UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

Relatórios Técnicos
do Departamento de Informática Aplicada
da UNIRIO
n° 0010/2011

WS-Policy: conceitos e propostas de uso

Douglas Machado Silva
Fabiana Jack Nogueira Santos
Leonardo Guerreiro Azevedo

Departamento de Informática Aplicada

UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
Av. Pasteur, 458, Urca - CEP 22290-240
RIO DE JANEIRO – BRASIL

WS-Policy: conceitos e propostas de uso

Douglas Machado Silva, Fabiana Jack Nogueira Santos, Leonardo Guerreiro Azevedo

Núcleo de Pesquisa e Prática em Tecnologia (NP2Tec) – Universidade Federal do Estado do Rio de Janeiro (UNIRIO)
Programa de Pós-Graduação em Informática (PPGI) – Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

{douglas.machado, fabiana.nogueira, azevedo}@uniriotec.br

Abstract. In environments that employ Service-Oriented Architecture approach, service discovery according to consumer requirements is a challenge. The UDDI pattern is presented as the main pattern for storing service descriptions, allowing quering services, update service information to consumers, and gathering information for service invocation. However, this pattern does not comply with all consumer requirements. Therefore, there are proposals of extensions of this pattern employing ontologies and WS-Policy for service discovery, as well as service monitoring during runtime. This work presents the main concepts of WS-Policy as well as research proposals of using WS-Policy and improving it for service discovery, service monitoring and definition of QoS attributes.

Keywords: SOA, web services, WS-Policy, QoS, UDDI, BPEL.

Resumo. Em Arquitetura Orientadas a Serviço (SOA - Service-Oriented Architecture) a descoberta do serviço que melhor antende às necessidades do consumidor é um desafio. O padrão UDDI é apresentado como principal padrão para armazenamento das descrições dos serviços, permitindo consultar serviços, divulgar atualizações sobre os mesmos e recuperar informações para invocação. No entanto, este padrão não atende a todos os requisitos dos consumidores. Dessa forma, autores propõem extensões deste padrão empregando ontologias e WS-Policy para descoberta de serviços, bem como para monitoramento de execução de serviços. Este trabalho apresenta os principais conceitos de WS-Policy bem como trabalhos da literatura que propõe o uso deste padrão e evolução do mesmo para descoberta de serviços, monitoramento de serviços e definição de atributos de qualidade de serviços (QoS).

Palavras-chave: SOA, web services, WS-Policy, QoS, UDDI, BPEL.

Sumário

1 Introdução

Ao se considerar o uso de aplicações distribuídas, um ponto importante a ser considerado é a dificuldade inerente à interação entre essas aplicações. Um desenvolvedor pode conhecer sua necessidade em termos de saber "o que" deseja alcançar, não sabendo necessariamente "como" alcançar [Chappell, 2007]. Da mesma forma, aquele que provê o serviço pode não saber exatamente quais são as necessidades daquele que consome, ou seja, pode não saber "como" fornecer o serviço.

A situação se torna ainda mais difícil se o usuário leigo for levado em consideração. Um usuário que deseja utilizar uma aplicação de *streaming* de vídeo, por exemplo, pode optar por diferentes níveis de qualidade para este vídeo, como "baixa", "média" e "alta". Entretanto, este usuário dificilmente compreenderia termos técnicos (como largura de banda e latência) para definir os parâmetros para a execução deste serviço [Fugini *et al*., 2006]. Outro ponto importante a se considerar são as especificidades do domínio em si. Consumidores de diferentes domínios podem ter restrições diferentes para o consumo de um mesmo serviço [Heinzl e Schmeling, 2010].

A partir dos problemas apresentados, alguns questionamentos podem surgir:

- Como escolher o melhor serviço para um determinado contexto?
- Como serviços podem se adaptar de acordo com um contexto?
- Como serviços podem transformar dados de acordo com um contexto?
- Como encontrar os serviços mais adequados para compor uma composição (*composite*)?

