcança o estado *Aborted*, a instância $P_c$ e todas as outras subinstâncias são colocadas no estado *Aborted* (comportamento não especificado em OWL-S).

Esse comportamento é definido através dos *triggers* mostrados no Quadro 34.

PUC-Rio - Certificação Digital Nº 0115612/CA

*InstanceInitiated*[1][“*SPLIT-JOIN*”]($P_c$)

entrada:

- uma instância $P_c$ de uma composição por *split-join*
$p_c = p_1 \ // \ p_2 \ // \ ... \ // \ p_n$

condição de disparo:

- $\sigma[P_c] = Initiated$
- todas as pré-condições de $P_c$ são verdadeiras

corpo:

- envie a mensagem $run(P_c)$ ao *Controlador*
- envie mensagens $initiate(p_1, P_c)$ , $initiate(p_2, P_c)$ , ..., $initiate(p_n, P_c)$ ao *Controlador*

---

*InstanceRunning*[1][“*SPLIT-JOIN*”]($P_c$)

entrada:

- uma instância $P_c$ de uma composição por *split-join*
$p_c = p_1 \ // \ p_2 \ // \ ... \ // \ p_n$

condição de disparo:

- $\sigma[P_c] = Running$
- $\sigma[P_1] = \sigma[P_2] = ... = \sigma[P_n] = PreparedToComplete$

corpo:

- envie a mensagem $complete(P_c)$ ao *Controlador*

---

*InstanceRunning*[2][“*SPLIT-JOIN*”]($P_c$)

entrada:

- uma instância $P_c$ de uma composição por *split-join*
$p_c = p_1 \ // \ p_2 \ // \ ... \ // \ p_n$

condição de disparo:

- $\sigma[P_c] = Running$
- $\sigma[P_i] = Aborted$, para algum $i \in [1, n]$

corpo:
- envie a mensagem $abort(P_c)$ ao *Controlador*

Quadro 34 – *Triggers* que definem o comportamento do construtor *SPLIT-JOIN*.

### 7.2.5 Choice

A composição por *choice* possui a sintaxe abstrata $p_c = p_1 \mid p_2 \mid ... \mid p_n$, onde $p_i$ são processos, para $i \in [1, n]$. Dizemos que $p_i$ são *componentes* de $p_c$.

A composição $p_c$ determina que apenas um único processo $p_i$ do processo composto $p_c$ seja executado. Se a instância do processo $p_i$ escolhido é abortada, a instância $P_c$ de $p_c$ também é abortada.

Esse comportamento é definido pelos *triggers* apresentados no Quadro 35.

*InstanceInitiated*[1][“*CHOICE*”]($P_c$)

entrada:
- uma instância $P_c$ de uma composição por *choice* $p_c = p_1|p_2|...|p_n$

condição de disparo:
- $\sigma[P_c]$ = *Initiated*
- todas as pré-condições de $P_c$ são verdadeiras

corpo:
- envie a mensagem $run(P_c)$ ao *Controlador*
- escolha $i$, $i \in [1, n]$, e crie a variável interna $chosen(P_c)$ no *blackboard* $\beta[P_c]$, inicializando-a com o valor de $i$
- envie a mensagem $initiate(p_{chosen(P_c)}, P_c)$ ao *Controlador*

---

*InstanceRunning*[1][“*CHOICE*”]($P_c$)

entrada:
- uma instância $P_c$ de uma composição por *choice* $p_c = p_1|p_2|...|p_n$

PUC-Rio - Certificação Digital Nº 0115612/CA

condição de disparo:
- $\sigma[P_c] = Running$
- $\sigma[P_{chosen(P_c)}] = PreparedToComplete$

corpo:
- envie a mensagem $complete(P_c)$ ao *Controlador*

---

$InstanceRunning[2][\text{"}CHOICE\text{"}](P_c)$

entrada:
- uma instância $P_c$ de uma composição por *choice* $p_c = p_1|p_2|...|p_n$

condição de disparo:
- $\sigma[P_c] = Running$
- $\sigma[P_{chosen(P_c)}] = Aborted$

corpo:
- envie a mensagem $abort(P_c)$ ao *Controlador*

Quadro 35 – *Triggers* que definem o comportamento do construtor *CHOICE*.

PUC-Rio - Certificação Digital Nº 0115612/CA

### 7.2.6 Any-Order

A composição por *any-order* possui a sintaxe abstrata $p_c = p_1$ - $p_2$ - ... - $p_n$, onde $p_i$ são processos, para $i \in [1, n]$. Dizemos que $p_i$ são *componentes* de $p_c$.

A composição $p_c$ determina que:

- todos os processos componentes $p_i$ de $p_c$, para $i \in [1, n]$, sejam executados, porém não concorrentemente;
- todos os processos componentes $p_i$ tenham sua execução completada com sucesso para que a instância do processo composto $p_c$ também a tenha.

Esse comportamento é definido pelos *triggers* apresentados no Quadro 36.

*InstanceInitiated*[1][“*ANY-ORDER*”]($P_c$)

entrada:

- uma instância $P_c$ de uma composição por *any-order*
$p_c = p_1$ - $p_2$ - ... - $p_n$

condição de disparo:

- $\sigma[P_c] =$ *Initiated*
- todas as pré-condições de $P_c$ são verdadeiras

corpo:

- envie a mensagem $run(P_c)$ ao *Controlador*
- crie no *blackboard* $\beta[P_c]$ uma variável interna $chosen(P_c)$ que representará o conjunto dos processos escolhidos para execução, inicializando-a com o conjunto vazio
- escolha $i$, $i \in [1, n]$, e acrescente $i$ a $chosen(P_c)$
- crie no *blackboard* $\beta[P_c]$ uma variável interna $last(P_c)$, inicializando-a com $i$
- envie a mensagem $initiate(p_{last(P_c)}, P_c)$ ao *Controlador*

---

*InstanceRunning*[1][“*ANY-ORDER*”]($P_c$)

entrada:

- uma instância $P_c$ de uma composição por *any-order*
$p_c = p_1$ - $p_2$ - ... - $p_n$

condição de disparo:

- $\sigma[P_c] =$ *Running*
- $\sigma[P_{last(P_c)}] =$ *PreparedToComplete*

corpo:

- escolha $i \in [1, n]$ tal que $i \notin chosen(P_c)$, e acrescente $i$ a $chosen(P_c)$
- mude o valor da variável interna $last(P_c)$ para $i$
- envie a mensagem $initiate(p_{last(P_c)}, P_c)$ ao *Controlador*

PUC-Rio - Certificação Digital Nº 0115612/CA

$InstanceRunning[2][\text{“}ANY\text{-}ORDER\text{”}](P_c)$

entrada:
- uma instância $P_c$ de uma composição por *any-order*
$p_c = p_1$ - $p_2$ - ... - $p_n$

condição de disparo:
- $\sigma[P_c] = Running$
- $\sigma[P_1] = \sigma[P_2] = ... = \sigma[P_n] = PreparedToComplete$

corpo:
- envie a mensagem $complete(P_c)$ ao *Controlador*

---

$InstanceRunning[3][\text{“}ANY\text{-}ORDER\text{”}](P_c)$

entrada:
- uma instância $P_c$ de uma composição por *any-order*
$p_c = p_1$ - $p_2$ - ... - $p_n$

condição de disparo:
- $\sigma[P_c] = Running$
- $\sigma[P_{last(P_c)}] = Aborted$

corpo:
- envie a mensagem $abort(P_c)$ ao *Controlador*

Quadro 36 – *Triggers* que definem o comportamento do construtor *ANY-ORDER*.

PUC-Rio - Certificação Digital Nº 0115612/CA

## 7.2.7 If-Then-Else

A composição por *if-then-else* possui a sintaxe abstrata $p_c = C?p_T{:}p_F$, onde $p_T$ e $p_F$ são processos. Dizemos que $p_T$ e $p_F$ são *componentes* de $p_c$.

A composição $p_c$ tem como base a condição de teste $C$, que determina o processo componente a ser executado. Esta condição é avaliada com base nos valores das variáveis no *blackboard* da instância de processo de $p_c$.

Suponha, esquematicamente, que $p_T$ seja o processo componente do processo composto $p_c$ que é executado quando é verdadeira a condição de teste $C$, e $p_F$ o processo executado quando é falsa esta condição.

O Quadro 37 apresenta os *triggers* que definem o comportamento desse construtor.

*InstanceInitiated*[1][“*IF-THEN-ELSE*”]($P_c$)

entrada:

- uma instância $P_c$ de uma composição por *if-then-else* $p_c = C?p_T{:}p_F$

condição de disparo:

- $\sigma[P_c] = Initiated$
- todas as pré-condições de $P_c$ são verdadeiras

corpo:

- envie a mensagem $run(P_c)$ ao *Controlador*

---

*InstanceRunning*[1][“*IF-THEN-ELSE*”]($P_c$)

entrada:

- uma instância $P_c$ de uma composição por *if-then-else* $p_c = C?p_T{:}p_F$

condição de disparo:

- $\sigma[P_c] = Running$
- $C$ é verdadeira

corpo:

- envie a mensagem $initiate(p_T, P_c)$ ao *Controlador*
- crie no *blackboard* $\beta[P_c]$ uma variável interna $chosen(P_c)$, inicializando-a com $T$

---

*InstanceRunning*[2][“*IF-THEN-ELSE*”]($P_c$)

entrada:

- uma instância $P_c$ de uma composição por *if-then-else* $p_c = C?p_T{:}p_F$

PUC-Rio - Certificação Digital Nº 0115612/CA

condição de disparo:

- $\sigma[P_c] = Running$
- $C$ é falsa

corpo:

- envie a mensagem $initiate(p_F, P_c)$ ao *Controlador*
- crie no *blackboard* $\beta[P_c]$ uma variável interna $chosen(P_c)$, inicializando-a com $F$

---

$InstanceRunning[3][\text{``}IF\text{-}THEN\text{-}ELSE\text{''}](P_c)$

entrada:

- uma instância $P_c$ de uma composição por *if-then-else* $p_c = C?p_T{:}p_F$

condição de disparo:

- $\sigma[P_c] = Running$
- $chosen(P_c) = T$
- $\sigma[P_T] = PreparedToComplete$

corpo:

- envie a mensagem $complete(P_c)$ ao *Controlador*

---

$InstanceRunning[4][\text{``}IF\text{-}THEN\text{-}ELSE\text{''}](P_c)$

entrada:

- uma instância $P_c$ de uma composição por *if-then-else* $p_c = C?p_T{:}p_F$

condição de disparo:

- $\sigma[P_c] = Running$
- $chosen(P_c) = F$
- $\sigma[P_F] = PreparedToComplete$

corpo:

- envie a mensagem $complete(P_c)$ ao *Controlador*

PUC-Rio - Certificação Digital Nº 0115612/CA

$InstanceRunning[5][\text{“}IF\text{-}THEN\text{-}ELSE\text{”}](P_c)$

entrada:

- uma instância $P_c$ de uma composição por *if-then-else* $p_c = C?p_T{:}p_F$

condição de disparo:

- $\sigma[P_c] = Running$
- $chosen(P_c) = T$
- $\sigma[P_T] = Aborted$

corpo:

- envie a mensagem $abort(P_c)$ ao *Controlador*

---

$InstanceRunning[6][\text{“}IF\text{-}THEN\text{-}ELSE\text{”}](P_c)$

entrada:

- uma instância $P_c$ de uma composição por *if-then-else* $p_c = C?p_T{:}p_F$

condição de disparo:

- $\sigma[P_c] = Running$
- $chosen(P_c) = F$
- $\sigma[P_F] = Aborted$

corpo:

- envie a mensagem $abort(P_c)$ ao *Controlador*

PUC-Rio - Certificação Digital Nº 0115612/CA

Quadro 37 – *Triggers* que definem o comportamento do construtor *IF-THEN-ELSE*.

## 7.3 Redefinição da Máquina Abstrata para OWL-S Estendida

Antes de iniciarmos a explicação sobre a máquina abstrata estendida, é importante lembrar que chamamos um *process:SimpleProcess* $p$ da ontologia de processos de OWL-S de *processo abstrato*, conforme apresentado na Seção 4.1. Além disto, chamamos de *processo concreto* o processo $p'$ escolhido como processo a ser executado no lugar de $p$, que não pode ser diretamente instanciado.

A Figura 7.3 apresenta a *máquina abstrata estendida* (MAE) para acomodar a semântica do mecanismo de tratamento de exceção para a flexibilização da execução de instâncias de processo.

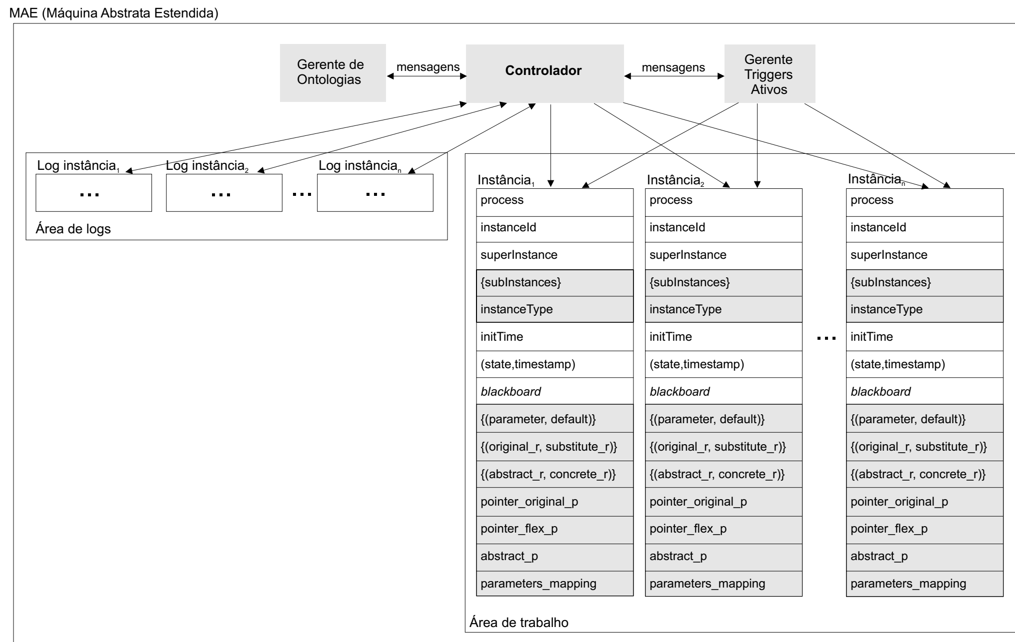


Figura 7.3: Máquina abstrata estendida (MAE).

Note que a MAE contém um componente a mais, o *Gerente de Ontologias*, responsável por implementar o mecanismo de tratamento de exceção proposto.

O acesso ao mecanismo de tratamento de exceção, disponibilizado como um serviço pelo *Gerente de Ontologias*, é feito através do envio de mensagens do *Controlador* para este componente. Da mesma forma, a resposta ao serviço é enviada por meio de mensagem do *Gerente de Ontologias* para o *Controlador* da MAE.

