6
Uma Ontologia de Processos e Recursos

Este capítulo apresenta a *ontologia de processos e recursos*, chamada aqui de ontologia *pr*, que é a base do mecanismo de tratamento de exceção. Este mecanismo é apresentado nos capítulos seguintes. A Seção 6.1 apresenta um resumo das metodologias para a modelagem de ontologias, uma das quais foi escolhida para este trabalho. A Seção 6.2 lista os requisitos levantados para a modelagem da ontologia *pr* criada. A Seção 6.3 apresenta as estratégias utilizadas para a definição da ontologia *pr*. A Seção 6.4 detalha as extensões propostas para a ontologia de processos de OWL-S. Por fim, a Seção 6.5 define as novas classes, propriedades e regras necessárias à definição da ontologia *pr*. A Seção 6.6 apresenta um resumo deste capítulo.

6.1
Metodologias para a Modelagem de Ontologias

Ontologias têm sido largamente utilizadas em diversas áreas de conhecimento [26]. Apesar de estar sendo utilizado sob diversas denotações, o termo *ontologia*, no seu uso mais comum em Ciências da Computação, refere-se a um vocabulário específico, usado para descrever um certo domínio do conhecimento, acompanhado de um conjunto de assertivas explicitando o significado dos termos deste vocabulário [56].

Sob este aspecto, ontologias podem ser utilizadas, por exemplo, para explicitar a definição de um domínio de conhecimento e compartilhá-la entre pessoas e agentes de software, possibilitando o reuso [87].

Diversas são as metodologias existentes na literatura para o desenvolvimento de ontologias, mas na prática nem sempre elas são seguidas porque normalmente não se adequam perfeitamente às necessidades da aplicação para a qual a ontologia está sendo construída. A seguir, resumiremos as metodologias mais conhecidas.

A metodologia de Uschold e King [115] divide o processo de desenvolvimento de ontologias em 4 etapas. Na primeira etapa, é identificado

o propósito da ontologia. Na segunda etapa a ontologia é construída, definindo-se seus conceitos e propriedades. Na terceira etapa, é feita uma avaliação da ontologia, na tentativa de verificar se ela atende a todos os requisitos da aplicação que vai utilizá-la, ou seja, se ela está atendendo ao propósito para o qual foi definida. Na última etapa, a ontologia é documentada.

A metodologia de Grüninger e Fox [55] divide o processo de desenvolvimento de ontologias em 6 etapas. Na primeira etapa, são descritos os problemas que a ontologia deve atender. Na segunda etapa, as questões de competência (questões que a ontologia deve ser capaz de responder) são informalmente formuladas, normalmente em linguagem natural. Na terceira etapa, a ontologia é formalmente especificada na linguagem escolhida. Na quarta etapa, as questões de competência definidas na segunda etapa são formalmente expressas. Na quinta etapa, são definidos em linguagem formal os axiomas que fornecem semântica aos termos da ontologia. Na última etapa, é verificada a completude da ontologia, ou seja, se ela de fato atende a todas as questões de competência formuladas.

A metodologia de Gómez-Perez [52], conhecida como METHONTOLOGY, desenvolvida no laboratório de Inteligência Artificial da Universidade de Madri, divide o processo de desenvolvimento de ontologias de acordo com o tipo de atividade a ser desenvolvida. Por exemplo, dentro do aspecto de gerenciamento de projeto, estão incluídas as atividades de planejamento, controle de atividades e garantia de qualidade. Dentre as atividades voltadas ao desenvolvimento, incluem-se a especificação da ontologia, a estruturação do domínio de conhecimento correspondente, a formalização dos conceitos que a ontologia deve incluir, a implementação e a manutenção da ontologia. Atividades de suporte nesta metodologia também são desempenhadas em paralelo às atividades de desenvolvimento, e incluem: aquisição de conhecimento, avaliação, integração e documentação das ontologias.

A metodologia de Noy e McGuinness [87] define um processo de desenvolvimento iterativo, no qual uma versão inicial da ontologia é criada e vai sendo progressivamente refinada, conforme as necessidades que vão surgindo. O passo inicial desta metodologia é a determinação do domínio da ontologia e da análise de ontologias já existentes no domínio requerido. O passo seguinte consiste em definir quais os termos mais importantes da ontologia, criá-los como classes e suas propriedades e organizá-los em uma hierarquia. O último passo de um ciclo é a criação das instâncias. No entanto, o processo de desenvolvimento da ontologia é iterativo e, portanto, qualquer

um dos passos pode ser repetido um número arbitrário de vezes.

Nesta tese, o sistema de gerência de workflows é um sistema governado por ontologias. Mais precisamente, a flexibilização da execução de processos, evitando que a execução seja interrompida em momentos em que ocorre indisponibilidade de recursos ou falta de informação a respeito do valor de uma variável da qual esta execução dependa, e ainda permitindo adiar a completa modelagem de processos para a fase de execução, está baseada no uso de ontologias. Para o desenvolvimento das ontologias subjacentes à flexibilização, foi adotada a metodologia sugerida em [87].

Nas seções seguintes, o termo "processo abstrato" será utilizado para se referir à classe (ou a instâncias dela) *SimpleProcess* da ontologia de processos de OWL-S. De forma semelhante, o termo "processo concreto" referenciará as classes (ou a instâncias delas) *AtomicProcess* e *CompositeProcess* desta mesma ontologia.

6.2 Requisitos para uma Ontologia de Processos e Recursos

Sistemas de gerência de workflow convencionais normalmente interpretam rigidamente a definição estática dos processos (ou seja, dos workflows), proibindo, durante a execução, qualquer tipo de desvio não previsto na modelagem.

No intuito de tornar a execução de um processo mais flexível e evitar longos períodos de espera, este trabalho apresenta um mecanismo de tratamento de exceção baseado em ontologias. Tal mecanismo:

- permite que o projetista defina abstratamente um processo em tempo de modelagem e que o sistema faça uma escolha dinâmica (semi−)automática, em tempo de execução, do processo (ou recurso) concreto associado ao abstrato definido no modelo estático do processo;
- distingue entre diversos tipos de exceção, desde as exceções temporais até as exceções por cancelamento, conforme definido na Seção 5.3.

A abordagem de flexibilização da execução empregada nesta tese, através do mecanismo de tratamento de exceção, parte do princípio de que a definição estática do processo, que determina a sua composição, está associada a um item de informação adicional. Este item de informação é baseado na *ontologia pr de processos e recursos*, à qual cada máquina de execução tem acesso através de um serviço de consulta oferecido pelo

repositório no qual a ontologia está inserida. O termo *pr* será utilizado para denotar esta ontologia no texto, por analogia com a noção de *namespace*. Aliás, todas as ontologias aqui definidas serão descritas como tendo o nome das abreviações de seus *namespaces*, por motivos de simplificação.

A ontologia *pr* contém as definições de classes e propriedades. As instâncias de processos atômicos e as dos processos complexos que têm alto grau de reuso para um domínio de aplicação, além das instâncias de recursos e suas propriedades, estão definidas em uma biblioteca de processos e recursos, à qual é dado o nome de *biblioteca lib*.

Para definir um processo de acordo com o vocabulário da ontologia *pr*, o projetista pode utilizar essa biblioteca *lib*, reaproveitando dela as instâncias de processos, recursos e suas propriedades já definidas. Vale ressaltar que este processo é definido em um outro *namespace*, que se refere a uma ontologia de aplicação, que então importa a biblioteca *lib*, que por sua vez importa a ontologia *pr* onde os termos estão definidos.

O projetista deve, antes de iniciar a criação de um processo de uma aplicação específica, definir na biblioteca *lib* os processos atômicos e compostos e os recursos a eles necessários que julgar serem reaproveitáveis para diversos processos. Como o procedimento de criação da biblioteca é iterativo, a elas podem ser acrescidas novas instâncias de processos, recursos e propriedades que o projetista julgar necessárias. Por exemplo, se um processo complexo está sendo utilizado na composição de diversos processos para as aplicações, esta definição pode ser levada dos processos onde ocorrem para a biblioteca *lib*, permitindo o reaproveitamento e evitando a sua repetição.

Considere, a partir de agora, que o *namespace app* refere-se à ontologia de aplicação na qual está definido o processo que um projetista deseja que seja executado.

Para atender as necessidades do mecanismo de tratamento de exceção proposto para a flexibilização da execução, a ontologia *pr* deve satisfazer os seguintes requisitos:

R_1 Definir processos abstratos e concretos.

R_2 Definir processos atômicos.

R_3 Definir formas de compor processos.

R_4 Definir recursos abstratos e concretos.

R_5 Dado um processo abstrato, definir quais são os processos concretos que o implementam e qual o peso do relacionamento entre o processo abstrato e cada um destes processos concretos.

R_6 Dado um recurso abstrato, definir quais são os recursos concretos que o implementam e qual o peso do relacionamento entre o recurso abstrato e cada um dos recursos concretos correspondentes.

R_7 Definir o tempo máximo que um processo pode esperar para iniciar sua execução (*beginTimeout*) e para terminá-la (*endTimeout*), uma vez iniciada.

R_8 Dado um processo, definir qual(is) o(s) processo(s) que trata(m) uma exceção temporal (de inicialização ou de terminação) deste processo e qual o peso deste relacionamento de tratamento de exceção.

R_9 Dado um processo atômico, definir qual(is) o(s) processo(s) que trata(m) uma exceção deste processo levantada para desfazer os efeitos de uma execução abortada (exceção por cancelamento) e qual o peso deste relacionamento.

R_{10} Dado um parâmetro associado a um processo, definir qual o valor default a ele associado.