A simples busca de um serviço em um repositório, como um UDDI (Universal Description, Discovery & Integration Specification) [UDDI, 2004], não garante que o serviço encontrado atende às necessidades daquele que vai consumi-lo. Restrições em relação ao fornecimento do serviço, muitas vezes, podem não estar sendo levadas em consideração. Um consumidor de um serviço pode necessitar escolher, dentre vários fornecedores de serviço, aquele que atenda suas restrições de prazo, por exemplo. Além disso, os fornecedores podem ter diferentes prazos dependendo da demanda do serviço. Caso o consumidor e o fornecedor do serviço não explicitem o tratamento destas restrições, não há como o consumidor escolher o fornecedor adequado [Sriharee *et al*., 2004].

Nota-se, desta forma, que os *web services* não podem ser vistos apenas sob o aspecto funcional. É necessário, também, considerar o estabelecimento de propriedades não-funcionais [Heinzl e Schmeling, 2010]. Sendo assim, o fornecedor de um serviço precisa especificar também suas capacidades em termos de provimento do serviço. Ou seja, não só o consumidor necessita estabelecer restrições para o uso do serviço. O fornecedor também pode necessitar verificar e validar os serviços que oferece. Da mesma forma, consumidores e fornecedores de serviço podem desejar terceirizar o monitoramento dos serviços, em termos de capacidade e do histórico de provimento [Baresi *et al*., 2006].

Embora todos esses aspectos sejam relevantes, um, em especial, despertou a atenção das pesquisas em um primeiro momento: a falta de segurança dos *web services*. Isso porque, sob o ponto de vista técnico, *web services* baseiam-se em padrões XML, como SOAP, UDDI e WSDL [Booth *et al*., 2004], o que expõe os serviços a ameaças de

segurança [Hung, 2004]. A falta de segurança associada à falta de monitoramento e ao ambiente compartilhado, inclusive, são apontadas como as principais barreiras para a implantação de *web services* pelas organizações [Hung, 2004].

Para endereçar as limitações dos *web services*, o padrão WS-POLICY [WS-Policy, 2006] foi proposto. O WS-Policy é um framework utilizado para que clientes e serviços possam especificar seus requisitos, capacidades ou preferências, chamados de políticas, de forma interoperável. Ele define um conjunto de construtores genéricos para definição e agrupamento das asserções de políticas, que representam uma preferência, capacidade ou requisito únicos. Por exemplo, uma asserção pode indicar que o serviço requer um determinado mecanismo de autenticação [Alonso *et al*., 2004].

O WS-Policy não define políticas específicas de domínio, ele apenas fornece um mecanismo para agrupá-las. Da mesma forma, ele não fornece mecanismos para anexar a política ao serviço, ele apenas descreve uma entidade abstrata chamada *policy subject* que é o *bound* da política, assim como são definidos os *endpoints* no documento WSDL [Alonso *et al*., 2004].

O objetivo deste trabalho é apresentar definições de WS-Policy, assim como algumas propostas de uso do WS-Policy na literatura.

Este relatório está organizado em X capítulos, sendo o capítulo 1 a presente introdução. No capítulo 2 são apresentados os principais conceitos de WS-Policy, de acordo com Box *et al.* [2004]. O capítulo 3 apresenta propostas da literatura de uso de WS-Policy. Os capítulos 4 e 5 apresentam as conclusões e referências bibliográficas, respectivamente.

2 Principais conceitos de WS-Policy

Esta seção apresenta os principais conceitos sobre WS-Policy descritos por Box *et al.* [2004].

2.1 Definições

O conceito de *policy* é relativamente simples. Segundo Chappell [2007], as *policies* são políticas que modificam a forma com que um componente interage com os demais. Além disso, modificam também o comportamento do componente localmente.

O *Web Services Policy Framework* (WS-Policy) fornece sintaxe e modelos gerais para descrever as políticas de um serviço web (*web service)*. Este *framework* define um conjunto de construtores que podem ser utilizados e estendidos por outras especificações dos *web services*, para descrever uma ampla gama de requisitos e capacidades de um serviço.

WS-Policy disponibiliza uma gramática extensível e flexível para expressar capacidades, requisitos e características gerais das entidades existentes em um sistema baseado em *web services* e XML.