Cada instância $P$ de um processo $p$ mantida na área de trabalho da MAE, conforme pode ser observado na Figura 7.3, armazena, além das informações já armazenadas pela instância mantida pela MA (máquina sem extensões), informações sobre (cada uma destas novas informações acrescentadas nas instâncias estão marcadas em cor cinza claro):

– um conjunto de identificadores das instâncias dos subprocessos diretos de $p$, ou das instâncias geradas a partir da flexibilização destes subprocessos. Esta informação está representada na figura pela entrada $\{subInstances\}$;

– o tipo da instância, representado na figura pela entrada $instanceType$. Este tipo indica se a instância é relativa à execução de um processo

PUC-Rio - Certificação Digital Nº 0115612/CA

previsto normalmente no fluxo de controle do superprocesso ou se é uma instância relativa a um processo de tratamento de exceção nas suas diversas variações;

- a escolha de valores default para parâmetros. Neste caso, a instância armazena uma lista de pares (parâmetro, valor default), escolhidos conforme resposta do mecanismo de tratamento de exceção para a flexibilização pelo uso de valor default. Esta informação está representada na figura pela entrada $\{(parameter, default)\}$;
- a substituição de recursos. Neste caso, a instância armazena uma lista de pares (recurso original, recurso substituto), escolhidos conforme resposta do mecanismo de tratamento de exceção pela substituição de recursos, apresentada na figura pela entrada $\{(original_r, substitute_r)\}$;
- a concretização de recursos. Neste caso, a instância armazena uma lista de pares (recurso abstrato, recurso concreto), escolhidos conforme resposta do mecanismo de tratamento de exceção pela concretização de recursos, apresentada na figura pela entrada $\{(abstract_r, concrete_r)\}$;
- qual a instância com relação à qual a flexibilização está sendo realizada, se este for o caso. Esta informação está representada na figura através da entrada *pointer_original_p*;
- qual a instância que está tratando da exceção levantada, se este for o caso. Esta informação está representada na entrada $pointer_flex_p$ da figura;
- a concretização de um processo abstrato. Neste caso, a instância $P$ armazena na entrada *abstract_p* da figura uma referência para o processo abstrato correspondente ao processo concreto que ela está executando. Neste caso, o tipo da instância é "*concretização*";
- o mapeamento de parâmetros necessário para que a máquina possa saber, diante de uma substituição ou de uma concretização de processos, qual a correspondência dos parâmetros do processo original (ou abstrato) com os parâmetros do processo substituto (ou concreto), considerando tanto parâmetros de entrada quanto parâmetros de saída. Esta informação está representada na figura pela entrada *parameters_mapping*. A instância que foi substituída é que armazena esta informação, assim que escolhe o processo substituto dentre aqueles possíveis encontrados pelo mecanismo de tratamento de exceção pela substituição. Também é visto como um

PUC-Rio - Certificação Digital Nº 0115612/CA

processo substituto para o caso de mapeamento de parâmetros um processo advindo do tratamento de uma exceção levantada pela instância quando do estado *Initiated* ou *Running* (exceção temporal). Na concretização, a instância do processo concreto registra este mapeamento (não é gerada instância para processos abstratos).

É importante ressaltar que a informação de mapeamento de recursos, que ocorre apenas no caso de concretização, não precisa ser armazenada explicitamente na instância, pois em nenhum momento futuro da execução esta informação é necessária. Ela fica apenas registrada no log de execução da instância concreta obtida a partir da concretização do processo abstrato correspondente. O mapeamento de parâmetros, por outro lado, precisa ser registrado na instância pois a MAE necessitar do valor de um parâmetro de um processo que teve sua instância substituída.

Assumimos que o tipo *instanceType* de uma instância $P'$ é igual à cadeia de caracteres:

- "*temporal*", quando $P'$ é relativa a um processo de tratamento de exceção temporal de uma outra instância $P$;
- "*concretização*", quando $P'$ é relativa a um processo concreto escolhido a partir de um processo abstrato encontrado na definição do superprocesso, durante o tratamento da exceção por concretização;
- "*substituição*", quando $P'$ é relativa a um processo que está substituindo o processo cuja instância $P$ levantou exceção para substituição de processo;
- "*undo*", quando $P'$ é relativa a um processo para desfazer os efeitos da instância atômica abortada (estado *PreparedToAbort*), para o tratamento da exceção por cancelamento; e
- "*null*", quando a instância $P'$ é relativa a um processo $p'$ previsto no fluxo de controle da definição do superprocesso de $p'$.

Nos três primeiros casos, dizemos que $P'$ é uma *instância de flexibilização*. Nos casos de exceção e substituição, dizemos que $P'$ *substituiu* $P$.

A entrada *pointer_original_p* de $P'$ identifica a instância $P$ que sofreu exceção e pela qual $P'$ está sendo executada. Por outro lado, a entrada *pointer_flex_p* de $P$ contém uma referência para $P'$. Se o valor de *instanceType* para $P'$ é "*null*", então o valor de *pointer_original_p* para $P'$ também será *null*. Isso porque como a instância $P'$ não está tratando

PUC-Rio - Certificação Digital Nº 0115612/CA

nenhuma exceção levantada por outra instância (*instanceType* é "*null*"), $P'$ não tem associada uma instância que tenha provocado a sua execução para o tratamento de exceção e, portanto, *pointer_original_p* de $P'$ é *null*.

Ressaltamos que assumimos que qualquer instância de flexibilização $P'$ também pode sofrer flexibilização, exceto instâncias que estão desfazendo os efeitos de uma instância de processo atômico abortada, no caso do tratamento de exceção por cancelamento. Assim, *pointer_original_p* e *pointer_flex_p* definem de fato definem uma lista duplamente encadeada.

A Tabela 7.1 apresenta um resumo dos valores de *instanceType* que podem ser assumidos por uma instância $P$ de processo, e os respectivos valores de *pointer_original_p*, *pointer_flex_p* e *abstract_p* da instância, assumindo que a instância $P$ não sofreu flexibilização.

| **instanceType** | **Exceção** | **pointer_ original_p** | **pointer_ flex_p** | **abstract_p** |
|---|---|---|---|---|
| "**temporal**" | temporal | $P$ | null | null |
| "**concretização**" | por concretização de processo | null | null | $p$ |
| "**substituição**" | para substituição de processo | $P$ | null | null |
| "**undo**" | por cancelamento | $P$ | null | null |
| "**null** | null | null | null | null |

Tabela 7.1: Resumo dos valores de uma instância que podem ser assumidos para *pointer_original_p*, *pointer_flex_p* e *abstract_p*, conforme o valor de *instanceType*.

Por exemplo, uma instância $P_1$ pode sofrer exceção, ser tratada por uma instância $P_2$, que por sua vez também pode sofrer exceção e ser tratada por outra instância $P_3$. Neste caso: (1) o ponteiro *pointer_flex_p* de $P_1$ aponta para $P_2$ e o ponteiro *pointer_original_p* de $P_2$ aponta para $P_1$; (2) o ponteiro *pointer_flex_p* de $P_2$ aponta para $P_3$ e o *pointer_original_p* de $P_3$ aponta para $P_2$. Assim, a flexibilização é transitiva, ou seja, consideramos que $P_3$ está tratando indiretamente da exceção levantada por $P_1$. A Figura 7.4 apresenta uma representação simplificada de $P_1$, $P_2$ e $P_3$, apenas demonstrando a lista duplamente encadeada formada por estas três instâncias.

PUC-Rio - Certificação Digital Nº 0115612/CA

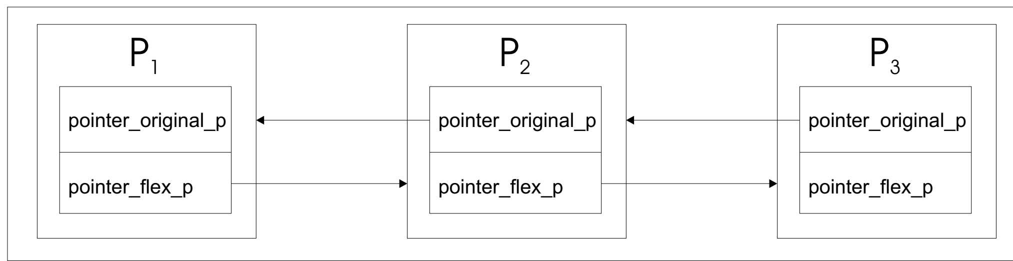


Figura 7.4: Representação simplificada da relação entre $P_1$, $P_2$ e $P_3$.

Essa é uma questão que pode ser tratada de outra forma, por exemplo, limitando a profundidade da flexibilização, ou mesmo impedindo que uma instância de flexibilização seja flexibilizada. Esta discussão fica como uma sugestão para trabalhos futuros.

Conforme já mencionado no Capítulo 5, em presença de flexibilização, a definição de subinstância é diferente. Com a flexibilização, as subinstâncias de uma determinada instância $P$ não são definidas diretamente pela estrutura de controle do processo $p$ de $P$. Considera-se subinstâncias de uma instância $P$ aquelas instâncias definidas na entrada $\{subInstances\}$ de $P$.

Considerando novamente o exemplo envolvendo $P_1$, $P_2$ e $P_3$, e o novo conceito de subinstância, a entrada $\{subInstances\}$ de $Q$, superinstância de $P_1$, deve apontar $P_3$ como a subinstância que substituiu $P_1$. Analogamente, a entrada $superInstance$ de $P_3$ deve conter uma referência para $Q$, neste caso. No entanto, repare que a entrada $superInstance$ de $P_1$ continua a apontar $Q$ como sua superinstância, apesar de $P_1$ não ser mais considerada subinstância de $Q$.

A entrada $\{subInstances\}$ de uma instância $Q$ sempre é atualizada quando há flexibilização de alguma subinstância $P_i$, exceto no caso de *undo*. Instâncias do tipo "*undo*" não possuem superinstância e a sua execução apenas determina a forma com que a instância original é terminada. Mais do que isso, assumimos que instâncias do tipo "*undo*" não sofrem flexibilização.

Esse esquema redundante de informação sobre as subinstâncias de uma determinada instância $Q$ permite aos *triggers* dos construtores verificar, de forma mais simples, o estado das subinstâncias de $Q$ apontadas na sua entrada $\{subInstances\}$, e não das subinstâncias geradas diretamente a partir de $Q$.

Isso significa que, conforme já explicado no Capítulo 5, a flexibilização força a geração dinâmica de uma árvore de instâncias, em oposição à árvore de processos estaticamente definida na definição de um processo, a partir da qual as instâncias são geradas.

PUC-Rio - Certificação Digital Nº 0115612/CA

Quando não há flexibilização, as subinstâncias de uma determinada instância $Q$ são relativas aos processos componentes de $q$. Se algum subprocesso de $q$ é flexibilizado, a entrada $\{subInstances\}$ de $Q$ é atualizada. Por outro lado, conforme já mencionado, quando há flexibilização pelo tratamento da exceção por cancelamento, ou seja, quando os efeitos de uma instância $P$ são desfeitos por uma instância $P'$, $P'$ não é vista como uma subinstância da superinstância de $P$.

Quando há flexibilização por concretização, ou seja, quando um processo abstrato em uma instância $Q$ é concretizado pela execução de uma instância $P'$ de um processo concreto $p'$, também $P'$ passa a ser subinstância de $Q$. Conforme já mencionado na Seção 5.2, um processo abstrato por si só não gera instâncias.

Quando uma exceção levantada por uma instância $P$ quando do estado *Initiated* ou *Running* é tratada por uma instância $P'$ de um processo de tratamento de exceção temporal, é como se $P'$ de fato substituísse $P$ e, portanto, $P'$ passa a ser a subinstância da superinstância $Q$ de $P$, no lugar de $P$.

A Figura 7.5 apresenta na MAE duas instâncias de processo criadas. Nesta figura, a instância $P_1$ é uma instância relativa a um processo atômico previsto no fluxo de controle de um superprocesso (é possível perceber isso porque $P_2$ é do tipo "*undo*") e $P_2$ é a instância que representa a execução de um processo que desfaz os efeitos de $P_1$, que foi abortada.

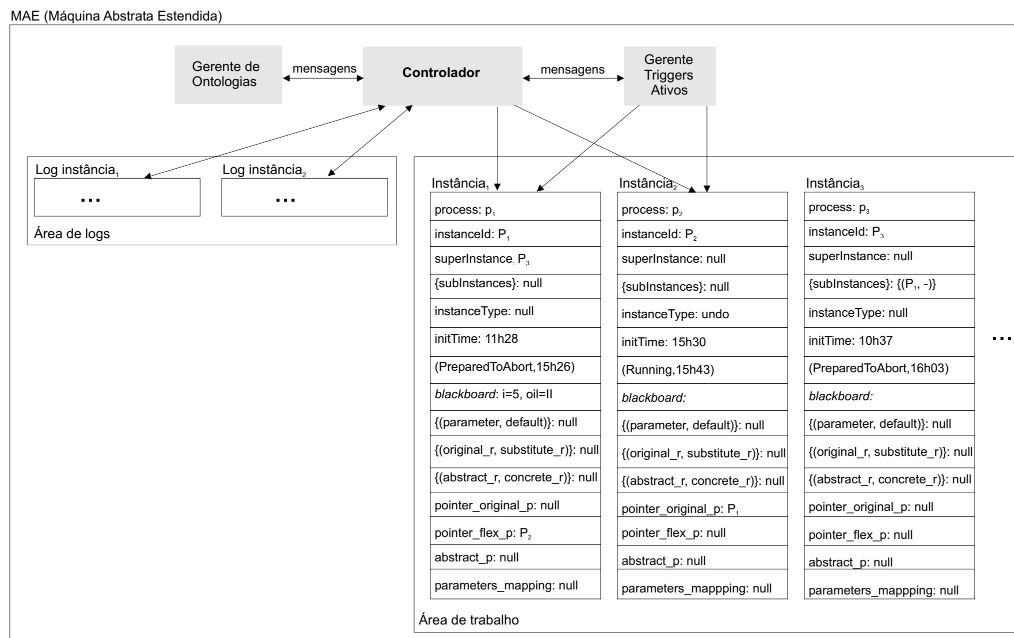


Figura 7.5: Exemplo de instâncias de processo controladas pela MAE.

Repare que $P_1$ aponta para $P_2$ indicando que ela é a instância que está

PUC-Rio - Certificação Digital Nº 0115612/CA

tratando da exceção levantada por $P_1$ (entrada *pointer_flex_p*). Ainda, $P_2$ aponta para $P_1$, indicando que ela é a instância com relação à qual ela está tratando a exceção. Este esquema de representação de instância permite que a instância original aponte para a instância relativa à flexibilização e que, por outro lado, a instância de flexibilização aponte para a instância original, sobre a qual está realizando flexibilização.

Repare também que $P_3$, superinstância de $P_1$, ainda encontra-se no estado *PreparedToAbort*, visto que $P_1$ ainda não alcançou um dos dois estados possíveis a partir de *PreparedToAbort*: *Aborted* e *Undone*.

A Figura 7.6 apresenta o diagrama de transição de estados de uma instância de processo, também modificado para acomodar a semântica do mecanismo de tratamento de exceção para a flexibilização da execução. Esta figura possui sete novos estados (ressaltados em ovais brancas), quais sejam *BeginTimed-out*, *EndTimed-out*, *Skipped*, *PreparedToByPass*, *Forced*, *PreparedToAbort* e *Undone*. Os novos estados e as novas transições são explicados nas seções seguintes.

Nessa figura, as setas pontilhadas indicam que as mensagens a elas correspondentes são processadas pelo *Controlador* durante o processamento relativo à superinstância da instância $P$ que está tendo seu estado alterado. Isso ocorre pelo comportamento transacional associado a uma instância de processo na MAE.

O diagrama de transição de estados apresentado na Figura 7.6 reflete apenas o ciclo de vida de uma instância de processo. Existem outros eventos que ocorrem no sistema, pela execução de uma determinada instância, que não estão representados no diagrama, mas que podem ser percebidos pelo corpo das mensagens processadas pelos componentes.