R_{11} Definir se a flexibilização da execução pode ocorrer, ou não, e de que forma deverá ser tratada. Isso é necessário porque nem sempre um processo pode ser flexibilizado por substituição ou pelo uso de valor default.

R_{12} Dado um processo, definir quais os processos que possuem algum relacionamento de proximidade semântica com ele e qual o peso deste relacionamento.

R_{13} Definir o custo de execução de um processo.

R_{14} Dado um processo, definir quais são os recursos necessários para a sua execução e em qual quantidade (no caso de recursos abstratos). Quando um recurso concreto é definido como necessário para a execução de um determinado processo, não existe quantidade associada, porque cada recurso concreto é único.

R_{15} Dado um recurso, definir quais são os recursos que possuem algum relacionamento de proximidade semântica com ele e qual o peso deste relacionamento.

R_{16} Definir o custo de utilização de um recurso.

R_{17} Definir quando um recurso está ou não disponível.

R_{18} Definir o mapeamento de recursos abstratos entre um processo abstrato e um processo concreto correspondente.

R_{19} Definir o mapeamento de parâmetros entre dois processos, sendo um deles abstrato e o outro concreto, ou sendo os dois abstratos ou os dois concretos.

R_{20} Definir quando um processo pode ser delegado para a execução por outra máquina de execução.

Neste sentido, a ontologia *pr* de processos e recursos é independente de domínio e, portanto, pode ser utilizada para vários domínios de conhecimento distintos. Basta que o projetista crie uma ontologia de aplicação com base em instâncias pertinentes ao domínio específico que necessite representar. Estas instâncias, conforme a definição apresentada, fazem parte da biblioteca *lib*. A ontologia de aplicação é referenciada pelo *namespace app*.

6.3 Estratégias para Definição da Ontologia de Processos e Recursos

Pelo estudo apresentado nos Capítulos 3 e 4, foi possível observar que a ontologia de processos de OWL-S possui grande parte das classes e dos relacionamentos necessários à ontologia *pr* que está sendo criada. A ontologia de processos de OWL-S cobre processos atômicos e compostos, e cobre processos simples como forma de abstração. Portanto, ela atende aos requisitos R_1, R_2 e R_3 (este último mesmo que de forma limitada) apresentados, e parcialmente ao requisito R_5 (parcialmente porque na ontologia de processos de OWL-S não existe a noção de peso de um relacionamento).

Além disso, a ontologia de recursos de OWL-S, conforme apresentado no Capítulo 4, descreve recursos utilizados por processos. Esta ontologia pode ser reusada para a definição dos recursos na ontologia *pr*, porque atende parcialmente ao requisito R_4. Parcialmente porque na ontologia original de OWL-S não existe a distinção entre recursos abstratos e recursos concretos.

Uma vez tendo analisado o propósito da ontologia *pr*, resumido nos requisitos R_1 a R_{20}, e as ontologias existentes que poderiam ser reusadas,

quais sejam, as ontologias de processo e de recurso de OWL-S, o passo seguinte é a criação da ontologia *pr* em si.

O Quadro 2 apresenta uma lista das abreviações criadas para cada um dos *namespaces* utilizados nas definições das ontologias. A partir deste ponto, cada ontologia será identificada pela abreviação do seu *namespace*.

- *process*: abreviação para
`http://www.daml.org/services/owl-s/1.1/Process.owl#`,
o *namespace* da ontologia de processos de OWL-S;

- *p-ext*: abreviação para
`http://www.inf.puc-rio.br/~tati/tese/Process-extended.owl#`,
o *namespace* da ontologia de processos de OWL-S estendida;

- *r*: abreviação para
`http://www.daml.org/services/owl-s/1.1/Resource.owl#`,
o *namespace* da ontologia de recursos de OWL-S;

- *r-ext*: abreviação para
`http://www.inf.puc-rio.br/~tati/tese/Resource-extended.owl#`,
o *namespace* da ontologia de recursos de OWL-S estendida;

- *pr*: abreviação para
`http://www.inf.puc-rio.br/~tati/tese/prOntology.owl#`,
o *namespace* da ontologia *pr*;

- *lib*: abreviação para
`http://www.inf.puc-rio.br/~tati/tese/prLibrary.owl#`,
o *namespace* da biblioteca de processos e recursos;

- *app*: abreviação para o *namespace* da ontologia de aplicação.

Quadro 2 − Abreviações para os *namespaces* utilizados.

Essas abreviações são possíveis em OWL através de declarações de *ENTITY*s [125]. Com elas, as definições de *namespaces* tornam-se simplificadas e alterações feitas às entidades propagam-se por toda a ontologia, consistentemente.

Para complementar o requisito R_3, incluindo um novo construtor de processos, atender parcialmente os requisitos R_8 e R_9, e também para

atender aos requisitos R_7, R_{10}, R_{11}, R_{13} e R_{20}, a ontologia de processos de OWL-S foi estendida. Esta ontologia estendida, chamada *p-ext*, importa a ontologia de processos *process* de OWL-S.

Para atender aos requisitos R_{16} e R_{17}, e parcialmente aos requisitos R_4 e R_6, foi criada uma extensão à ontologia de recursos de OWL-S, chamada aqui de ontologia *r-ext*.

Para atender integralmente aos requisitos R_5, R_6, R_8 e R_9, e também aos requisitos R_{12}, R_{14}, R_{15}, R_{18} e R_{19}, classes e propriedades foram criadas na ontologia *pr*.

Desse modo, a ontologia *pr* (ver Figura 6.1):

- importa a ontologia *p-ext*, que por sua vez importa a ontologia de processos de OWL-S e a estende;
- importa a ontologia *r-ext*, que por sua vez importa a ontologia de recursos de OWL-S e a estende;
- contém propriedades que relacionam processos com recursos (*requires* e *requiredBy*), e classes (*Relation_for_value*, *ParameterMap* e *ResourceMap*) e propriedades que permitem o relacionamento semântico entre instâncias de classes de processos e de recursos.

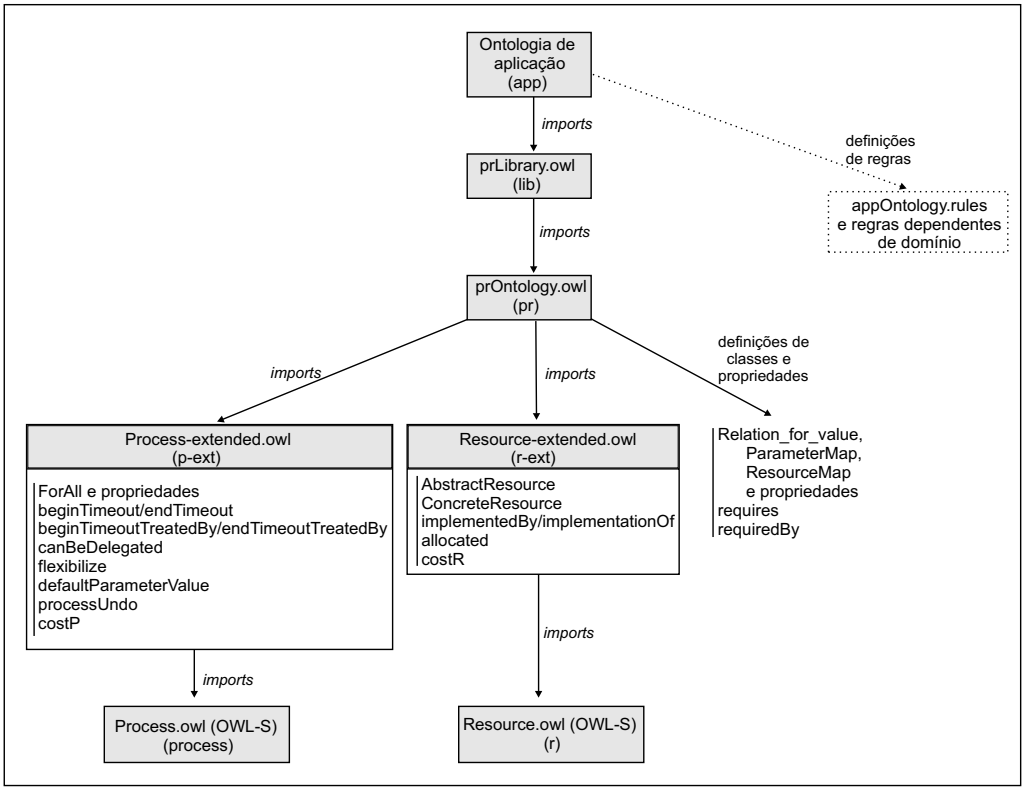

Figura 6.1: Representação ilustrativa da composição da ontologia *pr*.

Na Figura 6.1, a parte inferior dos retângulos relativos às ontologias de processo e de recurso estendidas apresentam as classes e propriedades nelas definidas.

A biblioteca de processos e recursos, *lib*, é uma ontologia que contém um conjunto de instâncias das classes e propriedades que serão compartilhadas por um conjunto de aplicações, visando reuso, portanto. Ela importa a ontologia *pr*.

A ontologia de aplicação, *app*, contém o conjunto de instâncias das classes e propriedades de *pr* específicos de uma aplicação. Ela importa a ontologia *lib* e contém regras, dependentes do domínio ou não, para garantir a consistência das definições dos processos.

Em adição, à ontologia *app* está associado um conjunto de regras independentes de domínio, definidas em `http://www.inf.puc-rio.br/~tati/tese/appOntology.rules`. Regras dependentes de domínio também podem ser criadas, o que deve ser feito em um arquivo de regras separado, mantendo a generalidade das regras independentes de domínio.

Naturalmente, tanto a ontologia *lib* quanto a *app* variam conforme a aplicação ou conjunto de aplicações.