WS-Policy define uma política (*policy*) como uma coleção de alternativas de política, onde cada alternativa é uma coleção de asserções da política (*policy assertions*). As asserções da política especificam requisitos tradicionais e capacidades que, em última instância, se manisfestarão na ligação (*wire*), por exemplo, no esquema de autenticação ou na seleção do protocolo de transporte. Outras asserções da política não possuem nenhuma manifestação na ligação (*wire*), ainda que sejam críticas para a seleção e uso

adequados do serviço, por exemplo, política de privacidade e características de qualidade de serviço (QoS – Quality of Service). Utilizando a gramática de WS-Policy é possível utilizar estes dois tipos de asserções para que sejam fundamentados de forma coerente.

WS-Policy não especifica como as políticas serão descobertas ou anexadas ao *web service*. Outras especificações podem definir mecanismos específicos para associação de políticas com várias entidades e recursos. WS-PolicyAttachment define tais mecanismos, especialmente para associação de políticas com elementos arbitrários em XML, artefatos WSDL e elementos do UDDI. As especificações subsequentes fornecerão perfis (*profiles*) de uso da WS-Policy com outras tecnologias de *web services*.

O objetivo do WS-Policy é fornecer mecanismos necessários para que aplicações desenvolvidas utilizando-se *web services* possam especificar informações de políticas. A especificação define um *infoset* em XML chamado expressão de política (*policy expression*) que contém informações específicas de domínio e um conjunto de construtores. Estes elementos permitem que as escolhas e/ou combinações de asserções de políticas específicas de domínio sejam indicadas.

WS-Policy foi desenvolvido para funcionar com os *frameworks* gerais de *web services*, incluindo as descrições de serviços em WSDL e os registros de serviços no UDDI [Box *et al*., 2004]. WS-Policy é considerado um padrão de metadados, assim como WSDL, WS-MetadataExchange e UDDI. O WSDL descreve as mensagens que o serviço pode receber e enviar, é a linguagem mais básica que descreve o contrato do serviço. O WS-Policy descreve a qualidade das características do serviço e os requisitos associados ao serviço. As políticas descrevem exigências de segurança de um serviço e otimizações suportadas pelo serviço. Já o WS-MetadaExchange é o protocolo que permite a recuperação dos documentos WSDL e WS-Policy de um serviço [Pavlik *et al*. 2007].

2.2 Exemplo

A Figura 1 apresenta um exemplo de política de segurança utilizando asserções definidas no WS-SecurityPolicy. As linhas de 1 até 10 representam uma política de autenticação. As linhas de 2 a 9 representam o operador de política *ExactlyOne*. Operadores de política agrupam asserções de políticas em alternativas de políticas. Uma interpretação válida para as linhas 3 até 8 seria que a invocação de um *web service* contém uma das asserções de *token* de segurança descritas: Kerberosv5TGT ou X509v3. As linhas de 3 a 5 e de 6 a 8 representam 2 asserções específicas de políticas de segurança que indicam que dois tipos de autenticação são suportados.

```
01 <wsp:Policy>
02   <wsp:ExactlyOne>
03     <wsse:SecurityToken>
04       <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
05     </wsse:SecurityToken>
06     <wsse:SecurityToken>
07       <wsse:TokenType>wsse:X509v3</wsse:TokenType>
08     </wsse:SecurityToken>
09   </wsp:ExactlyOne>
10 </wsp:Policy>
```

Figura 1 - Exemplo de política de segurança

2.3 Notações e terminologia

A Figura 2 representa os dados envolvidos na definição das políticas.