Para refletir a opção de "*undo*" de uma instância de processo atômico, os *triggers* na MAE devem garantir as seguintes propriedades, onde $P$ é uma instância de processo composto e a Propriedade $1'$ é uma modificação da Propriedade 1 anteriormente apresentada (veja página 150):

**Propriedade 1':** se $P$ terminou no estado *Completed*, então todas as subinstâncias diretas de $P$ terminaram no estado *Completed* ou *Forced*;

**Propriedade 2:** se $P$ terminou no estado *Aborted*, então todas as subinstâncias diretas de $P$ terminaram no estado *Aborted*, *Undone* ou *Forced* e pelo menos uma subinstância de $P$, direta ou indireta, terminou no estado *Aborted*;

PUC-Rio - Certificação Digital Nº 0115612/CA

PUC-Rio - Certificação Digital Nº 0115612/CA

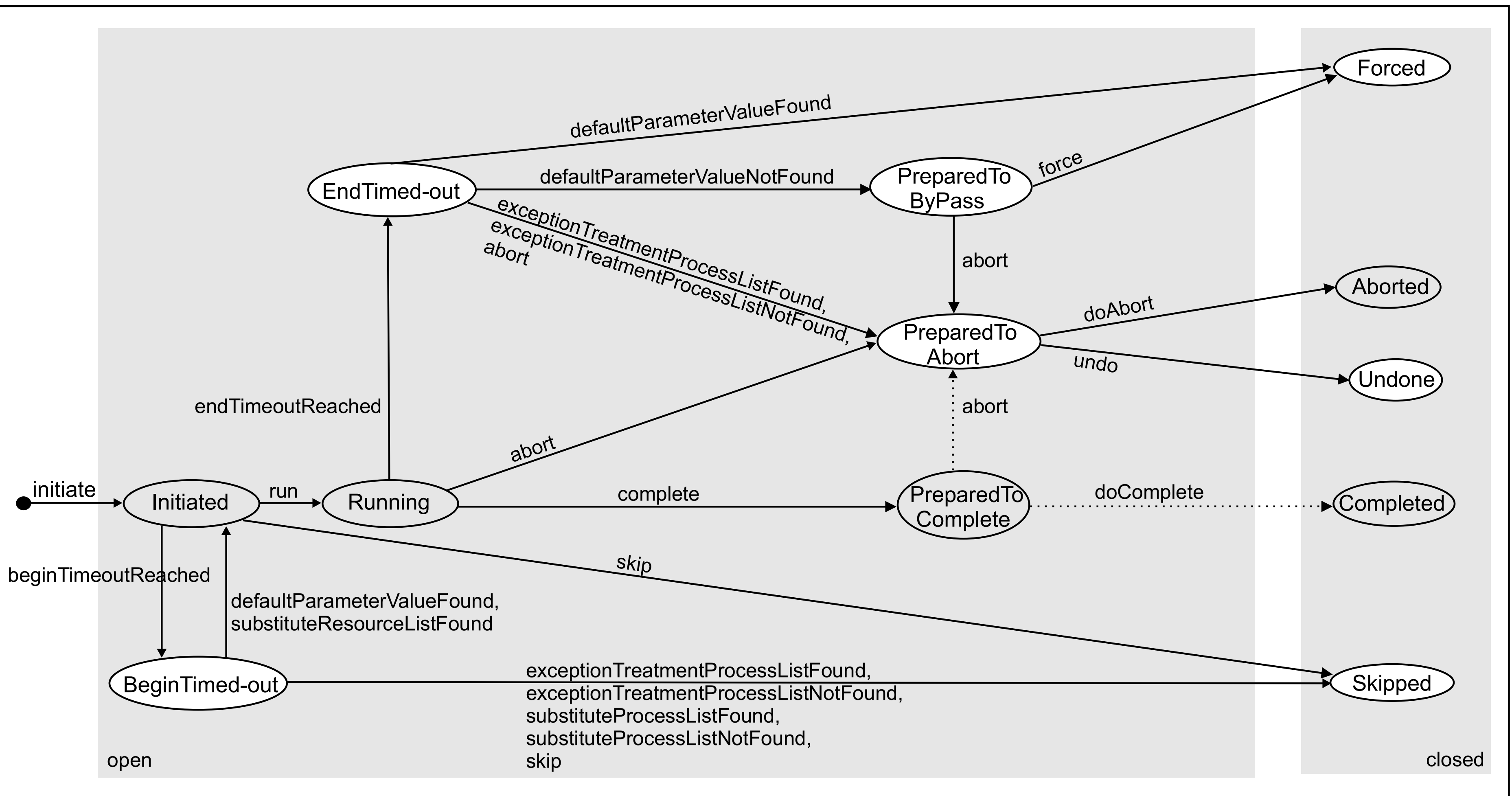


Figura 7.6: Diagrama de transição de estados modificado.

**Propriedade 3:** se $P$ terminou no estado *Undone*, então todas as subinstâncias diretas de $P$ terminaram no estado *Undone* ou *Forced*;

**Propriedade 4:** se $P$ terminou no estado *Forced*, então todas as subinstâncias diretas de $P$ terminaram no estado *Aborted* ou *Undone*.

Note que uma subinstância de uma instância $P$ é a instância que de fato executou, conforme informado na entrada $\{subInstances\}$ de $P$.

Dessa forma, os *triggers* que auxiliam o processamento das instâncias, quando associados a uma instância $P$ de um processo composto $p$, passam a testar o estado corrente, não das subinstâncias disparadas diretamente por $P$, mas sim o estado das subinstâncias registradas na entrada $\{subInstances\}$ de $P$.

De acordo com o diagrama de transição de estados modificado, pela introdução da opção de *undo*, os *triggers* dos construtores que testavam se o estado corrente das subinstâncias de uma instância $P$ era *Aborted* devem testar se o estado é *PreparedToAbort*.

Além disso, os *triggers* que testavam se o estado corrente das subinstâncias de uma instância $P$ era *PreparedToComplete* para poder completar $P$, devem agora testar se o estado das subinstâncias é *PreparedToComplete* ou *Forced*.

Ainda, de forma análoga ao que acontece na MA (veja página 154), o *Controlador* da MAE também precisa buscar o valor de cada um dos parâmetros de entrada necessários à execução de uma determinada instância de processo. Por força da flexibilização, tal busca não é tão simples pois a substituição de instâncias, seja no caso do uso do mecanismo de tratamento de exceção por substituição de processos, seja por *timeout*, deve ser levada em consideração.

Considere, por exemplo, que a instância $P$ de um processo $p$ necessite do valor de dois parâmetros, $pa_1$ e $pa_2$, provenientes do processo $p'$. Considere, ainda, a existência de mais do que uma instância relativa a $p'$, digamos $P'$ e $P''$.

Suponha que $P'$ tenha alcançado o estado *PreparedToComplete*, mas que $P''$ tenha sofrido flexibilização e alcançado o estado *Skipped*. Suponha, ainda, que a instância $P'''$ tenha sido inicializada para tratar da exceção levantada por $P''$.

Nesse caso, o *Controlador* deve percorrer a lista formada a partir da entrada $pointer_flex_p$ de $P''$, alcançando a última instância de flexibilização para $P''$, neste caso $P'''$. O *Controlador* compara então $P'$ e $P'''$, escolhendo os valores de $pa_1$ e $pa_2$ oriundos da instância mais recente.

PUC-Rio - Certificação Digital Nº 0115612/CA

Note que, para utilizar os valores oriundos de $P'''$, o *Controlador* deve considerar a informação de mapeamento de parâmetros de $P''$ para $P'''$.

De modo análogo, quando o *Controlador* necessita de um valor de parâmetro definido em um processo abstrato, ele utiliza o mapeamento de parâmetros na instância do processo concreto escolhido.

Tendo descrito a forma com que o *Controlador* consegue encontrar valores de parâmetros para a execução de instâncias de processo, passamos a apresentar agora a dinâmica do diagrama de transição de estados de uma instância na MAE.

As Seções 7.5 e 7.6 definem a semântica das extensões de tal forma que a execução de uma instância de processo na MAE está de acordo com as Propriedades $1'$, 2, 3 e 4. A O Apêndice A discute, de forma detalhada, como ocorre o processamento de uma instância de processo na MAE, de acordo com o diagrama de transição de estados apresentado. Neste apêndice são apresentadas todas as mensagens relativas ao processamento de uma instância na MAE.

PUC-Rio - Certificação Digital Nº 0115612/CA

## 7.4
## Transições de Estado na Máquina Abstrata Estendida

O Apêndice A apresenta em detalhes as mensagens processadas pelo *Controlador* e pelo *Gerente de Ontologias*, durante o processamento das instâncias de processo em execução. Este apêndice é bastante útil para a implementação dessa máquina de execução.

Esta seção visa apresentar, de forma suscinta, a dinâmica do diagrama de transição de estados na MAE.

Basicamente, quando uma instância $P$ de processo $p$ é inicializada, por meio do processamento da mensagem *initiate* pelo *Controlador*, ela alcança o estado *Initiated*. Assim que todas as suas pré-condições são satisfeitas, $P$ alcança o estado *Running* pelo processamento pelo *Controlador* da mensagem *run*.

Segundo a abordagem de flexibilização, se $P$ no estado *Initiated* sofre exceção temporal de inicialização, sendo alcançado o valor definido para a propriedade *p-ext:beginTimeout* de $p$, $P$ é colocada no estado *BeginTimed-out*. Deste estado, $P$ pode alcançar o estado *Skipped*, caso um processo $p'$ de tratamento de exceção temporal de inicialização para $p$ seja encontrado. Caso contrário, $P$ pode voltar ao estado *Initiated*, se valores default foram assumidos para os parâmetros de entrada que tinham seus valores desconhecidos (processamento da mensagem *defaultParameterValueFound*),

se este era o caso, ou se recursos alternativos foram escolhidos e alocados (processamento da mensagem *substituteResourceListFound*), se era o caso dos recursos necessários à execução de $P$ não estarem disponíveis.

No entanto, se $P$ não pode alcançar novamente o estado *Initiated*, pela flexibilização $P$ pode alcançar o estado *Skipped*, caso um processo alternativo a $p$ possa ser executado e substituindo a instância $P$. Se de qualquer modo nem $P$ nem qualquer outra instância possa ser executada em seu lugar, de acordo com o mecanismo de tratamento de exceção, $P$ alcança o estado *Skipped*, o que significa dizer que $P$ não é de fato executada, ou seja, que é abandonada antes mesmo de começar a sua execução. Neste caso, toda a hierarquia de execução é cancelada.

A partir do estado *Running*, $P$ pode alcançar *PreparedToComplete*, pelo processamento da mensagem *complete*, caso a sua execução tenha sido finalizada com sucesso. Por outro lado, se $P$ for abortada, $P$ alcança o estado *PreparedToAbort*. Ainda, se a exceção temporal de terminação é levantada por $P$, a mensagem *endTimeoutReached* é processada pelo *Controlador*, que coloca $P$ no estado *EndTimed-out*.

A partir do estado *PreparedToComplete*, $P$ pode alcançar 2 estados distintos:

- *PreparedToAbort*, caso pelo menos uma das outras subinstâncias da superinstância de $P$ tenha alcançado este estado;
- *Completed*, caso a execução de todas as subinstâncias da superinstância de $P$ tenham alcançado este estado.

A partir do estado *EndTimed-out*, $P$ pode alcançar 3 estados distintos:

- *PreparedToAbort*, caso um processo $p'$ de tratamento de exceção temporal de terminação para $p$ tenha sido encontrado, ou caso a instância $P$ seja abortada ou caso o processo de tratamento de exceção não seja encontrado e $p$ não possa sofrer flexibilização pelo uso de valor default;
- *PreparedToByPass*, caso não exista para $p$ um processo de tratamento de exceção temporal de terminação e não tenham sido encontrados para os parâmetros de saída de $P$ que não tenham tido seus valores produzidos os valores default correspondentes. Neste caso, $P$ alcança *PreparedToByPass* para que o mecanismo de tratamento de exceção busque valores default dos parâmetros de entrada da superinstância de $P$, que recebam como valores os valores dos parâmetros de saída de $P$;

PUC-Rio - Certificação Digital Nº 0115612/CA

– *Forced*, caso tenham sido encontrados valores default para os parâmetros de saída de $P$ para os quais ainda não tinham sido determinados os valores.

A partir de *PreparedToByPass*, uma instância $P$ pode alcançar os estados *Forced* e *PreparedToAbort*. $P$ vai alcançar *Forced* caso valores default sejam encontrados para seus parâmetros de saída, a partir dos parâmetros de saída de sua superinstância, no mesmo fluxo de dados. $P$ alcança *PreparedToAbort*, pelo processamento da mensagem *abort* caso valores default não sejam encontrados.

Do estado *PreparedToAbort*, $P$ pode alcançar 2 outros estados: *Aborted* e *Undone*. Se $P$ é uma instância de processo atômico, então a MAE, através do *Gerente de Ontologias*, pode buscar por processos que tratem a exceção por cancelamento de $p$. Caso um processo seja encontrado para desfazer os efeitos de $P$ e a sua execução, sem flexibilização, seja realizada com sucesso, então $P$ alcança o estado *Undone*. Caso contrário, $P$ alcança o estado *Aborted*, pelo processamento pelo *Controlador* da mensagem *doAbort*.

Se $P$ é uma instância de processo composto, então a sua terminação no estado *Aborted* ou no estado *Undone* depende da terminação das suas subinstâncias, conforme definido nos *triggers* *InstanceUndone* e *InstanceAborted*.

Se todas as subinstâncias de $P$ terminaram no estado *Undone* ou em *Forced*, então $P$ é finalizada no estado *Undone*. Caso contrário, $P$ alcança o estado *Aborted*.

Para exemplificar, considere a situação de execução descrita na Figura 7.7. Os números próximos às setas que ligam os estados determinam a ordem de processamento das mensagens durante a execução de uma instância $P$.

De acordo com o exemplo da Figura 7.7, uma instância $P$ de processo $p$ é inicializada, através da mensagem *initiate* (primeira mensagem processada) e alcança o estado *Initiated*. Neste estado, são testadas as pré-condições de $P$ que, segundo o exemplo, não são satisfeitas no tempo determinado para que $P$ comece a execução. Desta forma, é levantada a exceção temporal de inicialização, fazendo com que $P$ alcance o estado *BeginTimed-out*, pelo processamento da segunda mensagem, *beginTimeoutReached*.