O Apêndice B apresenta a ontologia de processos estendida de OWL-S, a ontologia de recursos estendida de OWL-S, a ontologia *pr* completa e as regras associadas à ontologia de aplicação. O Apêndice C contém uma biblioteca *lib* para um determinado domínio de aplicação e o código OWL de um processo, chamado *CoastalAreaOilCleaning*, neste domínio de exemplo.

6.4 Extensões Básicas da Ontologia de Processos e de Recursos de OWL-S

Recorde que o Capítulo 5 apresentou a definição das exceções propostas, enquanto que o Capítulo 4 resumiu tanto a ontologia de processos quanto a ontologia de recursos de OWL-S. Esta seção introduz a sintaxe das extensões propostas a estas ontologias, deixando para o Capítulo 7 a definição da semântica das extensões.

6.4.1 Construtor ForAll

Esta seção define as classes *ForAll* e *ForAllVar*, e as propriedades *hasForAllVar*, *fromList* e *do*, pertencentes à ontologia *p-ext*.

Conforme brevemente comentado na Seção 3.3, a especificação da linguagem OWL-S ainda possui várias lacunas. Em particular, o *ServiceModel* da ontologia de serviços de OWL-S não inclui alguns construtores bastante úteis para a descrição de processos mais complexos.

Dentro deste contexto, esta seção define o construtor *p-ext:ForAll* na ontologia *p-ext* como subclasse de *ControlConstruct*, especificando um comportamento semelhante ao do construtor *Split-Join*, onde um mesmo processo (ou uma composição de processos) é disparado para cada um dos elementos de uma lista de objetos. Define ainda, na ontologia *p-ext*, a classe *ForAllVar*, analogamente a *ResultVar* da ontologia de processos de OWL-S, para descrever a variável utilizada para representar cada elemento da lista. Assim, a instância de *ForAllVar* recebe a cada nova chamada de processo o valor do objeto corrente da lista.

Assim, a propriedade *p-ext:hasForAllVar* possui como domínio *ForAll*. Com o mesmo domínio, a propriedade *p-ext:fromList* indica sobre qual lista de objetos as execuções são realizadas. Por fim, a propriedade *p-ext:do*, que possui como contra-domínio a classe *process:ControlConstruct*, permite representar os construtores (ou composições de processos) a serem invocados para cada elemento da lista.

Note que o construtor *p-ext:ForAll* não é uma iteração, já que vários processos serão executados simultaneamente, cada um para um elemento da lista de objetos especificada. O construtor *PARLOOP*, semelhante ao *ForAll* aqui apresentado, é definido na linguagem de definição de workflows apresentada em [15].

```
<owl:Class rdf:ID="ForAll">
  <rdfs:subClassOf rdf:resource="&process;ControlConstruct"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasForAllVar"/>
      <owl:cardinality
        rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#fromList"/>
      <owl:cardinality
        rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
```

```
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#do"/>
      <owl:cardinality
        rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="ForAllVar">
  <rdfs:subClassOf rdf:resource="&process;Parameter"/>
  <owl:disjointWith rdf:resource="&process;Input"/>
  <owl:disjointWith rdf:resource="&process;Output"/>
  <owl:disjointWith rdf:resource="&process;Local"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasForAllVar">
  <rdfs:label>hasForAllVar</rdfs:label>
  <rdfs:domain rdf:resource="#ForAll"/>
  <rdfs:range rdf:resource="#ForAllVar"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="fromList">
  <rdfs:domain rdf:resource="#ForAll"/>
  <rdfs:range rdf:resource="&shadow-rdf;List"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="do">
  <rdfs:domain rdf:resource="#ForAll"/>
  <rdfs:range rdf:resource="&process;ControlConstruct"/>
</owl:ObjectProperty>
```

Quadro 3 – Construtor *FORALL* e suas propriedades pertencentes ao *namespace p-ext*.

6.4.2 Classes AbstractResource e ConcreteResource e Propriedades implementedBy e implementationOf

Esta seção define as classes *AbstractResource* e *ConcreteResource*, e as propriedades *implementedBy* e *implementationOf*, pertencentes à ontologia *r-ext*.

As classes *r-ext:AbstractResource* e *r-ext:ConcreteResource* representam, de forma análoga às classes *process:SimpleProcess* e *process:AtomicProcess* e *process:CompositeProcess*, recursos abstratos e

recursos concretos, respectivamente. Um recurso é *abstrato* quando não pode ser diretamente utilizado, e é *concreto* quando é um recurso específico, único. Intuitivamente, recursos abstratos são semelhantes a tipos de recursos (pás, cadeiras, um tipo de software) e recursos concretos são "implementações" destes recursos (uma determinada pá, uma determinada cadeira, um determinado software, e assim por diante). A classe *r:Resource* original da ontologia de recursos de OWL-S passa então a ser definida como a união destas duas outras classes.

Dessa forma, a propriedade *p-ext:implementedBy* relaciona um recurso abstrato com um ou mais recursos concretos. De forma inversa, a propriedade *r-ext:implementationOf* define qual recurso abstrato um recurso concreto "implementa".

O Quadro 4 apresenta a definição dessas classes e propriedades em OWL.

```
<owl:Class rdf:about="&r;Resource">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AbstractResource"/>
    <owl:Class rdf:about="#ConcreteResource"/>
  </owl:unionOf>
</owl:Class>

<owl:Class rdf:ID="AbstractResource"/>
<owl:Class rdf:ID="ConcreteResource"/>

<owl:ObjectProperty rdf:ID="implementedBy">
  <rdfs:domain rdf:resource="#AbstractResource"/>
  <rdfs:range rdf:resource="#ConcreteResource"/>
<owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="implementationOf">
  <rdfs:domain rdf:resource="#ConcreteResource"/>
  <rdfs:range rdf:resource="#AbstractResource"/>
  <rdf:inverseOf rdf:resource="#implementedBy"/>
<owl:ObjectProperty>
```

Quadro 4 – Classes *AbstractResource* e *ConcreteResource* e propriedades relacionadas, pertencentes ao *namespace r-ext*.

6.4.3
Propriedades requires, requiredBy e allocated

Esta seção define as propriedades *requires* e *requiredBy*, pertencentes à ontologia *pr*, e a propriedade *allocated*, pertencente à ontologia *r-ext*.

A propriedade *pr:requires*, apresentada no Quadro 5, possibilita que um processo indique quais recursos necessita para a sua execução.

A propriedade inversa *pr:requiredBy* facilita a descoberta de processos que utilizam um determinado recurso.

A propriedade *r-ext:allocated* indica se um recurso está ou não alocado para ser utilizado por um determinado processo.

Observe que a propriedade *r-ext:allocated* refere-se a recursos e não a valores de parâmetros utilizados por instâncias de processos. Veremos, nos capítulos seguintes, que os valores de parâmetros são acessados, no caso da máquina abstrata sem extensões, pela máquina de execução, para a execução de qualquer instâncias sob o seu controle. No caso da arquitetura distribuída, os valores dos parâmetros associados a uma determinada instância são acessados pela máquina de execução desta instância, para a execução de quaisquer outras instâncias sob o seu controle. Os demais valores poderão, ainda assim, serem obtidos por meio do gerente central, como veremos no Capítulo 8.


```
<owl:ObjectProperty rdf:ID="requires">
  <rdfs:domain rdf:resource="&process;Process"/>
  <rdfs:range rdf:resource="&r;Resource"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="requiredBy">
  <rdfs:domain rdf:resource="&r;Resource"/>
  <rdfs:range rdf:resource="&process;Process"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="allocated">
  <rdfs:domain rdf:resource="ConcreteResource"/>
  <rdfs:range rdf:resource="&xsd;boolean"/>
</owl:DatatypeProperty>

<owl:Class rdf:about="#ConcreteResource">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#allocated"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
      1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Quadro 5 − Propriedades *requires* e *requiredBy*, pertencentes ao *namespace pr*, e propriedade *allocated*, do *namespace r-ext*.

6.5 Extensões Voltadas para a Flexibilização

6.5.1 Propriedades costR e costP

Esta seção define a propriedade *costR*, pertencente ao *namespace r-ext*, e a propriedade *costP*, pertencente ao *namespace p-ext*.

A propriedade *r-ext:costR* define o custo de utilização de um recurso. A propriedade de processo *p-ext:costP* indica o custo de execução de um processo. Estas duas propriedades estão definidas conforme apresentado no Quadro 6.

```
<owl:DatatypeProperty rdf:ID="costR">
  <rdfs:domain rdf:resource="&r;Resource"/>
  <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="costP">
  <rdfs:domain rdf:resource="&process;Process"/>
  <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>
```

Quadro 6 – Propriedades *costR* e *costP*, pertencentes, respectivamente, aos *namespace r-ext* e *namespace p-ext*.

Essas propriedades são utilizadas pelo modelo de custo empregado pelo mecanismo de tratamento de exceção, conforme será apresentado na Seção 7.6.4.

Vale ressaltar que o custo é definido tanto para processos e recursos concretos quanto para processos e recursos abstratos. Por exemplo, o custo de um processo abstrato pode ser definido como o maior valor de custo de um processo concreto associado, o menor valor ou a média dos custos de todos os processos concretos associados. Deixamos esta definição para o projetista da ontologia.

6.5.2 Propriedade flexibilize

Esta seção define a propriedade *flexibilize* pertencente à ontologia *p-ext*.

Por definição, são sempre tratadas:

– as exceções levantadas quando a execução de uma instância de um processo atinge um processo ou recurso abstrato;

– as exceções temporais de inicialização e de terminação;

– a exceção por cancelamento, no caso de instância de processo atômico.

Por outro lado, o tratamento de exceções para a substituição de processos ou recursos e para o uso de valores default de parâmetros pode ser ou não permitido para um determinado processo. Isso porque pode haver casos tais que a substituição não seja aconselhada ou que o uso de valores default para parâmetros seja indesejado.

Dessa forma, a propriedade *p-ext:flexibilize* indica qual o tipo de exceção que pode ser tratada para o processo em questão, excluindo as que são obrigatoriamente tratadas.

A propriedade *p-ext:flexibilize* aplica-se a dois tipos distintos de instâncias. Primeiro, ela se aplica a instâncias da classe *process:Process* de OWL-S, indicando qual tipo de condição pode ser tratada. Segundo, ela aplica-se a instâncias da classe *process:Perform* de OWL-S, através da qual processos são invocados dentro de uma composição de um outro processo. Dessa forma, a um processo pode ser permitido o emprego de um tipo de flexibilização (através do tratamento das diversas exceções, segundo a hierarquia definida), quando em uma composição, e de outro tipo, quando em outra composição, já que a propriedade aplica-se também à classe *process:Perform*. No entanto, se esta propriedade estiver definida diretamente para *process:Process*, então o seu valor deve ser respeitado quando o processo é inserido em outra composição, mesmo que para esta composição também seja definida esta propriedade. Maiores detalhes podem ser encontrados na Seção 7.6.2.

A propriedade *p-ext:flexibilize*, apresentada no Quadro 7, é de fundamental importância para o mecanismo de tratamento de exceção atuar durante a execução de uma instância de processo, como será mostrado no Capítulo 7.