Nesse estado, o *Gerente de Ontologias* busca por processos de tratamento de exceção temporal de inicialização para $P$. Segundo o exemplo, estes processos não são encontrados. Deste modo, o *Gerente de Ontologias* identifica que $P$ não teve sua execução iniciada porque não eram conhecidos

PUC-Rio - Certificação Digital Nº 0115612/CA

PUC-Rio - Certificação Digital Nº 0115612/CA

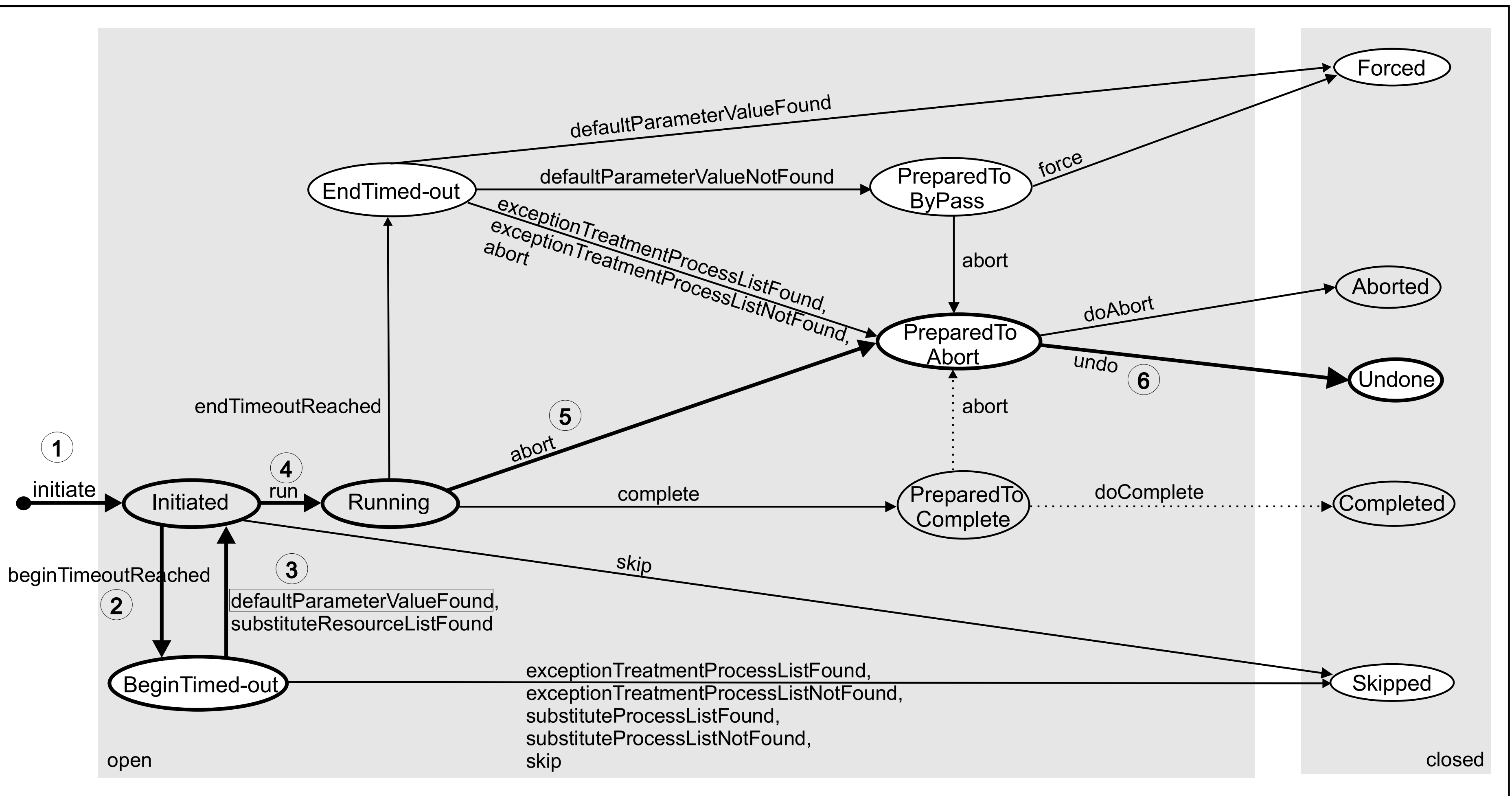


Figura 7.7: Exemplo de execução apresentado no diagrama de estados da MAE.

os valores de todos os seus parâmetros de entrada, e levanta para $P$ uma exceção para uso de valor default.

Tendo sido aplicados os valores default de todos os parâmetros de entrada de $P$ que tinham seus valores desconhecidos, $P$ alcança novamente o estado *Initiated*, pelo processamento da terceira mensagem, *defaultParameterValueFound*.

No estado *Initiated*, as pré-condições de $P$ são mais uma vez avaliadas e, conforme o exemplo, são agora avaliadas como verdadeiras, fazendo com que $P$ alcance o estado *Running*, pelo processamento da quarta mensagem, a citar *run*.

No estado *Running*, $P$ tem sua execução cancelada, alcançando o estado *PreparedToAbort*, por meio do processamento pelo *Controlador* da quinta mensagem, *abort*.

Sendo $P$ uma instância de processo atômico, uma exceção por cancelamento é levantada, fazendo com que o *Gerente de Ontologias* busque por processos que desfaçam os efeitos de $P$. Tendo encontrado pelo menos um processo para desfazer os efeitos de $P$ e tendo sido este processo executado com sucesso, $P$ alcança então o estado *Undone*, pelo processamento da sexta e última mensagem da seqüência de execução de exemplo, *undo*.

A Figura 7.8 apresenta um fluxograma que explica, passo a passo, a dinâmica do diagrama de estados de uma instância na MAE. Observe que os estados finais estão preenchidos em cinza.

O fluxograma apresentado na Figura 7.8 é bastante extenso e complexo, já que envolve cada uma das etapas de processamento de uma instância de processo na MAE. Passamos agora a apresentar esta dinâmica, dividindo o diagrama em partes menores.

A Figura 7.9 apresenta uma primeira parte do fluxograma que explica a dinâmica do diagrama de estados de uma instância na MAE, focalizando os estados intermediários e apenas dois estados finais, *Skipped* e *Forced*.

Conforme mostra a figura, quando uma instância $P$ é inicializada, pelo processamento da mensagem *initiate*, ela alcança o estado *Initiated*. Se esta instância não for mais executada, ou seja, se for cancelada no estado *Initiated*, pelo processamento da mensagem *skip* ela alcança o estado terminal *Skipped*, que indica que a $P$ nem mesmo começou sua execução.

No estado *Initiated*, o *Controlador* testa as pré-condições de $P$. Se todas elas são avaliadas como verdadeiras antes que uma exceção temporal de inicialização seja levantada, $P$ alcança o estado *Running*, quando o *Controlador* processa a mensagem *run*.

PUC-Rio - Certificação Digital Nº 0115612/CA

PUC-Rio - Certificação Digital Nº 0115612/CA

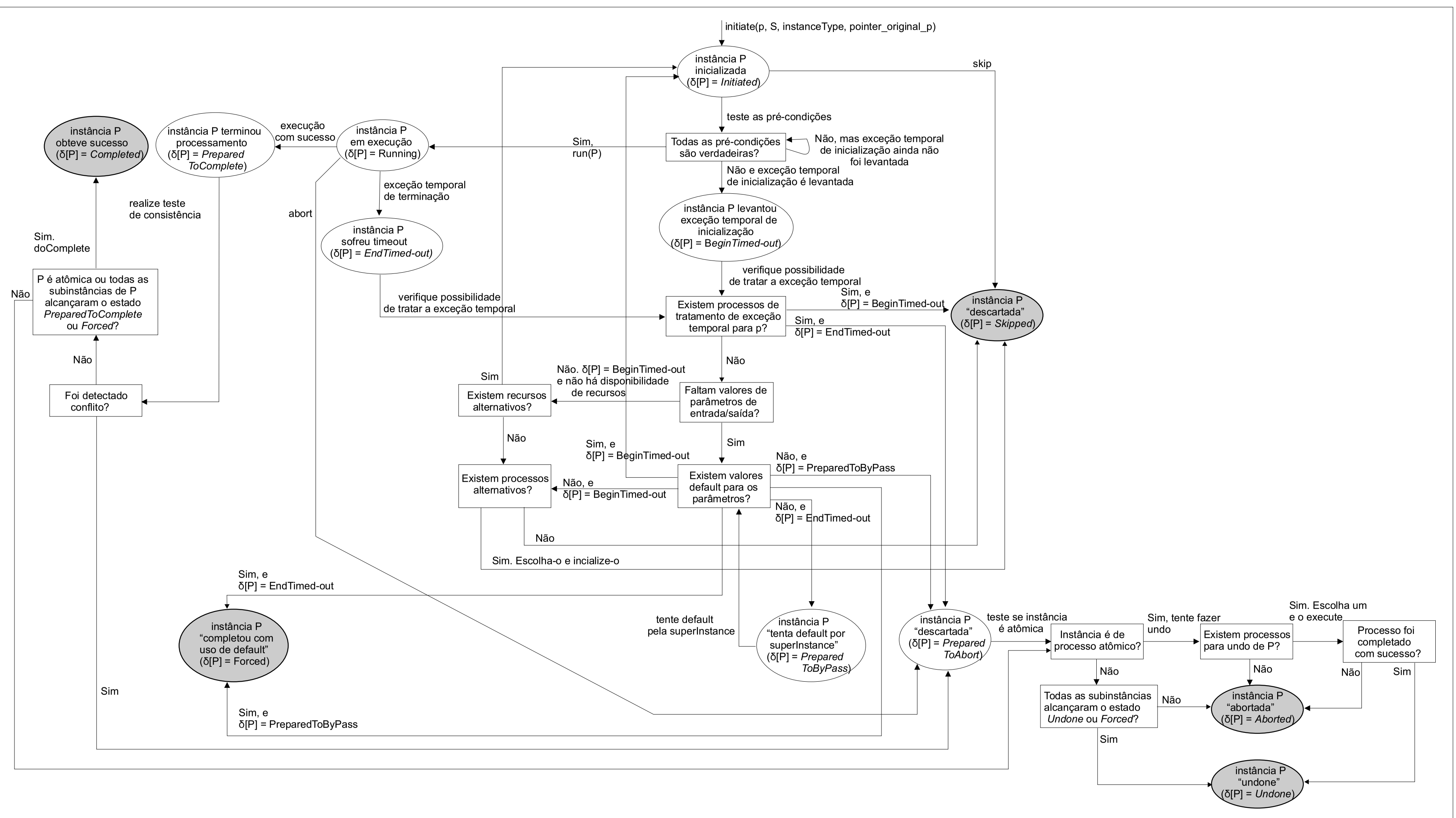


Figura 7.8: Fluxograma que explica a dinâmica do diagrama de estados de uma instância na MAE.

Enquanto $P$ não tem ultrapassado o limite temporal para inicialização e suas pré-condições são falsas, $P$ permanece no estado *Initiated*, onde suas pré-condições são constantemente avaliadas. Se a exceção temporal de inicialização é levantada, ou seja, se as pré-condições de $P$ não se tornaram verdadeiras antes que o tempo determinada para o início de sua execução fosse alcançado, $P$ alcança o estado *BeginTimed-out*.

Nesse estado *BeginTimed-out*, o *Gerente de Ontologias* verifica se existem para o processo $p$ de $P$ processos que tratem sua exceção temporal de inicialização. Caso exista pelo menos um e este possa ser executado, então $P$ é "descartada", alcançando o estado *Skipped*. Neste caso, a instância $P'$ de tratamento de exceção temporal é inicializada e substitui $P$.

Caso não existam processos que tratem a exceção temporal de inicialização de $P$, o *Gerente de Ontologias* verifica porque a exceção temporal foi levantada: porque faltam valores dos parâmetros de entrada de $P$ ou porque nem todos os recursos necessários à execução de $P$ estão disponíveis.

Se faltam valores para parâmetros de entrada de $P$, então o é levantada a exceção para uso de valor default e o *Gerente de Ontologias* verifica se existem valores default para os parâmetros cujos valores são desconhecidos. Caso estes valores existam, e como $P$ encontra-se no estado *BeginTimed-out*, estes valores default são associados aos parâmetros e $P$ volta ao estado *Initiated*, onde suas pré-condições são novamente testadas.

Caso não existam os valores default para os parâmetros cujos valores são desconhecidos, a exceção para substituição de processo é levantada e o *Gerente de Ontologias* busca por processos alternativos a $p$. Se pelo menos um processo alternativo (semanticamente próximo) é encontrado para $p$ e pode ser inicializado, $P$ alcança o estado *Skipped*, porque é substituída pela nova instância gerada a partir do processo alternativo. Por outro lado, $P$ também alcança o estado *Skipped* se processos alternativos não foram encontrados, porque a sua execução então não pode ser continuada e nem mesmo existe um processo que possa substituir $p$.

Seguindo outro caminho no fluxograma, a partir do ponto em que $P$ encontra-se no estado *BeginTimed-out* e $P$ alcançou este estado por não estarem todos os recursos necessários disponíveis, a exceção para a substituição de recursos é levantada e o *Gerente de Ontologias* busca por recursos alternativos. Se existem recursos alternativos para os recursos indisponíveis de que $P$ necessita, estes recursos alternativos são alocados para $P$ e $P$ volta ao estado *Initiated*, onde suas pré-condições são novamente avaliadas.

PUC-Rio - Certificação Digital Nº 0115612/CA

PUC-Rio - Certificação Digital Nº 0115612/CA



Figura 7.9: Parte 1 do fluxograma que explica a dinâmica do diagrama de estados de uma instância na MAE.

Assim como ocorre quando não existem valores default de parâmetros para uma instância $P$ no estado *BeginTimed-out*, se não existem recursos alternativos para $P$, a exceção para substituição de processo é levantada e o *Gerente de Ontologias* busca por processos alternativos para $p$. Se eles são encontrados e um deles é inicializado, $P$ é substituído pela nova instância gerada e alcança o estado *Skipped*. Caso contrário, $P$ alcança *Skipped* indicando que não pode ser executada, o que implica no cancelamento de toda a hierarquia de execução.

A partir do estado *Running*, $P$ alcança o estado *PreparedToAbort* se a sua execução é abortada, pelo processamento pelo *Controlador* da mensagem *abort*. A partir de *Running*, $P$ também pode alcançar o estado *EndTimed-out*, pelo levantamento da exceção temporal de terminação, diante da qual o *Gerente de Ontologias* verifica se não existem processos que tratem esta exceção. Se existe pelo menos um destes processos e ele pode ser inicializado, então $P$ é colocada pelo *Controlador* no estado *PreparedToAbort*, porque é substituída pela nova instância gerada, mas já tinha iniciado sua execução.

Caso processos de tratamento de exceção temporal de terminação não sejam encontrados para $P$, a exceção para uso de valor default é levantada e o *Gerente de Ontologias* busca por valores default para os parâmetros de saída de $P$ que ainda não tem seus valores determinados.

Se são encontrados valores default para todos os parâmetros de saída de $P$ que tinham seus valores desconhecidos, $P$ alcança o estado *Forced*, indicando que os valores de saída não foram todos produzidos por meio da execução.

Se os valores default não foram encontrados, $P$ alcança o estado *PreparedToByPass*, para que, caso haja um fluxo de dados de $P$ para sua superinstância, então sejam buscados os valores default dos parâmetros de saída da superinstância, que devem, por definição, ser iguais aos valores default dos parâmetros de saída de $P$.

Caso sejam encontrados os valores default para os parâmetros de saída da superinstância de $P$, e como $P$ estava em *PreparedToByPass*, $P$ alcança o estado *Forced*. Caso contrário, a execução de $P$ não pode ser continuada, o que faz com que $P$ alcance *PreparedToAbort*.

A Figura 7.10 apresenta então a segunda parte do fluxograma que explica a dinâmica do diagrama de estados de uma instância na MAE, enfatizando as transições de estado a partir de *PreparedToComplete* e *PreparedToAbort*.