```
<owl:DatatypeProperty rdf:ID="flexibilize">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="&process;Process"/>
        <owl:Class rdf:about="&process;Perform"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
```

```
    <rdfs:range>
      <owl:DataRange>
        <owl:oneOf rdf:parseType="Resource">
          <rdf:first rdf:datatype="&xsd;string">
            by_substitution
          </rdf:first>
          <rdf:rest rdf:parseType="Resource">
            <rdf:first rdf:datatype="&xsd;string">
              by_default
            </rdf:first>
            <rdf:rest rdf:parseType="Resource">
              <rdf:first rdf:datatype="&xsd;string">
               all
              </rdf:first>
              <rdf:rest rdf:resource="&rdf;nil"/>
            </rdf:rest>
          </rdf:rest>
        </owl:oneOf>
      </owl:DataRange>
    </rdfs:range>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  </owl:DatatypeProperty>
```

Quadro 7 – Propriedade *flexibilize* pertencente ao *namespace p-ext*.

Vale mencionar que essa propriedade é uma *FunctionalProperty* para evitar que mais do que um valor seja definido para ela, gerando inconsistência. Para permitir que um processo seja flexibilizado tanto através do tratamento de exceção levantada para o uso de valor default quanto levantada para a substituição, foi definido o valor "*all*" para esta propriedade.

6.5.3
Propriedades para o Tratamento de Exceção

Esta seção define as propriedades *beginTimeout*, *endTimeout*, *beginTimeoutTreatedBy*, *endTimeoutTreatedBy*, *defaultParameterValue* e *processUndo* pertencentes à ontologia *p-ext*. Estas propriedades são de fundamental importância para a flexibilização da execução de uma instância de processo, como será discutido no capítulo seguinte.

As propriedades *p-ext:beginTimeout* e *p-ext:endTimeout*, apresentadas no Quadro 8, permitem definir *timeouts* associados diretamente a processos (e não a seus construtores), para a definição das situações de exceção temporal de inicialização e de terminação, respectivamente. Portanto,

estas propriedades distinguem *timeout* alcançado quando o processo foi inicializado, mas ainda não começou execução, e *timeout* quando o processo está em execução e não alcançou seu fim, gerando os valores de saída. Os valores de *timeout* são definidos em minutos.

```
  <owl:ObjectProperty rdf:ID="beginTimeout">
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;AtomicProcess"/>
          <owl:Class rdf:about="&process;CompositeProcess"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range rdf:resource="&xsd;float"/>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="endTimeout">
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="&process;AtomicProcess"/>
          <owl:Class rdf:about="&process;CompositeProcess"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range rdf:resource="&xsd;float"/>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>
```

Quadro 8 – Propriedades *beginTimeout* e *endTimeout* pertencentes ao *namespace p-ext*.

Vale ressaltar que não faz sentido definir *timeouts* para processos abstratos e, por isso, o domínio destas duas propriedades é a união das classes relativas aos processos atômicos e compostos. Além disso, as duas são do tipo *Functional Property* porque não pode existir para um mesmo processo mais do que um valor de *p-ext:beginTimeout* e mais do que um valor de *p-ext:endTimeout*.

As propriedades *p-ext:beginTimeoutTreatedBy* e *p-ext:endTimeoutTreatedBy* determinam o relacionamento entre um processo OWL-S e outros processos OWL-S que tratem de exceções temporais levantadas pelo primeiro. Estas propriedades não são aplicadas sobre processos abstratos (instâncias da

classe *SimpleProcess*). A definição destas propriedades é apresentada no Quadro 9.

```
<owl:ObjectProperty rdf:ID="beginTimeoutTreatedBy">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="&process;AtomicProcess"/>
        <owl:Class rdf:about="&process;CompositeProcess"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="&process;AtomicProcess"/>
        <owl:Class rdf:about="&process;CompositeProcess"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="endTimeoutTreatedBy">
   <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="&process;AtomicProcess"/>
        <owl:Class rdf:about="&process;CompositeProcess"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="&process;AtomicProcess"/>
        <owl:Class rdf:about="&process;CompositeProcess"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
</owl:ObjectProperty>
```

Quadro 9 – Propriedades *beginTimeoutTreatedBy* e *endTimeoutTreatedBy* pertencentes ao *namespace p-ext*.

A propriedade *p-ext:processUndo*, cujo domínio é a classe de processos atômicos, especifica um processo que desfaz os efeitos de um processo atômico p, segundo o mecanismo de tratamento de exceção por cancelamento. Isso é necessário quando a execução de p é abortada (alcança o estado *PreparedToAbort* da Figura 7.6, no capítulo seguinte). Pode existir

mais do que um processo de *undo* associado a um determinado processo. O Quadro 10 contém a definição desta propriedade em OWL.

```
<owl:ObjectProperty rdf:ID="processUndo">
  <rdfs:domain rdf:resource="&process;AtomicProcess"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="&process;AtomicProcess"/>
        <owl:Class rdf:about="&process;CompositeProcess"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
</owl:ObjectProperty>
```

Quadro 10 – Propriedade *processUndo* pertencente ao *namespace p-ext*.

Na extensão definida sobre a ontologia de processos de OWL-S, a propriedade *p-ext:defaultParameterValue*, cujo domínio é a classe *Parameter*, permite determinar um valor default para um parâmetro de um processo. No entanto, este valor também pode ser dependente de valores correntes da execução e, portanto, determinado através de *regras de suposição* de acordo com o domínio de aplicação.

Conforme já apresentado no Capítulo 5, *regras de suposição*, quando definidas, agregam ao parâmetro o poder de ter seu valor default descoberto dinamicamente, levando em consideração o estado da computação, e não apenas informações estáticas.

O Quadro 11 apresenta a definição da propriedade *p-ext:defaultParameterValue* em OWL.

```
<owl:DatatypeProperty rdf:ID="defaultParameterValue">
  <rdfs:comment>
    The default value for a process parameter.
  </rdfs:comment>
  <rdfs:domain rdf:resource="&process;Parameter"/>
  <rdfs:range rdf:resource="&rdf;XMLLiteral"/>
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:DatatypeProperty>
```

Quadro 11 – Propriedade *defaultParameterValue* pertencente ao *namespace p-ext*.

É necessário lembrar que, obviamente, deve haver compatibilidade entre o tipo do parâmetro definido através da propriedade *process:parameterType* e este valor default.

Vale ressaltar, ainda, que valores default devem ser únicos para um determinado parâmetro e, mais do que isso, se o valor de um parâmetro pa_1 de um processo p_1 é recebido por um parâmetro pa_2 de um processo p_2, através da definição do fluxo de dados, então se pa_1 e pa_2 possuem valores default, estes valores devem ser iguais.

6.5.4 Classe Relation_for_value

Esta seção define as classes *Relation_for_value*, *ResourceMap* e *ParameterMap* e as propriedades a que elas se aplicam, pertencentes à ontologia *pr*.

A questão mais importante da ontologia *pr*, e que torna o mecanismo de tratamento de exceção bastante interessante, diz respeito à valoração semântica dos relacionamentos entre as instâncias das classes da ontologia.

A valoração semântica dos relacionamentos seria melhor modelada como um atributo de um relacionamento binário. Porém, isto não é diretamente representável em OWL, que apenas permite a definição de relacionamentos binários, sem atributos.

A classe *pr:Relation_for_value* (juntamente com as propriedades à ela relacionadas) possibilita então a atribuição de pesos ou valores aos relacionamentos (binários) [88]. A Figura 6.2 é uma representação gráfica informal da classe *pr:Relation_for_value* e das propriedades relacionadas.

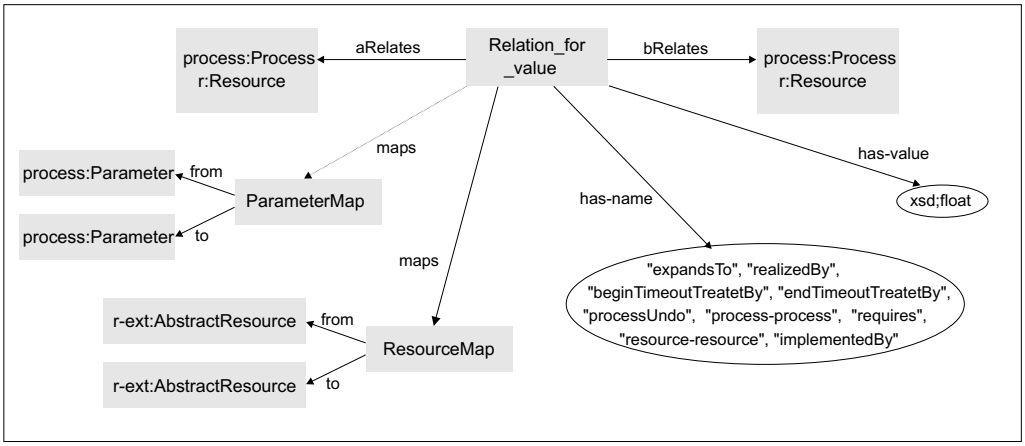

Figura 6.2: Representação gráfica informal da classe *Relation_for_value* e suas propriedades pertencentes à ontologia *pr*.

Vale aqui ressaltar que a modelagem da classe *pr:Relation_for_value* e das propriedades e classes à ela relacionadas poderia seguir uma outra linha de raciocínio, evitando o uso da propriedade *pr:has-name*. Neste caso,

ao invés de criarmos as propriedades *pr:aRelates* e *pr:bRelates* genéricas, atendendo a processos e recursos diversos, criaríamos diversas propriedades, cada uma representando uma situação específica. Por exemplo, uma propriedade que representasse o relacionamento *process:expandsTo* entre um processo abstrato e um processo composto.

A abordagem escolhida para essa modelagem na tese possui a desvantagem de tornar necessária a definição de várias regras restringindo os contra-domínios de cada uma das propriedades, de acordo com a situação que se deseja modelar. No entanto, o uso da outra abordagem citada no parágrafo anterior apresenta o inconveniente de tornar necessária a definição de um grande número de propriedades. Deixamos para trabalhos futuro uma discussão mais aprofundada sobre esta modelagem e sobre os impactos à ela relacionados.

A classe *pr:Relation_for_value* definida na ontologia *pr* relaciona instâncias de processos a outras instâncias de processos ou a instâncias de recursos (além de, em alguns casos, a instâncias de parâmetros), e também relaciona instâncias de recursos a outras instâncias de recursos.

A maioria dos relacionamentos que essa classe representa, no entanto, existe diretamente na ontologia, mas sem atribuição de peso. Por exemplo, quando uma instância da propriedade *process:expandsTo* existe na ontologia, representando a ligação entre um processo simples (ou abstrato) e um processo composto, uma instância da classe *pr:Relation_for_value* é criada para representar que, quando estes dois processos se relacionam por uma instância desta propriedade, este relacionamento apresenta um determinado peso (valor semântico).

Por outro lado, existem casos em que uma instância da classe *pr:Relation_for_value* e de suas propriedades representa relacionamentos (instâncias de propriedades) não presentes na ontologia *pr*, como o relacionamento de proximidade semântica entre dois processos ou entre dois recursos.

Para indicar o tipo de relacionamento que está sendo representado por meio da instância da classe *pr:Relation_for_value*, a esta classe é vinculada a propriedade *pr:has-name*.

O relacionamento entre dois processos pode ocorrer de seis formas distintas:

1. através da propriedade *process:expandsTo* que relaciona um *process:SimpleProcess* e um *process:CompositeProcess*. Neste caso, uma instância da classe *pr:Relation_for_value* é criada para determinar o peso deste relacionamento, que indica o quão adequado é o processo

composto correspondente na implementação daquele processo simples. A propriedade *pr:has-name* desta instância assume como valor a cadeia de caracteres "*expandsTo*";

2. através da propriedade *process:realizedBy* que relaciona um *process:SimpleProcess* e um *process:AtomicProcess*. Neste caso, uma instância da classe *pr:Relation_for_value* é criada para determinar o peso deste relacionamento, que indica o quão adequado é o processo atômico correspondente na implementação daquele processo simples. A propriedade *pr:has-name* desta instância assume como valor a cadeia de caracteres "*realizedBy*";

3. através da propriedade *p-ext:beginTimeoutTreatedBy* que relaciona um par de processos concretos (atômicos ou compostos). Neste caso, uma instância da classe *pr:Relation_for_value* é criada para determinar o peso deste relacionamento, que indica o quão adequado é um processo para o tratamento da exceção temporal de inicialização levantada pelo outro processo, quando ele ainda não tinha começado a execução porque não tinham sido satisfeitas as suas pré-condições. A propriedade *pr:has-name* desta instância assume como valor a cadeia de caracteres "*beginTimeoutTreatedBy*";

4. através da propriedade *p-ext:endTimeoutTreatedBy* que relaciona um par de processos concretos (atômicos ou compostos). Neste caso, uma instância da classe *pr:Relation_for_value* é criada para determinar o peso deste relacionamento, que indica o quão adequado é um processo para o tratamento da exceção temporal de terminação levantada pelo outro processo, quando ele ainda não tinha produzido os resultados de saída. A propriedade *pr:has-name* desta instância assume como valor a cadeia de caracteres "*endTimeoutTreatedBy*";

5. através da propriedade *p-ext:processUndo* que relaciona um processo atômico com um processo concreto (atômico ou composto). Neste caso, uma instância da classe *pr:Relation_for_value* é criada para determinar o peso deste relacionamento, que indica o quão adequado é um processo para desfazer os efeitos de um outro processo, através do tratamento da exceção por cancelamento. A propriedade *pr:has-name* desta instância assume como valor a cadeia de caracteres "*processUndo*";

6. diretamente por meio de uma instância da classe *pr:Relation_for_value*, sem que exista algum outro relacionamento entre eles previamente

definido na ontologia de aplicação. Este é o caso quando se quer definir a proximidade semântica entre pares de processos, quando ambos são processos simples, ou quando cada um deles é atômico ou composto. Neste caso, a propriedade *pr:has-name* desta instância assume como valor a cadeia de caracteres "*process-process*".

Ainda, a classe *pr:Relation_for_value* permite que o relacionamento entre um processo e um recurso abstrato seja valorado. Neste caso, a propriedade *pr:has-name* da instância da classe *pr:Relation_for_value* correspondente assume como valor a cadeia de caracteres "*requires*", e o peso do relacionamento que esta instância representa indica a quantidade deste recurso (sempre abstrato) necessária para a execução do processo correspondente. Se um recurso concreto é necessário à execução de um processo, este relacionamento está especificado apenas na descrição do processo através de sua propriedade *pr:requires* e não existe quantidade relacionada, já que cada recurso concreto é único. Vale ressaltar que processos abstratos apenas podem estar vinculados a recursos abstratos.

A classe *pr:Relation_for_value* permite também representar e atribuir peso ao relacionamento entre dois recursos, que pode ocorrer de duas formas distintas:

1. através da propriedade *r-ext:implementedBy* que relaciona um *r-ext:AbstractResource* e um *r-ext:ConcreteResource*. Neste caso, uma instância da classe *pr:Relation_for_value* é criada para determinar o peso deste relacionamento, que indica o quão adequado é o recurso concreto correspondente na implementação daquele recurso abstrato. A propriedade *pr:has-name* desta instância assume como valor a cadeia de caracteres "*implementedBy*";

2. diretamente por meio de uma instância da classe *pr:Relation_for_value*, sem que haja algum outro relacionamento entre eles previamente definido na ontologia de aplicação. Este é o caso quando se quer definir a similaridade entre pares de recursos, quando ambos os recursos são abstratos ou quando ambos são concretos. Neste caso, a propriedade *pr:has-name* desta instância assume como valor a cadeia de caracteres "*resource-resource*".

Além da propriedade *pr:has-name*, é aplicada à classe *pr:Relation_for_value* a propriedade *pr:has-value*, que indica o peso do relacionamento que está sendo representado por meio dela, e duas outras propriedades: *pr:aRelates* e *pr:bRelates*. Estas duas propriedades possuem

como contra-domínio as classes *process:Process* e *r:Resource*, onde *process* é a abreviação do *namespace* da ontologia de processos de OWL-S e *r* a abreviação do *namespace* da ontologia de recursos de OWL-S, conforme anteriormente apresentado. Assim, elas podem assumir como valor instâncias de processos simples, atômicos, compostos e recursos.

Para efeito de generalização, o valor do relacionamento (peso) representado pela propriedade *pr:has-value* será chamado a partir de agora de valor de *proximidade semântica*, exceto no caso da instância de *pr:Relation_for_value* estar representando um relacionamento entre uma instância de processo e uma instância de recurso, quando este valor significa a quantidade deste recurso necessária à execução deste processo. Assim, é possível determinar o quão semanticamente são próximos, por exemplo, dois processos através do peso de seu relacionamento.

Por definição, cada instância da classe *pr:Relation_for_value* representa um relacionamento direcional. Por exemplo, se a instância da classe *pr:Relation_for_value* possui como valor da propriedade *pr:has-name* a cadeia de caracteres "*expandsTo*", significa que em *pr:aRelates* está determinado o processo abstrato e em *pr:bRelates* o processo concreto que a ele se relaciona. A Tabela 6.1 apresenta um resumo do valor que deve ser assumido por estas propriedades, de acordo com o tipo de relacionamento que a instância de *Relation_for_value* está representando.

6.5.4.1 Mapeamento de Parâmetros

Para as corretas concretização e substituição de processos, é necessário que a classe *pr:Relation_for_value* mantenha informação de mapeamento de parâmetros entre os processos. Para isso, a esta classe aplica-se a propriedade *pr:maps*, cujo contra-domínio é a classe *ParameterMap*, também definida na ontologia *pr*. Esta classe tem por objetivo representar todos os mapeamentos de parâmetros do processo original ou abstrato para parâmetros do processo substituto ou concreto. Vale ressaltar que a execução de uma instância de processo de tratamento de exceção também representa uma forma de substituição, na qual o processo que sofreu a exceção é substituído pelo processo que deve tratá-la.

A essa classe *pr:ParameterMap* aplicam-se duas propriedades, *pr:from* e *pr:to*, ambas tendo como contra-domínio a classe *process:Parameter* da ontologia de processos de OWL-S. A propriedade *pr:from* indica o parâmetro do processo original ou abstrato e a propriedade *pr:to* o

has-name	**aRelates**	**bRelates**
expandsTo	processo abstrato	processo concreto composto
realizedBy	processo abstrato	processo concreto atômico
beginTimeoutTreatedBy	processo em execução	processo a ser executado
endTimeoutTreatedBy	processo em execução	processo a ser executado
processUndo	processo atômico em execução	processo a ser executado
process-process	processo em execução	processo a ser executado
requires	processo	recurso abstrato
implementedBy	recurso abstrato	recurso concreto
resource-resource	recurso original	recurso a ser utilizado

Tabela 6.1: Valor que deve ser assumido pelas propriedades *pr:aRelates* e *pr:bRelates* em *pr:Relation_for_value*.

parâmetro correspondente no processo substituto ou concreto. Como será visto adiante, quando forem apresentadas as regras, este mapeamento só é válido para instâncias da classe *pr:Relation_for_value* cujo valor da instância da propriedade *pr:has-name* é igual à cadeia de caracteres "*expandsTo*", "*realizedBy*" e "*process-process*", ou seja, o mapeamento de parâmetros ocorre nas concretizações e nas substituições de processos.

6.5.4.2 Mapeamento de Recursos

Para a correta concretização de processos, é necessário ainda que a classe *pr:Relation_for_value* e suas propriedades relacionadas mantenham informação de mapeamento de recursos entre os processos. Para isso, à esta classe aplica-se a propriedade *pr:maps*, cujo contra-domínio é a classe *ResourceMap*, também definida na ontologia *pr*.

A essa classe *pr:ResourceMap* aplicam-se as duas propriedades, *pr:from* e *pr:to*, aplicadas também à classe *pr:ParameterMap*. No entanto, neste caso elas contêm como valores recursos abstratos, já que este mapeamento de recursos é sempre entre pares de recursos abstratos. Como será visto nas regras *relation-for-value-expandsToResourceRule* e

relation-for-value-realizedByResourceRule, este mapeamento só é válido para instâncias da classe *pr:Relation_for_value* cujo valor da instância da propriedade *pr:has-name* é igual à cadeia de caracteres "*expandsTo*" e "*realizedBy*", que identificam a concretização de processos. Dessa forma, recursos (abstratos) de processos abstratos precisam ser mapeados em recursos abstratos de processos concretos.

A definição das instâncias da classe *Relation_for_value* e de suas propriedades, no entanto, não é por si só suficiente para evitar que instâncias que representem dados incorretos sejam criadas e, além disso, para garantir que todas as instâncias necessárias desta classe sejam geradas. Por isso, um conjunto de regras independentes de domínio foi definido utilizando-se a linguagem de regras de Jena (veja sintaxe no Capítulo 4).

6.5.4.3 Regras relativas à Classe Relation_for_value

A regra chamada *relation-for-value-expandsToRule*, apresentada no Quadro 12, garante que, se existe na ontologia de aplicação cuja abreviação do *namespace* é *app* um relacionamento entre dois processos (e sabemos que são processos porque a definição desta propriedade nos garante isso) do tipo *process:expandsTo*, sendo um processo simples $p1$ e o outro um processo composto $p2$, então existe em *app* uma instância da classe *pr:Relation_for_value* cujo valor de *pr:aRelates* é $p1$ e de *pr:bRelates* é $p2$, e com *pr:has-name* igual à cadeia de caracteres "*expandsTo*".