PUC-Rio - Certificação Digital Nº 0115612/CA

PUC-Rio - Certificação Digital Nº 0115612/CA

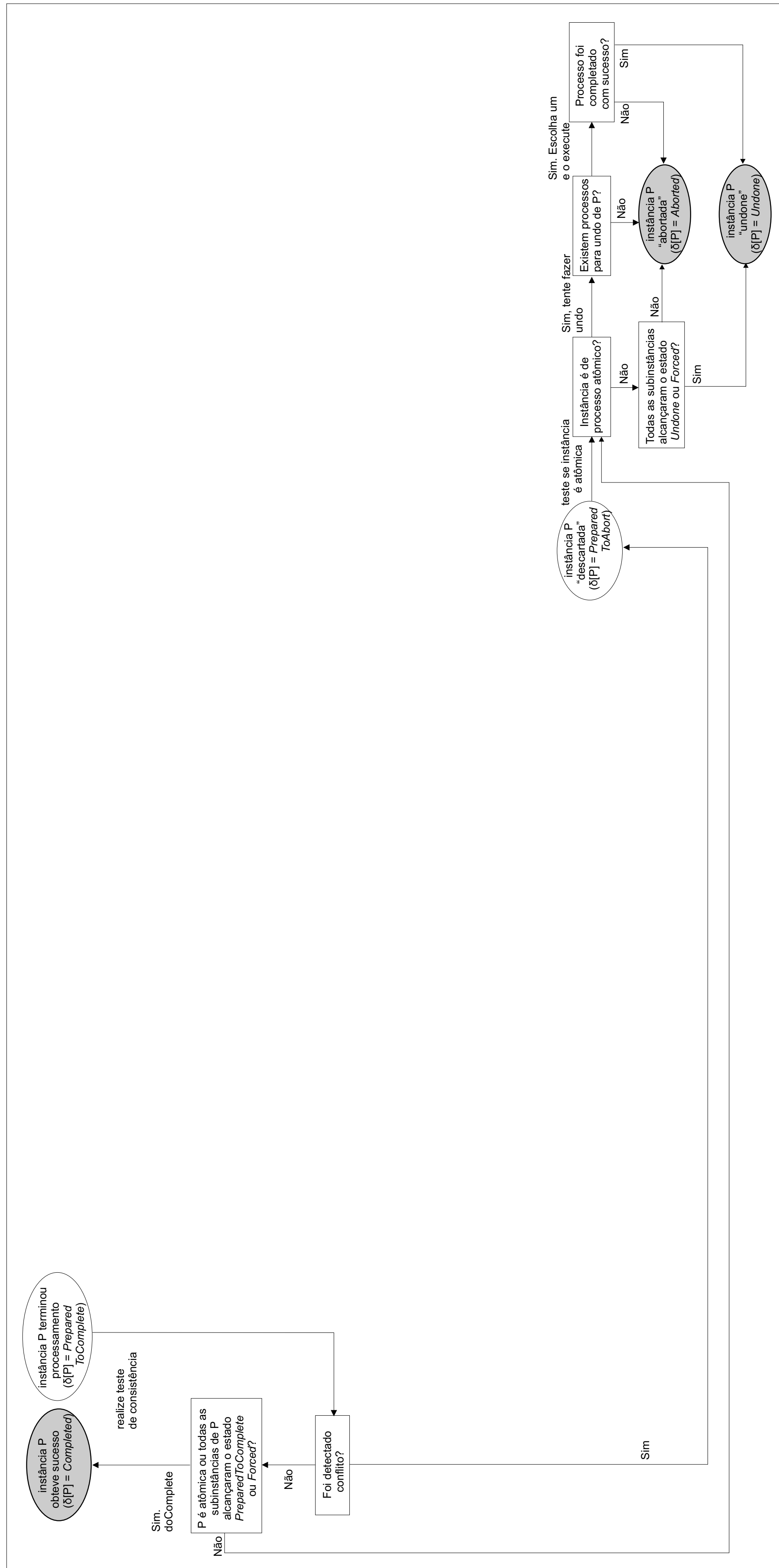


Figura 7.10: Parte 2 do fluxograma que explica a dinâmica do diagrama de estados de uma instância na MAE.

A partir do estado *PreparedToComplete*, o *Controlador* realiza teste de consistência (veja Seção 7.8), para garantir que não ocorreu conflito diante de uso de valor default em parâmetros da instância. O teste é realizado com relação a $P$ se $P$ fez uso de valor default quando no estado *BeginTimed-out*. Se nenhum conflito foi detectado e todas as subinstâncias de $P$ alcançaram o estado *PreparedToComplete* ou *Forced*, ou se $P$ é uma instância de processo atômico e foi processada a mensagem *doComplete* para $P$, então $P$ é colocada no estado *Completed*.

Se conflito não foi detectado e nem todas as subinstâncias de $P$ alcançaram *PreparedToComplete* ou *Forced*, ou se $P$ é atômico mas a mensagem *doComplete* não foi processada para $P$, é porque algum problema ocorreu. Neste caso, se $P$ é atômico, então $P$ foi cancelada. Desse modo, é levantada a exceção por cancelamento para $P$. Se não existem processos que possam desfazer os efeitos de $P$, $P$ é colocada pelo *Controlador* no estado *Aborted*. Se pelo menos um processo que desfaça os efeitos de $P$ foi encontrado e executado com sucesso, então $P$ alcança o estado *Undone*. Se este processo de flexibilização não foi executado com sucesso para desfazer os efeitos de $P$, $P$ é colocada no estado *Aborted*.

Por outro lado, se $P$ era uma instância de processo composto e todas as suas subinstâncias alcançaram o estado *Undone* ou o estado *Forced*, $P$ é colocada pelo *Controlador* no estado *Undone*. Caso contrário, é colocada no estado *Aborted*.

Ainda, se no teste de consistência é detectado um conflito, a instância $P$ é colocada no estado *PreparedToAbort*.

Todas as mensagens processadas durante a execução de uma instância de processo na MAE são apresentadas em detalhes no Apêndice A.

## 7.5
## A Semântica das Extensões Básicas da Ontologia de Processos e de Recursos de OWL-S

Esta seção discute a semântica das extensões básicas propostas às ontologias de processos e de recursos de OWL-S. A sintaxe das extensões pode ser consultada na Seção 6.4.

PUC-Rio - Certificação Digital Nº 0115612/CA

### 7.5.1 Construtor ForAll

O construtor *p-ext:ForAll* possui a sintaxe abstrata $p_c = V; L; d$, onde $V$ é uma variável, indicada na propriedade *p-ext:hasForAllVar*, $L$ é uma lista de elementos, indicada na propriedade *p-ext:fromList*, e $d$ é o corpo a ser executado, indicado na propriedade *p-ext:do*. Dizemos que $d$ é um *componente* de $p_c$.

Para cada elemento $l$ de $L$, uma instância do corpo $d$ é criada e executada, em que $V$ recebe o valor $l$; se $L$ for vazia, a execução termina sem que nada seja alterado. Quando todas as instâncias de $d$ terminarem, $P_c$ termina.

O Quadro 38 apresenta os *triggers* que definem o comportamento desse construtor. Note que, assumindo a existência de um comando de designação da forma $x := t$ em OWL-S, o construtor *p-ext:ForAll* assemelha-se então ao construtor *SPLIT-JOIN*.

PUC-Rio - Certificação Digital Nº 0115612/CA

$InstanceInitiated[1][\text{“}FORALL\text{”}](P_c)$

entrada:
- uma instância $P_c$ de uma composição $p_c = V; L; d$

condição de disparo:
- $\sigma[P_c] = Initiated$
- todas as pré-condições de $P_c$ são verdadeiras

corpo:
- envie a mensagem $run(P_c)$ *Controlador*
- para cada $l$ em $L$, construa o processo $d_l$ da forma $V := l; d$ e envie a mensagem $initiate(d_l)$ ao *Controlador* ($d_l$ é então uma subinstância de $P_c$)

---

$InstanceRunning[1][\text{“}FORALL\text{”}](P_c)$

entrada:
- uma instância $P_c$ de uma composição $p_c = V; L; d$

condição de disparo:
- $\sigma[P_c] = Running$

- todas as subinstâncias de $P_c$ alcançaram o estado *PreparedToComplete*

corpo:
- envie a mensagem $complete(P_c)$ ao *Controlador*

---

$InstanceRunning[2][\text{“}FORALL\text{”}](P_c)$

entrada:
- uma instância $P_c$ de uma composição $p_c = V; L; d$

condição de disparo:
- $\sigma[P_c] = Running$
- $\sigma[d_l] = PreparedToAbort$, para alguma subinstância $d_l$ de $P_c$

corpo:
- envie a mensagem $abort(P_c)$ ao *Controlador*

Quadro 38 – *Triggers* que definem o comportamento do construtor *FORALL*.

PUC-Rio - Certificação Digital Nº 0115612/CA

### 7.5.2 Classes AbstractResource e ConcreteResource e Propriedades implementedBy e implementationOf

O mecanismo de tratamento de exceção por concretização de recursos implementado pelo *Gerente de Ontologias* permite que recursos abstratos encontrados na definição de um processo $p$ em execução por meio de uma instância $P$ sejam substituídos por recursos concretos a eles associados.

Um recurso abstrato é definido como uma instância da classe *r-ext:AbstractResource*, ao passo que um recurso concreto é uma instância da classe *r-ext:ConcreteResource*, conforme apresentado na Seção 6.4.2, que descreve estas extensões sobre a ontologia de recursos de OWL-S. Para se encontrar quais recursos concretos implementam um determinado recurso abstrato $r$, o mecanismo de tratamento de exceção utiliza a propriedade *r-ext:implementedBy* de $r$, por meio de instâncias da classe *pr:Relation_for_value* cujo valor da propriedade *pr:has-name* seja igual à cadeia de caracteres “*implementedBy*”.

No entanto, como será visto na Seção 7.6.4, se nenhum recurso concreto de um recurso abstrato $r$ é encontrado, podem ser buscados recursos concretos de outro recurso abstrato $r_1$ semanticamente próximo a $r$, conforme definição de proximidade semântica já apresentada na página 133.

### 7.5.3 Propriedades requires, requiredBy e allocated

As propriedades *pr:requires* e *pr:requiredBy* indicam, respectivamente, os recursos necessários à execução de um determinado processo e, de modo oposto, dado um recurso, os processos que dele necessitam para executar.

A propriedade *pr:requires* é utilizada quando o mecanismo de tratamento de exceção é invocado para:

– encontrar processos de tratamento de exceção temporal de um determinado processo $p$;
– encontrar processos concretos relativos a um determinado processo abstrato $p$ (exceção por concretização de processo);
– encontrar processos alternativos a um determinado processo $p$ (exceção para a substituição de processo);
– encontrar processos para desfazer os efeitos de uma determinada instância de um processo atômico $p$ (exceção por cancelamento).

Além de buscar os recursos necessários para a execução de um determinado processo $p$, seja ele um processo de tratamento de exceção temporal, um processo concreto, um processo alternativo ou um processo para desfazer os efeitos de uma instância de processo atômico, o *Gerente de Ontologias* ainda precisa:

1. verificar a quantidade necessária de cada recurso, caso o recurso especificado seja uma instância de *r-ext:AbstractResource*;

2. verificar a disponibilidade da quantidade de recursos requeridos, no caso de recursos abstratos, ou a disponibilidade dos recursos concretos especificados.

Para verificar a quantidade de cada recurso abstrato requerida, o *Gerente de Ontologias* realiza uma consulta às instâncias da classe *pr:Relation_for_value* cujo valor da propriedade *pr:has-name* seja igual à

PUC-Rio - Certificação Digital Nº 0115612/CA

cadeia de caracteres "*requires*" e cujo valor de *pr:aRelates* seja uma referência para o processo $p$.

Para verificar a disponibilidade de recursos, o *Gerente de Ontologias* consulta o valor da instância da propriedade *r-ext:allocated* de cada recurso, seja ele um recurso concreto previamente definido ou um recurso concreto encontrado a partir de um dado recurso abstrato.

Existem basicamente duas políticas para a alocação de recursos: a política pessimista e a política otimista. De acordo com a política pessimista, todos os recursos necessários à execução de uma alternativa $A$ de uma instância $P$ de processo são alocados para $A$ no momento em que a verificação da sua disponibilidade ocorre. Note que se um recurso já estiver sido alocado para uma alternativa $A'$ de $P$, a alternativa $A$ não é desconsiderada, uma vez que o recurso já está reservado indiretamente para $P$, seja qual for a alternativa escolhida. Essa reserva é completamente diferente do caso em que o recurso está de fato alocado para alguma outra instância de processo $P'$. Neste caso, a alternativa $A$ de $P$ seria desconsiderada. A desvantagem desta política pessimista é que ela pode levar à uma situação de *deadlock*.

De acordo com a política otimista, quando o *Gerente de Ontologias* encontra as alternativas de processos, ele verifica momentaneamente a disponibilidade de recursos, mas não pré-aloca os recursos relativos a todas aquelas alternativas. As alternativas para as quais não estão disponíveis todos os recursos são desconsideradas. A desvantagem desta política é que com ela pode ser necessário recomeçar a busca pelas alternativas (através do mecanismo de tratamento de exceção para a substituição, por concretização, por cancelamento ou mesmo no momento da busca por processos de tratamento de exceção temporal) diversas vezes, se quando a alternativa final for escolhida os recursos necessários não mais estiverem disponíveis. Esta situação é conhecida na literatura como a situação de *livelock*.

Conforme foi definido nesta tese, a política de alocação de recursos adotada é a pessimista (recursos não podem ser compartilhados quando estão em uso por uma determinada instância de processo) e ela é de responsabilidade do *Gerente de Ontologias*. Ele mantém registro de que instância de processo alocou qual recurso.

Além disso, quando da escolha de processos, sejam eles processos semanticamente próximos a um processo $p$ (veja conceito de semanticamente próximo na Seção 7.6.4), processos de tratamento de exceção temporal associados ao processo $p$, processos concretos ou processos para desfazer

PUC-Rio - Certificação Digital Nº 0115612/CA

os efeitos da instância de $p$, diante de exceção por cancelamento, o *Gerente de Ontologias* controla a alocação de recursos, registrando as pré-alocações para as alternativas de processos. Quando uma das alternativas é escolhida, o *Gerente de Ontologias* desaloca aqueles recursos que estavam alocados para as demais alternativas que não foram escolhidas, conforme poderá ser observado no corpo das mensagens.

## 7.6 A Semântica das Extensões Voltadas para a Flexibilização

### 7.6.1 Propriedades costR e costP

A propriedade *r-ext:costR* indica o custo de utilização de um recurso. A propriedade *p-ext:costP* indica o custo de execução de um processo, sem levar em consideração o custo de utilização dos recursos necessários à sua execução. Elas são utilizadas, conforme apresentado na Seção 7.6.4, pelo modelo de custo empregado pelo mecanismo de tratamento de exceção para a flexibilização:

– temporal;

– por concretização;

– para substituição;

– por cancelamento.

PUC-Rio - Certificação Digital Nº 0115612/CA

### 7.6.2 Propriedade flexibilize

A propriedade *p-ext:flexibilize* associada a um processo $p$ define se uma instância deste processo em execução pode ou não sofrer flexibilização pela substituição e pelo uso de valor default. Em alguns casos pode não ser viável a utilização do mecanismo de tratamento de exceção nestes dois casos.

A propriedade *p-ext:flexibilize* também pode estar associada a *process:Perform*, que indica a referência a um processo $p$ em um processo composto. Se este construtor possuir esta propriedade, isso significa que, naquele contexto, o processo $p$ deve obedecer o esquema de flexibilização por ela indicado. Porém, se para o processo $p$ também está definida esta propriedade, vale o esquema de flexibilização definido por $p$. Isso porque

processos podem ser rígidos e críticos de tal forma que, independentemente de contexto, não se pode deixá-los flexibilizar.

Para permitir a flexibilização, a propriedade *p-ext:flexibilize* pode conter os seguintes valores:

– *by_substitution*, indicando que o mecanismo de tratamento de exceção para substituição de processos ou de recursos pode ser invocado;
– *by_default*, indicando que o mecanismo de tratamento de exceção para uso de valor default pode ser invocado;
– *all*, indicando que ambas as alternativas podem ser utilizadas.

Note que não são considerados para essa propriedade valores correspondentes ao uso de processos de tratamento de exceção temporal, por concretização e por cancelamento, porque sob estas condições é obrigatório permitir a flexibilização da execução de qualquer instância de processo.

PUC-Rio - Certificação Digital Nº 0115612/CA

### 7.6.3
### Propriedades para o Tratamento de Exceção

A flexibilização da execução pelo alcance do valor de *beginTimeout* de um processo ocorre quando a instância não pode ser executada porque as suas pré-condições permanecem falsas por indisponibilidade dos recursos necessários à execução ou não podem ser avaliadas por falta de valores de variáveis (parâmetros de entrada ou não) por um tempo superior ao permitido, levantando uma exceção temporal de inicialização. Por *endTimeout*, a flexibilização ocorre quando a instância está demorando (tempo maior do que o determinado pelo valor da propriedade *p-ext:endTimeout* do processo) para terminar e gerar a resposta (dados de saída) esperada.

A propriedade *p-ext:beginTimeout* indica por quanto tempo o *Controlador* deve aguardar para que a instância de processo passe do estado *Initiated* para o estado *Running*. Isso significa que o *Controlador* espera o tempo determinado na instância de *p-ext:beginTimeout* para que as pré-condições da instância de processo sejam avaliadas como verdadeiras.

A propriedade *p-ext:endTimeout* indica o período de tempo que o *Controlador* deve esperar até que uma instância de processo termine sua execução, gerando os valores de saída.

O mecanismo de tratamento de exceção implementado pelo *Gerente de Ontologias* para tratar uma exceção temporal levantada por uma instância, pelo alcance dos valores das propriedades *p-ext:beginTimeout* ou

*p-ext:endTimeout* do processo $p$ correspondente, atua sobre a ontologia de aplicação, buscando por instâncias de processos $p'$ que tratem a exceção correspondente.

Um processo de tratamento de exceção $p'$ está associado ao processo $p$ do qual trata a exceção por meio das propriedades *p-ext:beginTimeoutTreatedBy* e *p-ext:endTimeoutTreatedBy* deste processo.

Processos referenciados pelas instâncias da propriedade *p-ext:beginTimeoutTreatedBy* tratam das exceções temporais de inicialização levantadas quando a instância encontra-se no estado *Initiated* e o tempo desde que alcançou este estado é maior do que o tempo definido pela instância da propriedade *p-ext:beginTimeout*.

Tratar uma exceção neste caso significa:

– pesquisar um processo $p'$ definido para tratar este tipo de exceção para $p$;
– criar uma instância $P'$ de $p'$ para representar a execução de $P$;
– colocar $P'$ como subinstância da superinstância de $P$, em lugar de $P$.

Processos referenciados pelas instâncias da propriedade *p-ext:endTimeoutTreatedBy* tratam das exceções temporais de terminação levantadas quando a instância encontra-se no estado *Running* e o tempo desde que alcançou este estado é maior do que o tempo definido pela instância da propriedade *p-ext:endTimeout*.

A explicação intuitiva para o tratamento deste tipo de exceção é inteiramente semelhante ao caso anterior.

Como podem ser vários os processos de tratamento de exceção associados a um determinado processo $p$, para cada um dos casos (*p-ext:beginTimeout* e *p-ext:endTimeout*), a resposta do mecanismo de tratamento de exceção a uma requisição é uma lista de $(process, resources, cost_model_value, parameters_mapping)$, ordenada de acordo com o custo encontrado pela aplicação do modelo de custo, conforme explicado na Seção 7.6.4.

As instâncias da propriedade de processo *p-ext:processUndo* indicam, para um determinado processo atômico $p$, os processos que podem ser executados para desfazer os efeitos de uma instância $P$ de $p$ que não obteve sucesso na execução (exceção por cancelamento). O estado final de uma instância atômica $P$ que não obteve sucesso é dependente do estado final da instância $P'$ que está tratando a exceção levantada por $P$. Vale relembrar que instâncias do tipo "*undo*" não podem sofrer flexibilização.

PUC-Rio - Certificação Digital Nº 0115612/CA

PUC-Rio - Certificação Digital Nº 0115612/CA

Seguindo a hierarquia de exceções apresentada na página 104, o mecanismo de tratamento de exceção para o uso de valor default para uma instância $P$ de um processo $p$ pode ser invocado, quando permitido (através da propriedade *flexibilize* de $p$ ou de *process:Perform* para a execução de $p$):

- quando a instância $P$ encontra-se no estado *BeginTimed-out* e as pré-condições de $P$ não podem ser avaliadas porque são desconhecidos alguns dos valores de parâmetros;
- quando a instância $P$ encontra-se no estado *EndTimed-out*, porque ela não conseguiu terminar sua execução gerando os valores de saída;
- quando não foram encontrados valores default para os parâmetros de saída de $P$ para os quais $P$ não produziu valor, mas existe um fluxo de dados de $P$ para sua superinstância $Q$. Neste caso, o *Controlador* coloca $P$ no estado *PreparedToByPass* e solicita ao *Gerente de Ontologias* encontrar valores default para os parâmetros de $q$ que recebem os valores dos parâmetros de $p$. É importante ressaltar, novamente, que se $pa_1$ é parâmetro de saída em $P$ e seu valor é enviado ao parâmetro de saída $pa_2$ de $P'$, se houver valores default para $pa_1$ e $pa_2$, estes valores devem ser iguais, por questão de consistência.

Vale lembrar que como exceção de segundo nível, a exceção para uso de valor default apenas é levantada para uma determinada instância $P$ quando a exceção temporal já foi levantada para esta mesma instância e não pode ser resolvida.

O valor default associado a um determinado parâmetro pode ser determinado:

- diretamente, através da propriedade *p-ext:defaultParameterValue* definida para parâmetros de processos;
- indiretamente, através do uso de *regras de suposição*, que podem levar em consideração valores e condições diversos, tais como o estado de execução da instância, conforme explicado na página 108.

Ressaltamos que é assumido que o valor default é único, quando informado, para cada parâmetro associado a um processo.

### 7.6.4
### Classe Relation_for_value

A classe *pr:Relation_for_value*, e as propriedades que a ela se aplicam, dão suporte ao mecanismo de tratamento de exceção temporal, por concretização, para substituição (de processos e de recursos) e por cancelamento, conforme descrito a seguir.

Em todos esses casos, o trabalho efetuado pelo mecanismo é um trabalho de *matching*, durante o qual o mecanismo busca na ontologia de aplicação alternativas a um determinado processo ou recurso, utilizando um modelo de custo pré-definido para ordená-las.

Quando uma exceção temporal é levantada por uma instância $P$ de um processo $p$, alcançado o valor da instância da propriedade *p-ext:beginTimeout* ou de *p-ext:endTimeout* de $p$, o *Controlador* solicita ao *Gerente de Ontologias* encontrar processos que tratem desta exceção.

O mecanismo de tratamento de exceção, uma vez invocado, busca na ontologia de aplicação por todas as instâncias da classe *pr:Relation_for_value* cujo valor da propriedade *pr:has-name* seja igual à cadeia de caracteres "*beginTimeoutTreatedBy*" ou "*endTimeoutTreatedBy*", conforme o caso, e cujo valor de *pr:aRelates* seja uma referência para o processo $p$. Para cada uma destas instâncias, o mecanismo obtém o valor da propriedade *pr:has-value*, que indica a proximidade semântica entre os processos referenciados em *pr:aRelates* e *pr:bRelates*. No entanto, apenas este valor de proximidade semântica não é suficiente para determinar a escolha do processo.

Para cada processo de tratamento de exceção $p'$ encontrado, é necessário verificar a disponibilidade dos recursos dos quais necessita para a execução e realizar a pré-alocação. Além disso, deve ser considerado o custo de utilização destes recursos na computação do custo total de execução de $p'$.

A quantidade de um recurso abstrato $r$ necessária à execução de $p'$ é dada pelo valor da propriedade *pr:has-value* de instâncias da classe *pr:Relation_for_value* que tenham como valor da propriedade *pr:has-name* a cadeia de caracteres "*requires*" e, claro, como valor de *pr:aRelates* o processo $p'$ e de *pr:bRelates* o recurso $r$. Recursos concretos diretamente definidos como necessários à execução de um processo não têm a eles, obviamente, um valor de quantidade associada.

O *modelo de custo* empregado pelo mecanismo de tratamento de exceção leva em consideração:

– a proximidade semântica entre os processos sendo comparados. Por

PUC-Rio - Certificação Digital Nº 0115612/CA

exemplo, no caso de se estar procurando por um processo $p'$ de tratamento de exceção temporal de inicialização para um processo $p$, é considerada a proximidade semântica entre $p$ e $p'$;

– o fator de custo do processo $p'$; e

– para cada tipo de recurso necessário à sua execução, a quantidade necessária, caso o recurso seja abstrato, e o fator de custo a ele associado. No caso de recursos concretos, apenas é levado em consideração o seu custo, já que um recurso concreto é único e, portanto, não podem ser requeridas mais do que uma unidade dele para a execução de um processo.

Observe que é função do projetista definir o valor de proximidade semântica entre dois processos ou dois recursos. Este valor é subjetivo e depende da natureza da aplicação.

Dessa forma, considere a Figura 7.11, na qual é apresentado um exemplo esquemático de instâncias de processos e recursos, e seus relacionamentos, levados em consideração pelo modelo de custo empregado, por exemplo, pelo mecanismo de tratamento de exceção temporal. Nesta figura, instâncias de processo estão representadas por círculos e recursos por quadrados. O recurso $r_7$ representado por dois quadrados, um interno ao outro, é um recurso concreto.

PUC-Rio - Certificação Digital Nº 0115612/CA

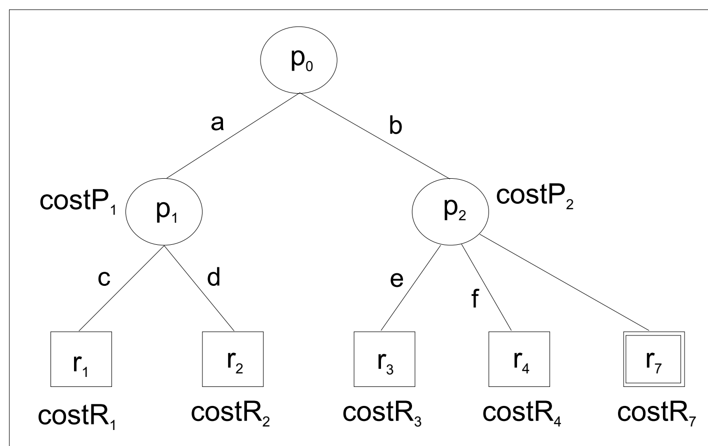


Figura 7.11: Exemplo esquemático para o cálculo de custo da utilização de $p_1$ ou $p_2$ para o tratamento da exceção levantada por uma instância de $p_0$.

Na Figura 7.11, $p_1$ e $p_2$ não são subprocessos do processo $p_0$, mas possuem proximidade semântica de tratamento de exceção $a$ e $b$, respectivamente, com relação a $p_0$. Estes relacionamentos estão definidos por instâncias da classe *pr:Relation_for_value* cujo valor da propriedade *pr:has-name* é igual à cadeia de caracteres "*beginTimeoutTreatedBy*" ou

"*endTimeoutTreatedBy*", conforme o caso, e da propriedade *pr:has-value* é $a$, para o caso do relacionamento de $p_0$ com $p_1$, e $b$, para o caso do relacionamento de $p_0$ com $p_2$.

O custo de utilizar $p_1$ para tratar a exceção temporal levantada pela instância de $p_0$ é dado pela Equação 7-1.

$$Custo(p_0, p_1) = a \times costP_1 + (c \times costR_1 + d \times costR_2) \quad (7\text{-}1)$$

e o custo de utilizar $p_2$ para tratar a exceção temporal levantada pela instância de $p_0$ é dado pela Equação 7-2.

$$Custo(p_0, p_2) = b \times costP_2 + (e \times costR_3 + f \times costR_4 + costR_7) \quad (7\text{-}2)$$

onde $a$ e $b$ são valores de proximidade semântica e $c$, $d$, $e$ e $f$ correspondem às quantidades de recursos concretos do tipo dos recursos abstratos correspondentes necessários para a execução dos processos. $costP_1$ e $costP_2$ são os custos de se executar, respectivamente, os processos $p_1$ e $p_2$. $costR_1$, $costR_2$, $costR_3$, $costR_4$ e $costR_7$ são os custos associados à obtenção de cada um dos recursos correspondentes. Observe que não existe valor de quantidade de recursos para $r_7$ com relação a $p_2$ (arco não rotulado) pelo fato deste ser um recurso concreto.

Se durante a execução do mecanismo de tratamento de exceção for verificado que a utilização de, digamos, $p_1$, para tratar a exceção levantada por uma instância de $p_0$ resulta necessariamente na substituição do recurso $r_2$ por um outro recurso $r_5$, sendo $g$ a proximidade semântica entre eles e $costR_5$ o custo de $r_5$, então o custo de utilizar $p_1$ para $p_0$ neste novo cenário é dado pela seguinte equação:

$$Custo(p_0, p_1) = a \times costP_1 + (c \times costR_1 + d \times (g \times costR_5)) \quad (7\text{-}3)$$

ou seja, o modelo de custo considera que as escolhas são independentes entre si, levando em conta apenas a proximidade semântica e os custos individuais.

Naturalmente, as equações 7-1 a 7-3 são apenas um exemplo do modelo de custo adotado. A equação de custo final é uma generalização deste exemplo.

Esse modelo de custo permite que o mecanismo obtenha e retorne, para cada alternativa, o seu valor de custo encontrado. A escolha de um processo alternativo pode ser feita automaticamente, sob o controle do *Controlador*, ou com a intervenção do usuário.

O mesmo procedimento é empregado para se encontrar processos:

PUC-Rio - Certificação Digital Nº 0115612/CA

PUC-Rio - Certificação Digital Nº 0115612/CA

– de concretização, para o qual o mecanismo busca pelas instâncias de *pr:Relation_for_value* com valor da instância da propriedade *pr:has-name* igual à cadeia de caracteres "*realizedBy*" ou "*expandsTo*";

– substitutos de um processo $p$, quando o mecanismo busca pelas instâncias de *pr:Relation_for_value* com valor da instância da propriedade *pr:has-name* igual à cadeia de caracteres "*process-process*"; e

– para desfazer os efeitos de uma instância de um processo atômico $p$, quando o mecanismo busca pelas instâncias de *pr:Relation_for_value* com valor da instância da propriedade *pr:has-name* igual à cadeia de caracteres "*processUndo*".

O valor de custo obtido pelo modelo de custo empregado determina como comparar duas alternativas possíveis a um mesmo processo $p$, mas não determina quanto custaria esperar até que a instância do processo $p$ pudesse continuar sua execução. Isso porque este custo depende da aplicação e por isso não é facilmente determinado.

Desejando comparar o quanto custaria executar um processo alternativo com quanto custaria esperar até que a execução da instância $P$ do processo $p$ pudesse ser continuada, assumimos que o custo de aguardar tende ao infinito e, portanto, executar qualquer alternativa é melhor do que esperar que o processo original possa ser executado ou ter sua instância em execução terminada.

A propriedade *pr:maps* da classe *pr:Relation_for_value* é utilizada para indicar os mapeamentos de parâmetros necessários para a correta substituição/concretização de processos, e para indicar os mapeamentos de recursos abstratos necessários no momento da concretização de processos. Assim, o mecanismo de tratamento de exceção temporal, para substituição de processos e por concretização, deve devolver como resultado não apenas um conjunto de triplas contendo os processos alternativos, seus recursos e o valor de custo obtido, mas, para cada tripla, também o conjunto de mapeamentos de parâmetros necessários, provenientes das propriedades *pr:from* e *pr:to* das instâncias da classe *pr:ParameterMap* apontadas pelas instâncias da propriedade *pr:maps* da instância da classe *pr:Relation_for_value*.

No caso de concretização de processos, o mecanismo também devolve como parte do resultado as informações de mapeamento de recursos entre o processo abstrato e cada um dos possíveis correspondentes concretos. Processos abstratos só podem referenciar recursos abstratos e, uma vez o

fazendo, deve haver um mapeamento definido entre cada recurso abstrato do processo abstrato para um recurso também abstrato associado ao processo concreto. Para este mapeamento é utilizada a propriedade *pr:maps*, com contra-domínio na classe *pr:ResourceMap*.

É importante frisar que, assim como ocorre no caso dos "relacionamentos" *process-process*, nos relacionamentos *process:expandsTo* e *process:realizedBy* de OWL-S, e nos relacionamentos *p-ext:beginTimeoutTreatedBy* e *p-ext:endTimeoutTreatedBy*, os processos relacionados devem possuir parâmetros de entrada e de saída do mesmo tipo ou de tipos da mesma hierarquia de tipos. Deste modo, fica garantida a correta substituição de um processo por outro processo alternativo. Os valores mantidos pelo superprocesso do processo original são repassados ao processo alternativo escolhido como seus valores de entrada e os valores de saída deste processo são repassados como valores de saída que deveriam ter sido gerados pelo processo original.