```
[relation-for-value-expandsToRule:
 (app:?p1  process:expandsTo  app:?p2) ->
 (app:?w   rdf:type           pr:Relation_for_value)
 (app:?w   pr:has-name        'expandsTo')
 (app:?w   pr:aRelates        app:?p1)
 (app:?w   pr:bRelates        app:?p2)]
```

Quadro 12 − Regra *relation-for-value-expandsToRule*.

A regra *relation-for-value-expandsTo-converseRule*, apresentada no Quadro 13, garante que, se existe uma instância da classe *pr:Relation_for_value* com *pr:has-name* igual à cadeia de caracteres "*expandsTo*", *pr:aRelates* igual a $p1$ e *pr:bRelates* igual a $p2$, então estes dois processos $p1$ e $p2$ originalmente estão associados na ontologia de aplicação através da propriedade *process:expandsTo*. Desta forma, fica garantido que uma instância de *pr:Relation_for_value* com *pr:has-name* igual à cadeia de

caracteres "*expandsTo*" existe se, e somente se, existe um relacionamento entre os dois processos em *app* através da propriedade *process:expandsTo*.

```
[relation-for-value-expandsTo-converseRule:
 (app:?w   rdf:type           pr:Relation_for_value)
 (app:?w   pr:has-name        'expandsTo')
 (app:?w   pr:aRelates        app:?p1)
 (app:?w   pr:bRelates        app:?p2) ->
 (app:?p1  process:expandsTo  app:?p2)]
```

Quadro 13 – Regra *relation-for-value-expandsTo-converseRule*.

A regra *relation-for-value-expandsToParameterRule*, apresentada no Quadro 14, garante que se existe uma instância da classe *pr:Relation_for_value* com *pr:has-name* igual à cadeia de caracteres "*expandsTo*" e o processo do relacionamento referenciado em *pr:aRelates* possui um parâmetro, então este parâmetro deve ser mapeado através de uma instância da propriedade *pr:maps* de *pr:Relation_for_value* para outro parâmetro do processo referenciado em *pr:bRelates*.

```
[relation-for-value-expandsToParameterRule:
 (app:?p1   process:hasParameter  app:?pa1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'expandsTo')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:m     rdf:type      pr:ParameterMap)
 (app:?m    pr:from       pr:?pa1)
 (app:?m    pr:to         pr:?pa2)
 (app:?p2   process:hasParameter  app:?pa2)]
```

Quadro 14 – Regra *relation-for-value-expandsToParameterRule*.

A regra *relation-for-value-expandsToResourceRule*, apresentada no Quadro 15, garante que se existe uma instância da classe *pr:Relation_for_value* com *pr:has-name* igual à cadeia de caracteres "*expandsTo*" e o processo do relacionamento referenciado em *pr:aRelates* possui um recurso abstrato (assumimos que processos abstratos apenas podem ser associados a recursos abstratos), então este recurso deve ser mapeado através de uma instância da propriedade *pr:maps* de *pr:Relation_for_value* para outro recurso do processo referenciado em *pr:bRelates*.

```
[relation-for-value-expandsToResourceRule:
 (app:?p1   pr:requires   app:?r1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'expandsTo')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:m     rdf:type      pr:ResourceMap)
 (app:?m    pr:from       pr:?r1)
 (app:?m    pr:to         pr:?r2)
 (app:?p2   pr:requires   app:?r2)]
```

Quadro 15 – Regra *relation-for-value-expandsToResourceRule*.

De forma análoga, para a propriedade *realizedBy* existem quatro regras: *relation-for-value-realizedByRule*, *relation-for-value-realizedBy-converseRule*, *relation-for-value-realizedByParameterRule* e *relation-for-value-realizeByResourceRule*.

Em conjunto, as duas primeiras regras, apresentadas no Quadro 16, garantem que existe uma instância da propriedade *process:realizedBy* relacionando dois processos $p1$ e $p2$ na ontologia de aplicação se, e somente se, existe uma instância da classe *pr:Relation_for_value* nesta ontologia (ou na biblioteca *lib* que ela importa) com *pr:has-name* igual à cadeia de caracteres "*realizedBy*" e com referência para $p1$ através da propriedade *pr:aRelates* e para $p2$ pela propriedade *pr:bRelates*, sendo $p1$ um processo abstrato e $p2$ um processo atômico.

```
[relation-for-value-realizedByRule:
 (app:?p1  process:realizedBy  app:?p2) ->
 (app:?w   rdf:type            pr:Relation_for_value)
 (app:?w   pr:has-name         'realizedBy')
 (app:?w   pr:aRelates         app:?p1)
 (app:?w   pr:bRelates         app:?p2)]

[relation-for-value-realizedBy-converseRule:
 (app:?w   rdf:type            pr:Relation_for_value)
 (app:?w   pr:has-name         'realizedBy')
 (app:?w   pr:aRelates         app:?p1)
 (app:?w   pr:bRelates         app:?p2) ->
 (app:?p1  process:realizedBy  app:?p2)]
```

Quadro 16 – Regras *relation-for-value-realizedByRule* e *relation-for-value-realizedBy-converseRule*.

A regra *relation-for-value-realizedByParameterRule*, apresentada no Quadro 17, garante que se existe uma instância da classe *pr:Relation_for_value* com

pr:has-name igual à cadeia de caracteres "*realizedBy*" e o processo do relacionamento referenciado em *pr:aRelates* possui um parâmetro p_1, então p_1 deve ser mapeado através de uma instância da propriedade *pr:maps* de *pr:Relation_for_value* para outro parâmetro do processo em *pr:bRelates*, digamos p_2.