Vale ainda ressaltar que os processos original (ou abstrato) e substituto (ou concreto) não precisam necessariamente possuir o mesmo número de parâmetros. No entanto, o número de parâmetros do processo substituto (ou concreto) deve ser maior ou igual do que o número de parâmetros do processo original (ou abstrato). Cada parâmetro do processo original (ou abstrato) deve estar mapeado diretamente em algum parâmetro do processo substituto (ou concreto), não obrigatoriamente na mesma ordem de ocorrência.

Quando a substituição é de recursos, o mecanismo busca na ontologia de aplicação por todas as instâncias da classe *pr:Relation_for_value* cujo valor da propriedade *pr:has-name* seja igual à cadeia de caracteres "*resource-resource*" e cujo valor de *pr:aRelates* seja uma referência para cada recurso $r$. Para cada uma destas instâncias, o mecanismo obtém o valor da propriedade *pr:has-value*, que indica a proximidade semântica entre o recurso $r$ referenciado em *pr:aRelates* e o recurso referenciado em *pr:bRelates*. Vale lembrar que apenas é possível definir relacionamento de proximidade semântica entre dois recursos abstratos ou entre dois recursos concretos.

O modelo de custo é aplicado de modo semelhante ao que ocorre no mecanismo para substituição de processos. No entanto, neste caso são considerados apenas o peso do relacionamento entre os recursos e o custo de cada recurso substituto. O retorno deste mecanismo é uma lista de pares $(resource, cost_model_value)$, para cada recurso $r$, ordenada pelo valor *cost_model_value* de custo computado dos recursos alternativos

PUC-Rio - Certificação Digital Nº 0115612/CA

encontrados.

Na concretização de recursos, o *Gerente de Ontologias* considera instâncias da classe *r-ext:AbstractResource* encontradas na definição do processo e instâncias da classe *r-ext:ConcreteResource* encontradas na ontologia de aplicação. Instâncias de recursos concretos se relacionam com instâncias de recursos abstratos por meio de instâncias da propriedade *r-ext:implementedBy*. O peso do relacionamento de concretização entre um recurso abstrato e um recurso concreto é dado pelo valor da instância da propriedade *pr:has-value* da instância da classe *pr:Relation_for_value* cujo valor de *pr:aRelates* seja um uma referência para o recurso abstrato e de *pr:bRelates* uma referência para o recurso concreto, e cujo valor de *pr:has-name* seja igual à cadeia de caracteres "*implementedBy*". O mesmo cálculo de custo é empregado.

Vale ressaltar que, conforme anteriormente explicado, caso um recurso concreto não seja encontrado para um determinado recurso abstrato, o mecanismo busca na ontologia de aplicação outro recurso abstrato semanticamente próximo do original. Neste caso, uma vez encontrado um abstrato, os seus concretos são procurados.

O Quadro 39 apresenta um resumo de como é utilizada a classe *pr:Relation_for_value* na busca de processos alternativos, de recursos alternativos, de processos ou recursos concretos, de processos de tratamento de exceção temporal e de processos para desfazer os efeitos de uma determinada instância, diante de uma exceção por cancelamento.

1) Mecanismo de tratamento de exceção para substituição de processos

a) busca na ontologia de aplicação por instâncias da classe *pr:Relation_for_value*, tais que:
- a propriedade *pr:has-name* = "*process-process*"
- a propriedade *pr:aRelates* seja uma referência para o processo $p$

b) para cada instância encontrada, consulta o valor da propriedade *pr:has-value*, que indica a proximidade semântica entre o processo $p$ e o outro $p'$, indicados em *pr:aRelates* e *pr:bRelates*, respectivamente

c) para cada processo $p'$ encontrado associado a $p$ através de uma instância de *pr:Relation_for_for_value*, o mecanismo busca na ontologia de aplicação as instâncias desta mesma classe, tais que:

PUC-Rio - Certificação Digital Nº 0115612/CA

- a propriedade *pr:has-name* = "*requires*"
- a propriedade *pr:aRelates* seja uma referência para $p'$

d) para cada instância encontrada, consulta o valor da propriedade *pr:has-value* que indica a quantidade necessária de recursos do tipo $r_a$ (*r-ext:AbstractResource*) para a execução de $p'$

e) invoca o modelo de custo sobre os processos encontrados, levando em consideração a proximidade semântica entre $p$ e $p'$, o fator de custo de $p'$ e, para cada recurso abstrato, a quantidade necessária de recursos concretos correspondentes e o fator de custo associado. Ainda, leva em consideração o custo de cada recurso concreto $r_c$. Assim:

$$Custo(p, p') = proximidade(p, p') \times costP(p') + \Sigma(qtd(r_a, p') \times costR(r_a)) + \Sigma(costR(r_c))$$

de modo que quanto menor o custo, melhor é o processo alternativo

f) o processo escolhido é instanciado

---

2) Mecanismo de tratamento de exceção para substituição de recursos

a) busca na ontologia de aplicação as instâncias da classe *pr:Relation_for_value* tais que:
- a propriedade *pr:has-name* = "*resource-resource*"
- a propriedade *pr:aRelates* seja uma referência para $r$

b) aplica o modelo de custo sobre os recursos encontrados, levando em consideração a proximidade semântica entre eles e o custo do recurso alternativo. Assim:

$$Custo(r, r') = proximidade(r, r') \times costR(r')$$

c) o recurso escolhido é colocado no lugar do recurso $r$ que estava indisponível

PUC-Rio - Certificação Digital Nº 0115612/CA

PUC-Rio - Certificação Digital Nº 0115612/CA

3) Mecanismo de tratamento de exceção por concretização de processos

a) idêntico ao caso 1), exceto que no item a) temos
*pr:has-name* = "*realizedBy*" ou "*expandsTo*"

---

4) Mecanismo de tratamento de exceção por concretização de recursos

a) idêntico ao caso 1), exceto que no item a) temos
*pr:has-name* = "*implementedBy*"

---

5) Mecanismo de tratamento de exceção temporal

a) idêntico ao caso 1), exceto que no item a) temos
*pr:has-name* = "*beginTimeoutTreatedBy*", se $\sigma[P_c]$ = *BeginTimed-out*,
ou *pr:has-name* = "*endTimeoutTreatedBy*", se $\sigma[P_c]$ = *EndTimed-out*

---

6) Mecanismo de tratamento de exceção por cancelamento

a) idêntico ao caso 1), exceto que no item a) temos
*pr:has-name* = "*processUndo*"

---

7) Mecanismo de tratamento de exceção para uso de valor default

a) busca na ontologia de aplicação pelo valor da instância da propriedade *p-ext:defaultParameterValue* da instância relativa ao parâmetro especificado, ou através das *regras de suposição*

b) o valor default encontrado é atribuído ao parâmetro correspondente na instância em execução

Quadro 39 – Utilização da classe *pr:Relation_for_value* pelo mecanismo de tratamento de exceção.

A Figura 7.12 apresenta um exemplo esquemático de como é o funcionamento do mecanismo de tratamento de exceção para substituição de

processos, no que diz respeito ao acesso à ontologia de aplicação, armazenada no que chamamos na figura de *Repositório de Ontologias*.

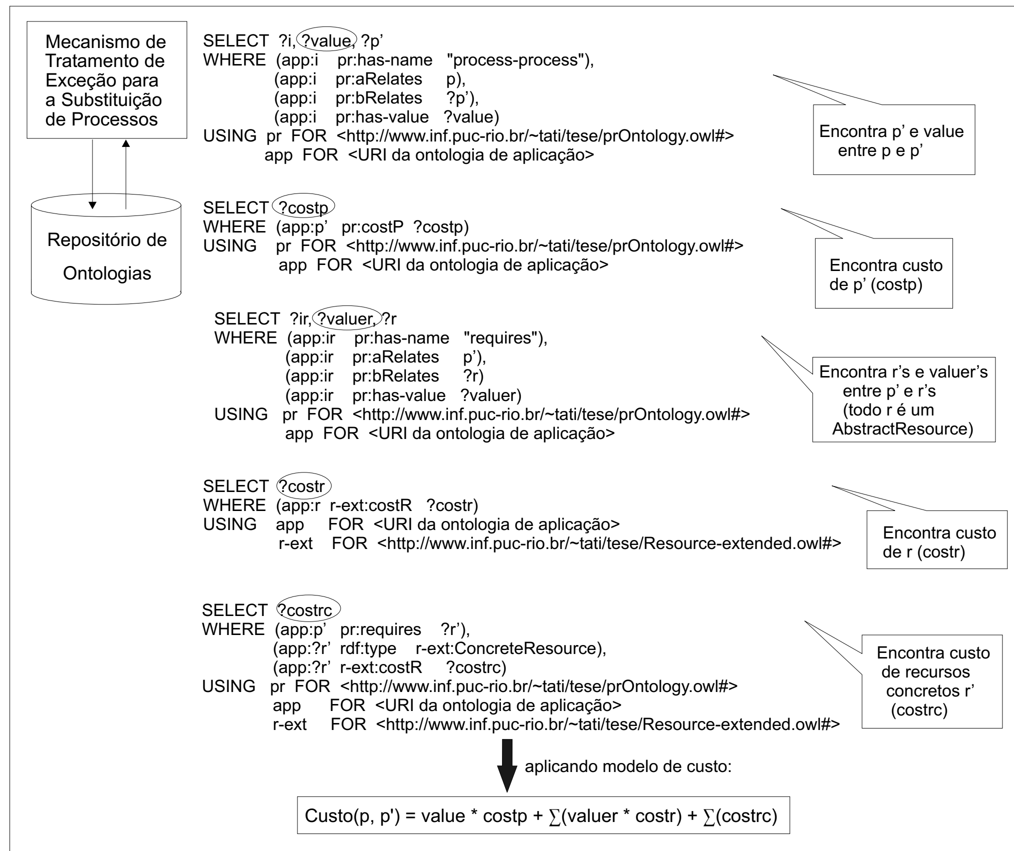


Figura 7.12: Exemplo esquemático do funcionamento do mecanismo para substituição de processos, prevendo consultas sobre a ontologia de aplicação através da linguagem RDQL (*RDF Data Query Language*) [106].

PUC-Rio - Certificação Digital Nº 0115612/CA

## 7.7 Log de Execução

Para cada instância de processo $P$ o *Controlador* da MAE mantém um log contendo as seguintes informações:

– as mudanças de estado de $P$ – registro $newState(state, timestamp)$;

– os valores de entrada de $P$ – registro $inputParameters(\{(parameter, value)\})$;

– os valores de saída produzidos por $P$ – registro $outputParameters(\{(parameter, value)\})$;

– os recursos concretos utilizados para a concretização dos recursos abstratos correspondentes – registro $resourcesImplementedBy(\{(abstract_r, concrete_r)\})$;

- os recursos substituídos em $P$ – registro $resourcesSubstitutedBy(\{(original_r, substitute_r)\})$;
- os valores default utilizados para os parâmetros de $P$ – registro $defaultValues(\{(parameter, fromProcess, value)\}, state)$;
- o processo abstrato ao qual a instância de processo concreto corresponde – registro $abstractProcess(p')$;
- o mapeamento de recursos abstratos do processo abstrato para recursos abstratos do processo concreto que $P$ representa. Observe que esta informação, conforme já dito anteriormente, é armazenada no log de uma instância $P$, mas não em $P$ propriamente dita – registro $resourcesMapping(\{r_pAbstract, r_pConcrete\})$;
- o mapeamento de parâmetros entre processos, realizado tanto durante concretização de processos quanto na substituição de processos Ű- registro $parametersMapping(\{source_parameter, destination_parameter\})$;
- a instância de processo que está tratando a exceção temporal levantada por $P$ – registro $exceptionTreatedBy(P', state)$;
- a instância de processo que está substituindo $P$ – registro $instanceSubstitutedBy(P')$;
- a instância de processo utilizada para desfazer os efeitos de $P$ – registro $undoInstance(P')$.

A cada vez que uma instância de processo $P$ tem seu estado alterado, o *Controlador* da MAE acrescenta um registro $newState(state, timestamp)$ ao log de execução de $P$.

Quando uma instância $P$ alcança o estado *Running*, o *Controlador* insere no log o registro $inputParameters(\{(parameter, value)\})$. Da mesma forma, quando $P$ alcança o estado *PreparedToComplete*, o *Controlador* insere no log o registro $outputParameters(\{(parameter, value)\})$.

Quando uma instância $P$ de $p$ utiliza recursos concretos de recursos abstratos definidos em $p$, o log desta instância recebe do *Controlador* o registro $resourcesImplementedBy(\{(abstract_r, concrete_r)\})$, onde $abstract_r$ representa o recurso abstrato e $concrete_r$ o recurso concreto correspondente.

Quando ocorre substituição de recursos em $P$, o *Controlador* insere no log de $P$ o registro $resourcesSubstitutedBy(\{(original_r, substitute_r)\})$, onde $original_r$ representa o recurso original que estava indisponível e $substitute_r$ o recurso substituto escolhido.

PUC-Rio - Certificação Digital Nº 0115612/CA

O *Controlador* também insere no log o registro $defaultValues(\{(parameter, fromProcess, value)\}, state)$, quando valores default são assumidos para determinados parâmetros do processo representado pela instância $P$, onde $fromProcess$ indica o processo que deveria ter gerado o valor de *parameter*. Observe que este registro especifica uma lista de valores default assumidos.

Quando um processo concreto $p$ de um abstrato $p'$ é utilizado, escolhido a partir da resposta do mecanismo de tratamento de exceção por concretização, o *Controlador* insere o registro $abstractProcess(p')$ no log da instância $P$ de $p$. Lembre que instâncias não são geradas para processos abstratos.

Além disso, o *Controlador* também insere no log de $P$, caso $P$ tenha sido gerada a partir da concretização de um processo abstrato, o registro $resourcesMapping(\{r_pAbstract, r_pConcrete\})$, indicando o mapeamento de recursos abstratos do processo abstrato para recursos abstratos do processo concreto que $P$ representa. Nos casos de concretização e de substituição de processos, o *Controlador* insere no log de $P$ o registro $parametersMapping(\{source_parameter, destination_parameter\})$.

Também é gerenciada pelo *Controlador* qualquer utilização do mecanismo de tratamento de exceção para a flexibilização. Quando um processo de tratamento de exceção é usado para tratar uma exceção temporal levantada por $P$, o *Controlador* insere no log de $P$ o registro $exceptionTreatedBy(P', state)$, onde $P'$ representa a instância disparada para tratar da exceção levantada e *state* indica se a exceção foi de inicialização (*beginTimeout*) ou de terminação (*endTimeout*), podendo assumir como valores a cadeia de caracteres "*BeginTimed-out*" ou "*EndTimed-out*".

Se a instância $P$ de processo $p$ foi substituída por uma instância de um processo $p'$, através do mecanismo de tratamento de exceção para substituição de processos, o *Controlador* insere no log da instância $P$ o registro $instanceSubstitutedBy(P')$, onde $P'$ é a instância de $p'$.

Ainda, se o *Controlador* utiliza uma instância $P'$ para desfazer os efeitos gerados pela instância $P$, ele insere no log de $P$ um registro do tipo $undoInstance(P')$.

Por fim, vamos assumir que toda a informação inserida no log pelo *Controlador* possa posteriormente ser pesquisada pelo próprio *Controlador*, através de predicados computados (*builtins* do Jena) do tipo *get+<nome_registro>(parâmetros, variável_saída)*. O uso deste tipo de predicado será apresentado no Apêndice $B$.

PUC-Rio - Certificação Digital Nº 0115612/CA

A Figura 7.14, na Seção 7.9, apresenta uns logs construídos a partir da execução de uma determinada instância de processo composto.