```
[relation-for-value-realizedByParameterRule:
 (app:?p1   process:hasParameter  app:?pa1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'realizedBy')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:m     rdf:type      pr:ParameterMap)
 (app:?m    pr:from       pr:?pa1)
 (app:?m    pr:to         pr:?pa2)
 (app:?p2   process:hasParameter  app:?pa2)]
```

Quadro 17 − Regra *relation-for-value-realizedByParameterRule*.

A regra *relation-for-value-realizedByResourceRule*, Quadro 18, garante que se existe uma instância da classe *pr:Relation_for_value* com *pr:has-name* igual à cadeia de caracteres "*realizedBy*" e o processo do relacionamento referenciado em *pr:aRelates* possui um recurso abstrato r_1, então este recurso r_1 deve ser mapeado através de uma instância da propriedade *pr:maps* de *Relation_for_value* para outro recurso r_2 do processo referenciado em *pr:bRelates*.

```
[relation-for-value-realizedByResourceRule:
 (app:?p1   pr:requires   app:?r1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'realizedBy')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:m     rdf:type      pr:ResourceMap)
 (app:?m    pr:from       pr:?r1)
 (app:?m    pr:to         pr:?r2)
 (app:?p2   pr:requires   app:?r2)]
```

Quadro 18 − Regra *relation-for-value-realizedByResourceRule*.

Novamente de forma análoga, os pares de regras
relation-for-value-beginTimeoutTreatedByRule e
relation-for-value-beginTimeoutTreatedBy-converseRule, e
relation-for-value-endTimeoutTreatedByRule e

relation-for-value-endTimeoutTreatedBy-converseRule, apresentados no Quadro 19, são relativos ao relacionamento de tratamento de exceção temporal de inicialização ou de terminação entre dois processos. Assim, essas regras garantem que existe uma instância de relacionamento de tratamento de exceção temporal entre dois processos na ontologia *app* se, e somente se, existe uma instância da classe *pr:Relation_for_value* com *pr:has-name* igual à cadeia de caracteres "*beginTimeoutTreatedBy*", no caso de exceção temporal de inicialização, ou à "*endTimeoutTreatedBy*", no caso de exceção temporal de terminação, e com referência através da propriedade *pr:aRelates* para o processo $p1$ que sofreu exceção durante a execução, e através de *pr:bRelates* para o processo $p2$ que trata da exceção levantada.

```
[relation-for-value-beginTimeoutTreatedByRule:
 (app:?p1  p-ext:beginTimeoutTreatedBy   app:?p2) ->
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'beginTimeoutTreatedBy')
 (app:?w   pr:aRelates   app:?p1)
 (app:?w   pr:bRelates   app:?p2)]

[relation-for-value-beginTimeoutTreatedBy-converseRule:
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'beginTimeoutTreatedBy')
 (app:?w   pr:aRelates   app:?p1)
 (app:?w   pr:bRelates   app:?p2) ->
 (app:?p1  p-ext:beginTimeoutTreatedBy  app:?p2)]

[relation-for-value-endTimeoutTreatedByRule:
 (app:?p1  p-ext:endTimeoutTreatedBy   app:?p2) ->
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'endTimeoutTreatedBy')
 (app:?w   pr:aRelates   app:?p1)
 (app:?w   pr:bRelates   app:?p2)]

[relation-for-value-endTimeoutTreatedBy-converseRule:
 (app:?w   rdf:type     pr:Relation_for_value)
 (app:?w   pr:has-name  'endTimeoutTreatedBy')
 (app:?w   pr:aRelates  app:?p1)
 (app:?w   pr:bRelates  app:?p2) ->
 (app:?p1  p-ext:endTimeoutTreatedBy  app:?p2)]
```

Quadro 19 − Regras
relation-for-value-beginTimeoutTreatedByRule,
relation-for-value-beginTimeoutTreatedBy-converseRule,
relation-for-value-endTimeoutTreatedByRule e
relation-for-value-endTimeoutTreatedBy-converseRule.

Ainda, as regras *relation-for-value-beginTimeoutTreatedByParameterRule* e *relation-for-value-endTimeoutTreatedByParameterRule*, apresentadas no Quadro 20, garantem que se existe uma instância da classe *pr:Relation_for_value* com *pr:has-name* igual à cadeia de caracteres "*beginTimeoutTreatedBy*" e à "*endTimeoutTreatedBy*", respectivamente, e o processo referenciado em *pr:aRelates* possui um parâmetro, então este parâmetro deve ser mapeado através de uma instância da propriedade *pr:maps* de *pr:Relation_for_value* para outro parâmetro do processo em *pr:bRelates*.

```
[relation-for-value-beginTimeoutTreatedByParameterRule:
 (app:?p1   process:hasParameter  app:?pa1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'beginTimeoutTreatedBy')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:m     rdf:type      pr:ParameterMap)
 (app:?m    pr:from       pr:?pa1)
 (app:?m    pr:to         pr:?pa2)
 (app:?p2   process:hasParameter  app:?pa2)]

[relation-for-value-endTimeoutTreatedByParameterRule:
 (app:?p1   process:hasParameter  app:?pa1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'endTimeoutTreatedBy')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:m     rdf:type      pr:ParameterMap)
 (app:?m    pr:from       pr:?pa1)
 (app:?m    pr:to         pr:?pa2)
 (app:?p2   process:hasParameter  app:?pa2)]
```

Quadro 20 – Regras *relation-for-value-beginTimeoutTreatedByParameterRule* e *relation-for-value-endTimeoutTreatedByParameterRule*.

O par de regras *relation-for-value-processUndoRule* e *relation-for-value-processUndo-converseRule*, apresentado no Quadro 21, garante que existe uma instância da classe *pr:Relation_for_value* na ontologia de aplicação *app* com *pr:has-name* igual à cadeia de caracteres "*processUndo*", *pr:aRelates* igual a $p1$ e *pr:bRelates* igual a $p2$ se, e somente se, em *app* $p1$ se relaciona com $p2$ através da propriedade *p-ext:processUndo*, definindo que $p2$ pode desfazer os efeitos de $p1$.

```
[relation-for-value-processUndoRule:
 (app:?p1  p-ext:processUndo   app:?p2) ->
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'processUndo')
 (app:?w   pr:aRelates   app:?p1)
 (app:?w   pr:bRelates   app:?p2)]

[relation-for-value-processUndo-converseRule:
 (app:?w   rdf:type     pr:Relation_for_value)
 (app:?w   pr:has-name  'processUndo')
 (app:?w   pr:aRelates  app:?p1)
 (app:?w   pr:bRelates  app:?p2) ->
 (app:?p1  p-ext:processUndo  app:?p2)]
```

Quadro 21 – Regras *relation-for-value-processUndoRule* e *relation-for-value-processUndo-converseRule*.

A regra *relation-for-value-requiresRule* apresentada no Quadro 22 garante que se existe em *app* uma instância da propriedade *pr:requires* que relaciona um processo p a um recurso abstrato r, então existe uma instância da classe *pr:Relation_for_value* com *pr:has-name* igual à cadeia de caracteres "*requires*", *pr:aRelates* referenciado p e *pr:bRelates* apontando para r.

```
[relation-for-value-requiresRule:
 (app:?p   pr:requires  app:?r)
 (app:?r   rdf:type     r-ext:AbstractResource) ->
 (app:?w   rdf:type     pr:Relation_for_value)
 (app:?w   pr:has-name  'requires')
 (app:?w   pr:aRelates  app:?p)
 (app:?w   pr:bRelates  app:?r)]
```

Quadro 22 – Regra *relation-for-value-requiresRule*.

A regra *relation-for-value-requires-converseRule* (Quadro 23), por sua vez, garante que se existe na ontologia de aplicação uma instância da classe *pr:Relation_for_value* com *pr:has-name* igual à cadeia de caracteres "*requires*", *pr:aRelates* apontando para o processo p e *pr:bRelates* apontando para o recurso r, então é porque existe nesta ontologia uma instância da propriedade *pr:requires* relacionando p com r.

```
[relation-for-value-requires-converseRule:
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'requires')
 (app:?w   pr:aRelates   app:?p)
 (app:?w   pr:bRelates   app:?r) ->
 (app:?p   pr:requires   app:?r)]
```

Quadro 23 – Regra *relation-for-value-requires-converseRule*.

As próximas regras são um pouco diferentes das apresentadas anteriormente, uma vez que elas tratam de um relacionamento que não existe previamente na ontologia de aplicação. As regras anteriores tratavam de garantir a existência de instâncias da classe *pr:Relation_for_value* quando existiam instâncias de propriedades definidas no *namespace app*, às quais se desejava associar um valor. No caso das regras agora apresentadas, o relacionamento que se deseja restringir está explícito apenas através da instância da classe *pr:Relation_for_value* e suas propriedades.

Assim, a regra *relation-for-value-processprocessRule1* garante que, se existe uma instância da classe *pr:Relation_for_value* com *pr:has-name* igual à cadeia de caracteres "*process-process*" e *pr:aRelates* e *pr:bRelates* apontando para dois objetos, então estes dois objetos são processos semanticamente próximos.

A regra *relation-for-value-processProcessRule2*, por sua vez, garante que, se existe uma instância da classe *pr:Relation_for_value* com *pr:has-name* igual a "*process-process*" e *pr:aRelates* e *pr:bRelates* apontando para dois objetos, se um deles é um processo simples, então o outro também é um processo simples. Isso significa que processos simples apenas podem ser semanticamente próximos a processos simples e que processos atômicos e compostos podem ser semanticamente próximos tanto a processos atômicos quanto a processos compostos.

A regra *relation-for-value-processProcessRule3* garante que se existe uma instância da classe *pr:Relation_for_value* com *pr:has-name* igual à cadeia de caracteres "*process-process*", *pr:aRelates* apontando para um processo $p1$ e *pr:bRelates* para um processo $p2$, e $p1$ possui um parâmetro, então este parâmetro deve ser mapeado através de uma instância da propriedade *pr:maps* de *pr:Relation_for_value* para outro parâmetro do processo $p2$ no relacionamento.

As regras *relation-for-value-processProcessRule1*, *relation-for-value-processProcessRule2* e *relation-for-value-processProcessRule3* são apresentadas no Quadro 24.

```
[relation-for-value-processprocessRule1:
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'process-process')
 (app:?w   pr:aRelates   app:?p1)
 (app:?w   pr:bRelates   app:?p2) ->
 (app:?p1  rdf:type      process:Process)
 (app:?p2  rdf:type      process:Process)]
```

```
[relation-for-value-processProcessRule2:
 (app:?w   rdf:type     pr:Relation_for_value)
 (app:?w   pr:has-name  'process-process')
 (app:?w   pr:aRelates  app:?p1)
 (app:?w   pr:bRelates  app:?p2)  ->
 [(app:?p1    rdf:type  process:SimpleProces) ->
  (app:?p2    rdf:type  process:SimpleProcess)]
 [(app:?p2    rdf:type  process:SimpleProces) ->
  (app:?p1    rdf:type  process:SimpleProcess)]]

[relation-for-value-processprocessRule3:
 (app:?p1   process:hasParameter  app:?pa1)
 (app:?w    rdf:type      pr:Relation_for_value)
 (app:?w    pr:has-name   'process-process')
 (app:?w    pr:aRelates   app:?p1)
 (app:?w    pr:bRelates   app:?p2) ->
 (app:?w    pr:maps       pr:?m)
 (app:?m    pr:from       pr:?pa1)
 (app:?m    pr:to         pr:?pa2)
 (app:?p2   process:hasParameter  app:?pa2)]
```

Quadro 24 – Regras *relation-for-value-processprocessRule1*, *relation-for-value-processProcessRule2* e *relation-for-value-processprocessRule3*.

A regra *relation-for-value-resourceResourceRule1*, apresentada no Quadro 25, faz um papel semelhante à regra *relation-for-value-processprocessRule1*, exceto pelo fato de que involve recursos ao invés de processos.

```
[relation-for-value-resourceResourceRule1:
 (app:?w   rdf:type      pr:Relation_for_value)
 (app:?w   pr:has-name   'resource-resource')
 (app:?w   pr:aRelates   app:?r1)
 (app:?w   pr:bRelates   app:?r2) ->
 (app:?r1  rdf:type      r:Resource)
 (app:?r2  rdf:type      r:Resource)]
```

Quadro 25 – Regra *relation-for-value-resourceResourceRule1*.

A regra *relation-for-value-resourceResourceRule2*, Quadro 26, faz papel análogo ao da regra *relation-for-value-processProcessRule2*, determinando que o relacionamento entre dois recursos ocorra sempre entre dois recursos concretos ou entre dois recursos abstratos.

```
[relation-for-value-resourceResourceRule2:
 (app:?w   rdf:type     pr:Relation_for_value)
 (app:?w   pr:has-name  'resource-resource')
```

```
(app:?w   pr:aRelates  app:?r1)
(app:?w   pr:bRelates  app:?r2)  ->
[(app:?r1    rdf:type  r-ext:AbstractResource) ->
 (app:?r2    rdf:type  r-ext:AbstractResource)]
[(app:?r2    rdf:type  r-ext:AbstractResource) ->
 (app:?r1    rdf:type  r-ext:AbstractResource)]]
```

Quadro 26 – Regra *relation-for-value-resourceResourceRule2*.

Por fim, o par de regras *relation-for-value-implementedByRule* e *relation-for-value-implementedBy-converseRule*, apresentado no Quadro 27, garante que existe uma instância da propriedade *r-ext:implementedBy* relacionando dois recursos $r1$ e $r2$ (e sabemos que são recursos porque a definição desta propriedade nos garante isso) no *namespace app* se, e somente se, existe uma instância da classe *pr:Relation_for_value* com *pr:has-name* igual à cadeia de caracteres "*implementedBy*" e com referência, através das propriedades *pr:aRelates* e *pr:bRelates*, para os dois recursos.


```
[relation-for-value-implementedByRule:
 (app:?r1  r-ext:implementedBy  app:?r2) ->
 (app:?w   rdf:type             pr:Relation_for_value)
 (app:?w   pr:has-name          'implementedBy')
 (app:?w   pr:aRelates          app:?r1)
 (app:?w   pr:bRelates          app:?r2)]

[relation-for-value-implementedBy-converseRule:
 (app:?w   rdf:type             pr:Relation_for_value)
 (app:?w   pr:has-name          'implementedBy')
 (app:?w   pr:aRelates          app:?r1)
 (app:?w   pr:bRelates          app:?r2) ->
 (app:?r1  r-ext:implementedBy  app:?r2)]
```

Quadro 27 – Regras *relation-for-value-implementedByRule* e *relation-for-value-implementedBy-converseRule*.

Note que o conjunto de regras apresentado nesta seção é independente do domínio da aplicação, apesar de estar definido sobre o *namespace app*. Isso ocorre porque as instâncias de processos, recursos, propriedades e da própria classe *pr:Relation_for_value* podem estar definidas tanto na biblioteca *lib* quanto na ontologia de aplicação. No entanto, estas regras se aplicam a qualquer ontologia *app*.

Regras adicionais podem se fazer necessárias para representar a complexidade do domínio com o qual se está trabalhando. Por exemplo, regras adicionais são necessárias para determinar o valor da propriedade

pr:has-value quando este valor é dependente de um conjunto de variáveis da execução.

6.6 Resumo

A abordagem para flexibilização da execução de processos empregada nesta tese utiliza informação adicional associada à definição estática dos processos. Mais precisamente, cada máquina de execução tem acesso a várias ontologias (organizadas como indica a Figura 6.1):

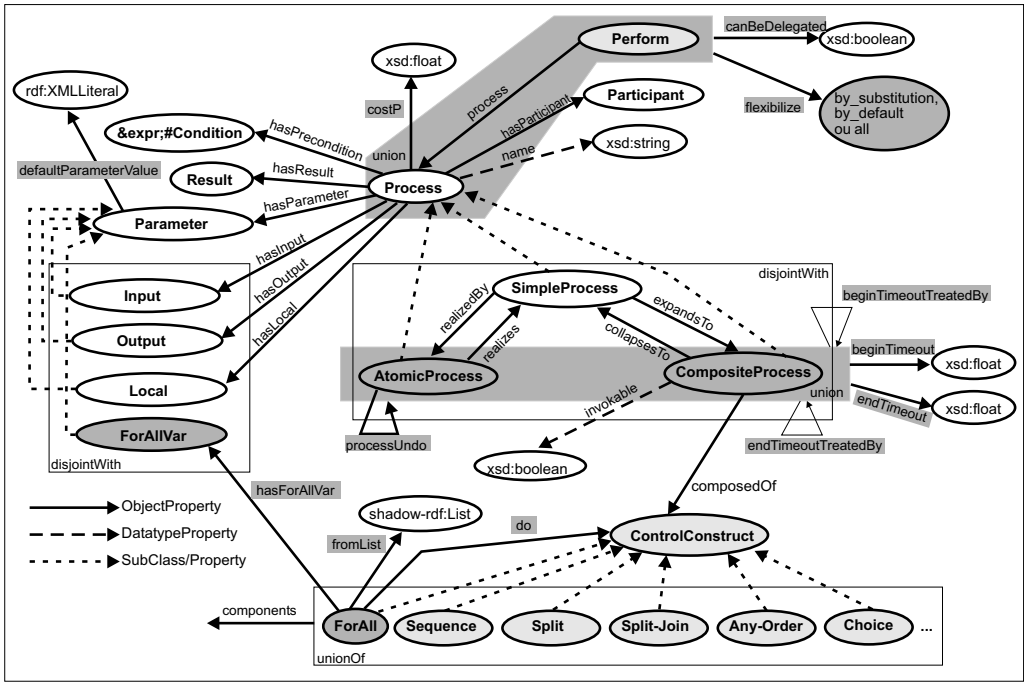

Figura 6.3: Extensões sobre a ontologia de processos de OWL-S.

- a ontologia *pr* é a base do mecanismo de tratamento de exceção e importa as ontologias *p-ext* e *r-ext*. Esta ontologia *pr* contém: (i) classes e propriedades para a valoração semântica de relacionamentos; (ii) classes e propriedades adicionais para a correta concretização e substituição de processos, mantendo informação sobre o mapeamento de recursos e parâmetros entre processos; e (iii) regras que complementam a semântica da classe *pr:Relation_for_value*;
- a ontologia *p-ext* importa a ontologia de processos de OWL-S e a estende com novas classes, como *ForAll*, e novas propriedades, *beginTimeout*, *endTimeout*, *beginTimeoutTreatedBy*, *endTimeoutTreatedBy*, *canBeDelegated*, *flexibilize*, *defaultParameterValue*,

processUndo e *costP*. Estas propriedades são de fundamental importância para a flexibilização da execução de uma instância de processo. As extensões propostas sobre a ontologia de processos de OWL-S estão destacadas em cinza escuro na Figura 6.3, originária da Figura 4.3 apresentada no capítulo anterior;

– a ontologia *r-ext* importa a ontologia de recursos de OWL-S e a estende com novas classes, *AbstractResource* e *ConcreteResource*, e novas propriedades, *implementedBy*, *implementationOf*, *allocated* e *costR*;

– a ontologia *lib* importa a ontologia *pr*. Esta ontologia contém as instâncias de processos, recursos e suas propriedades que têm alto grau de reuso para um domínio de aplicação;

– uma ontologia de aplicação, denominada genericamente por *app*, que importa a biblioteca *lib*. Esta ontologia reaproveita as instâncias de processos, recursos e suas propriedades já definidas em *lib*, úteis à aplicação sendo definida, acrescentando novas instâncias e possivelmente novas regras dependentes do domínio da aplicação.