## 7.8 Teste de Consistência

Além de garantir o comportamento transacional, o *Controlador* da MAE realiza testes de consistência para uma instância que passa pelo estado *BeginTimed-out* e adota valores default para os parâmetros de entrada. Este teste é realizado durante o processamento da mensagem *complete*.

Por outro lado, uma instância que passa pelo estado *EndTimed-out* e adota um valor default para um parâmetro de saída $s$ não provoca testes de consistência, já que tal valor default não pode gerar conflitos. De fato, a instância que deveria gerar o valor de $s$ é obviamente aquela para a qual o default de $s$ foi assumido.

A concretização não gera nenhum tipo de conflito. A troca de um processo abstrato por um processo concreto, ou de um recurso abstrato por um recurso concreto em tempo de execução apenas aponta uma flexibilidade maior na modelagem de processos, que podem agora ter a sua definição completada em tempo de execução.

A substituição de processos ou recursos também não causa conflitos. Por definição, processos modelados como semanticamente próximos, conforme visto na Seção 7.6.4, possuem parâmetros de entrada e de saída compatíveis, sendo os parâmetros do processo original $p$ mapeados nos parâmetros do processo substituto $p'$.

Além disso, se dois processos $p$ e $p'$ são semanticamente próximos, por definição, isso significa que trocar um processo pelo outro gera os mesmos efeitos. Desta forma, a condição sobre o resultado do processo alternativo, imposta através da propriedade *process:inCondition* de *process:Result* do OWL-S, deve necessariamente implicar a condição do resultado do processo original (o que deveria ter sido executado, conforme a definição do processo). Assim, fica garantido a priori que a execução do processo alternativo gera o mesmo resultado (ou um resultado compatível) que a execução do processo original e não é necessário nenhum mecanismo de tratamento de conflito.

Analogamente, recursos considerados semanticamente próximos também possuem efeitos semelhantes e, por isso, a substituição não gera conflitos.

Desfazer os efeitos de uma determinada instância de processo também, claramente, não causa conflito.

PUC-Rio - Certificação Digital Nº 0115612/CA

Dessa forma, a única forma de gerar conflito é através do uso de valores default para parâmetros de entrada.

Em mais detalhe, considere que um processo $p_1$ possua um parâmetro de saída $ps$, cujo valor é valor do parâmetro de entrada $pe$ de $p_2$. Se para $p_2$ está definida a instância da propriedade *p-ext:beginTimeout*, isso significa que, se $p_2$ não conseguir o valor vindo $pa$ antes que o tempo determinado por esta propriedade seja alcançado, o *Controlador* da MAE invoca o mecanismo de tratamento de exceção para uso de valores default, para obter um valor para o parâmetro $pe$ de $p_2$ correspondente.

Se o valor de $pe$ é obtido pelo mecanismo, a instância $P_2$ de $p_2$ passa do estado *BeginTimed-out* para o estado *Initiated*, no qual as suas pré-condições são novamente testadas. No entanto, enquanto $P_2$ segue sua execução pelo uso do valor default, a instância $P_1$ de $p_1$ segue sua execução normal, gerando então o valor de $ps$. Isso porque, considere, não havia sido definido o valor da propriedade *p-ext:endTimeout* para o processo $p_1$.

Desse modo, ao final da execução de $P_1$ e de $P_2$, o valor de $pe$ assumido por $P_2$ e o valor de $ps$ obtido por $P_1$ podem ser conflitantes. Neste caso, os efeitos da instância de processo $P_2$ deveriam ser refeitos de acordo com o valor real de $ps$.

Para refazer os efeitos de $P_2$, primeiramente poderia ser feito um *undo* desta instância e, posteriormente, ela poderia ser executada novamente, agora com o valor correto do parâmetro. A definição de um mecanismo para refazer ("*redo*") um processo é deixada para trabalhos futuros. Note que não se pode apenas ignorar os efeitos da instância $P_2$, porque eles podem ter sido refletidos em outros processos.

Ainda, se tivesse definido para o processo $p_1$ o valor da propriedade *p-ext:endTimeout*, a instância $P_1$ também poderia ter assumido um valor default para $ps$. No entanto, por definição, para cada parâmetro existe definido no máximo um valor default: diretamente, pela propriedade de processo *p-ext:defaultParameterValue*, ou por meio de *regras de suposição*.

Em suma, o *Controlador* da MAE é responsável, no momento do processamento da mensagem *complete*, por verificar em seu log se foi utilizado algum valor default para parâmetros, quando a instância se encontrava no estado *Initiated*. Para tal verificação, o *Controlador* busca no log da instância registros do tipo *defaultValues*, cujo valor de *state* seja igual à cadeia de caracteres "*BeginTimed-out*".

Desses registros, o *Controlador* extrai o identificador do processo (*fromProcess*) que deveria ter gerado o valor do parâmetro e busca no *blackboard* correspondente à instância que representou a execução mais

PUC-Rio - Certificação Digital Nº 0115612/CA

recente deste processo, conforme explicado na página 179, o valor deste parâmetro. Se os valores assumido e gerado são diferentes, então o *Controlador* detecta um conflito e processa a mensagem *abort* para a instância. Caso conflito não tenha sido detectado, a instância é de fato colocada no estado *PreparedToComplete*.

## 7.9 Exemplos de Execução

Considere que:

- o processo $a$ é a composição seqüencial de $b$ e $c$;
- o processo $b$ é a composição seqüencial de $d$ e $e$;
- o processo $e$ é a composição por *split* de $f$ e $g$; e
- os processos $c$, $d$, $g$ e $g$ são atômicos.

A Figura 7.13 apresenta a hierarquia de processos de exemplo formada a partir de $a$.

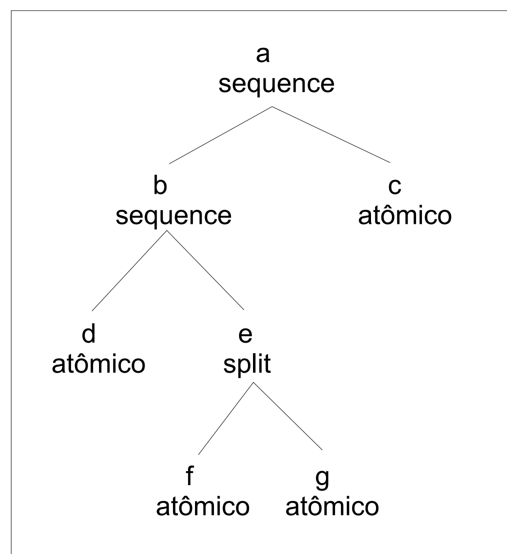


Figura 7.13: Hierarquia de processos de exemplo.

Considere que o processo $a$ foi inicializado, gerando a instância $A$. De acordo com a semântica definida por meio dos *triggers* relativos ao construtor *Sequence*, assim que as pré-condições de $A$ são satisfeitas, o *Controlador* inicializa $b$, gerando a instância $B$. Assim que $B$ é inicializado e as suas pré-condições são satisfeitas, o *Controlador* inicializa $d$, gerando a instância $D$.

PUC-Rio - Certificação Digital Nº 0115612/CA

Considere agora que $D$ está demorando para completar sua execução e que o valor determinado na propriedade *p-ext:endTimeout* de $b$ seja alcançado em $B$, levantando a exceção temporal de terminação. Neste caso, o *Controlador* recebe a mensagem *endTimeoutReached* para $B$ e, ao processá-la, coloca $B$ no estado *EndTimed-out* e solicita ao *Gerente de Ontologias* encontrar um processo que trate desta exceção levantada por $B$. Suponha que o *Gerente de Ontologias* encontre um ou mais processos para tratar a exceção. Ele envia então ao *Controlador* a lista de processos, através da mensagem *exceptionTreatmentProcessListFound*.

Suponha que, ao processar esta mensagem, o *Controlador* escolha o processo $b'$ para tratar da exceção de $b$, e gere uma instância $B'$ de $b$.

O *Controlador* coloca $B$ no estado *PreparedToAbort*, através do processamento de $abort(B)$. O *Controlador* também executa a mensagem *abort* para cada uma das subinstâncias de $B$, no caso apenas $D$, porque ela é a única que havia sido disparada, devido à semântica do construtor *Sequence*.

Quando $D$ alcança o estado *PreparedToAbort*, o *trigger* *InstanceRunning*$[3][\text{“}SEQUENCE\text{”}](B)$ não é ativado, no entanto, porque $B$ já não está mais no estado *Running*, mas sim no estado *PreparedToAbort*.

Por outro lado, quando $B$ passou para o estado *PreparedToAbort*, $B'$ foi colocada no lugar de $B$ na lista de subinstâncias de $A$. Assim, quando $B$ alcança esse estado, o *trigger* *InstanceRunning*$[3][\text{“}SEQUENCE\text{”}](A)$ não é ativado, porque na verdade ele considera as subinstâncias de $A$, mantidas na entrada $\{subInstances\}$ de $A$ (que agora contém $B'$ e não $B$).

Se $B'$ consegue terminar com sucesso, o *Controlador* passa $B'$ para o estado *PreparedToComplete*. O *trigger* *InstanceRunning*$[3][\text{“}SEQUENCE\text{”}](A)$ é ativado, já que $B'$ agora é subinstância de $A$ de interesse, e então o processo $c$ é inicializado, gerando uma instância $C$.

Suponha que $C$ seja abortada. Neste caso, o *trigger* *InstanceRunning*$[3][\text{“}SEQUENCE\text{”}](A)$ é ativado, fazendo com que o *Controlador* processe $abort(A)$. Ao processar esta mensagem, o *Controlador* coloca $A$ no estado *PreparedToAbort* e tenta colocar as suas subinstâncias também neste mesmo estado. $C$ já se encontra em *PreparedToAbort* e $B'$ está no estado *PreparedToComplete*. Portanto, $B'$ passa para o estado *PreparedToAbort*.

Cada instância de processo atômico colocada no estado *PreparedToAbort* tenta ter seus efeitos desfeitos. Se obtiver sucesso, passa para o estado *Undone*. Caso contrário, passa para o estado *Aborted*. Se todas as subinstâncias de uma instância alcançarem o estado *Undone*, esta

PUC-Rio - Certificação Digital Nº 0115612/CA

instância também alcança este estado. Caso contrário, ela alcança o estado *Aborted*.

A Figura 7.14 apresenta um esquema representativo dos logs formados durante a execução descrita anteriormente.

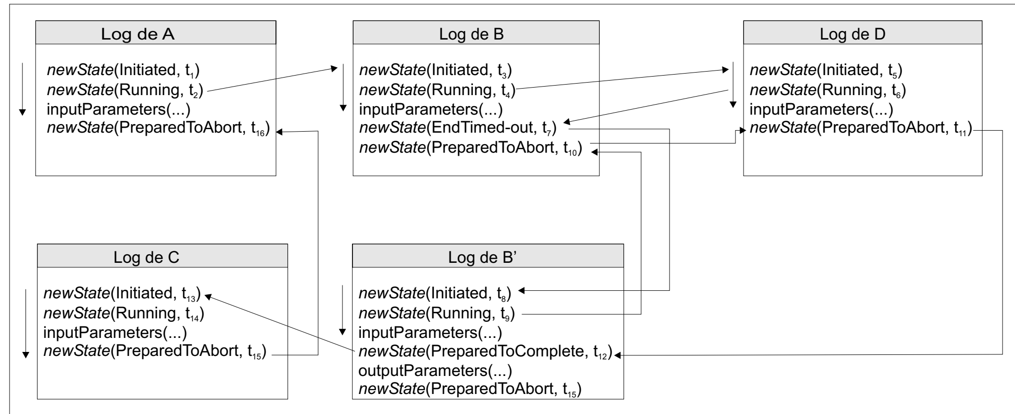


Figura 7.14: Exemplo de log relativo à primeira situação de execução descrita.

Imagine agora uma segunda situação envolvendo esta mesma hierarquia de processos. Considere que $D$ executou normalmente, sem que o valor da propriedade *p-ext:endTimeout* de $b$ fosse alcançado. Neste caso, o processo $e$ foi inicializado, segundo a semântica do construtor *Sequence* de $b$, resultando na instância $E$.

Considere agora que $E$ teve suas pré-condições avaliadas como verdadeiras, que foi colocada pelo *Controlador* no estado *Running*, e que suas subinstâncias $F$ e $G$ foram inicializadas. Assim que $F$ e $G$ alcançam o estado *Initiated*, pelo *trigger InstanceRunning*$[1][\text{“}SPLIT\text{”}](E)$, o *Controlador* coloca $E$ no estado *PreparedToComplete*.

Considere porém que a instância $F$ alcance o estado *PreparedToAbort* quando o *Controlador* processa a mensagem $abort(F)$. Pela semântica do construtor *Split*, nada acontece com a superinstância $E$, que já alcançou o estado *PreparedToComplete*.

A partir deste ponto, que ilustra o comportamento anômalo do construtor *SPLIT*, o exemplo segue normalmente.

Suponha agora uma terceira situação, novamente considerando a mesma hierarquia de processos formada a partir do processo $a$. Considere agora que $b$ é inicializado e depois $d$. No entanto, considere que a instância $D$ gerada a partir de $d$ alcança o valor determinado na propriedade *p-ext:endTimeout* de $d$, levantando a exceção temporal de terminação.

Suponha ainda que não existam processos de tratamento de exceção temporal para $d$, quando a exceção é levantada no estado *Running*, e que

PUC-Rio - Certificação Digital Nº 0115612/CA

não exista valores default para os parâmetros que $D$ deveria gerar.

Ao processar a mensagem *defaultParametersValueNotFound*, o *Controlador* então coloca $D$ no estado *PreparedToByPass*.

Considere que os valores dos parâmetros de saída de $D$ façam parte dos valores dos parâmetros de saída de $B$, ou seja, que exista um fluxo de saída de $D$ para $B$. Neste caso, o *Controlador*, ao processar a mensagem *instancePreparedToByPass*, solicita que o *Gerente de Ontologias* encontre os valores default dos parâmetros de $b$ que fazem parte do fluxo de dados relativo a $d$. Se o *Gerente de Ontologias* retornar a mensagem *defaultParametersValueFound*, o *Controlador* coloca nos parâmetros de $D$ os valores devolvidos e executa $force(D)$, não alterando o estado de $B$.

Quando $D$ alcança o estado *Forced*, o processo $e$ é inicializado, gerando uma instância $E$, segundo a semântica do construtor *Sequence* de $B$.

A partir deste ponto, que ilustra como uma instância alcança o estado *Forced*, o exemplo segue normalmente.

PUC-Rio - Certificação Digital Nº 0115612/CA

## 7.10 Resumo

Este capítulo apresentou inicialmente uma semântica operacional para a parte convencional de OWL-S, sob forma de uma máquina abstrata (MA).

A execução de uma instância $P$ de um processo $p$ atravessa uma seqüência de estados, conforme o diagrama de transição de estados definido no capítulo. As transições de estado não são definidas por funções, como nos autômatos tradicionais, mas através de *triggers*, que caracterizam a semântica dos processos atômicos e dos processos compostos. Os *triggers* são definidos de tal forma a garantir um comportamento transacional de uma instância de processo.

Em seguida, o capítulo expandiu a semântica operacional para incorporar os mecanismos de tratamento de exceção para flexibilização, sob forma de uma máquina abstrata estendida (MAE).

A MAE possui um componente a mais, o *Gerente de Ontologias*, fundamental para implementar o mecanismo de tratamento de exceção proposto. A MAE incorpora ainda um novo diagrama de transição de estados, com novos estados para acomodar a semântica do mecanismo de tratamento de exceção para a flexibilização da execução. Na MAE, os *triggers* são redefinidos de tal forma a garantir um novo comportamento transacional de uma instância, que leva em consideração questões relativas a execução flexível.