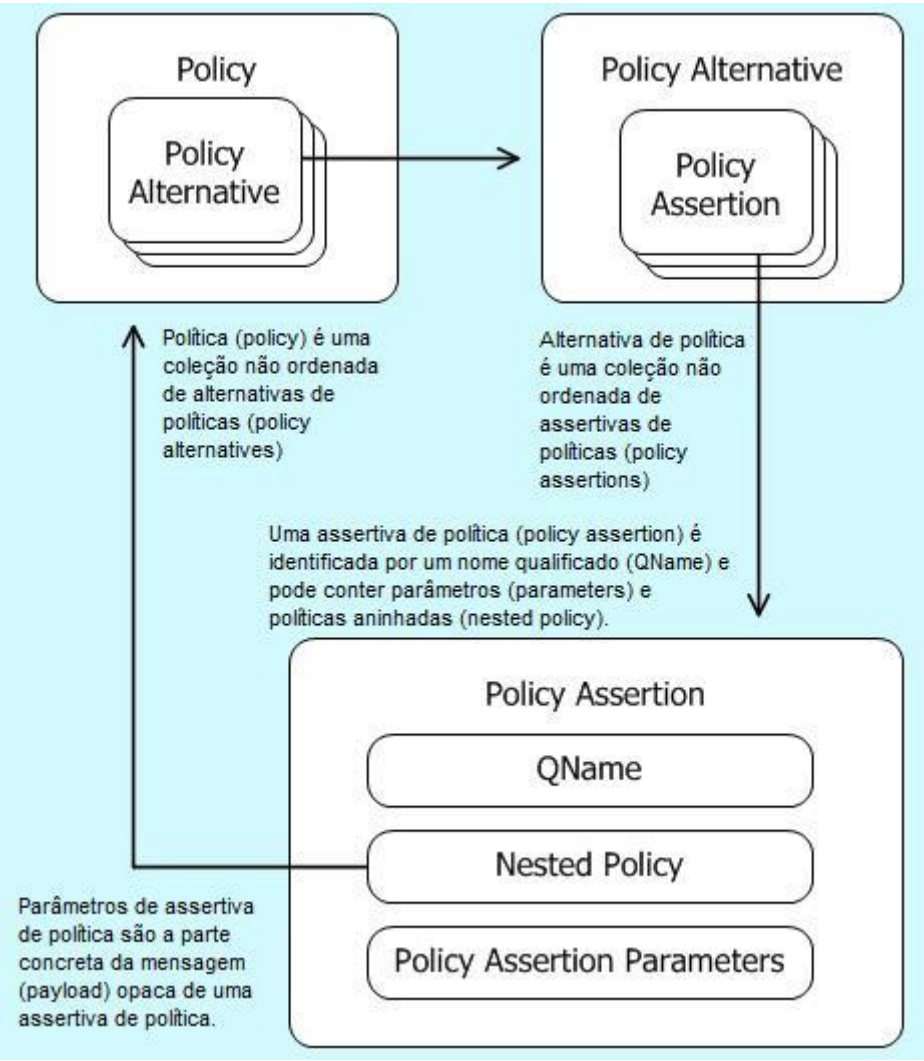

Figura 2 – Modelo de dados do WS-Policy [Web Services Policy 1.5 - Primer].

A especificação do WS-Policy utiliza uma sintaxe semelhante à de XML. Os principais termos apresentados por Box *et al.* [2004] são apresentados a seguir.

- Política (*policy*): é um conjunto de alternativas de políticas;
- Alternativa de política (*policy alternative*): é uma coleção de asserções de política;
- Asserção de política (*policy assertion*): identifica o comportamento que é um requisito individual ou uma capacidade de um assunto de política (*policy subject*);
- Tipo de asserções de política (*policy assertion type*): representa uma classe de asserções de políticas e implica em um esquema para instâncias da asserção e uma semântica específica da asserção;
- Vocabulário de política (*policy vocabulary*): é um conjunto de todos os tipos de asserções utilizados em uma política;
- Expressão de política (*policy expression*): é uma representação de uma política através de um *infoset* XML, de uma forma normal ou de uma forma compacta equivalente;
- Assunto de política (*policy subject*): é uma entidade com a qual uma política pode ser associada;
- Escopo da política (*policy scope*): é uma coleção de assuntos de política para a qual a política se aplica;
- Anexo da política (*policy attachment*): é o mecanismo para associar políticas a um ou mais escopos de política.

2.4 Modelo de Políticas

A asserção de política identifica o comportamento que é um requisito ou capacidade de um assunto de política (*policy subject*). As asserções indicam semântica específica de um domínio e são definidas em especificações separadas, específicas do domínio. Essas informações são fortemente tipadas.

Por exemplo, uma asserção para identificar o suporte para um mecanismo específico de mensagem confiável deve incluir um item de atributo de informação para indicar quanto tempo o *endpoint* vai esperar antes de enviar um reconhecimento.

Uma alternativa de política é uma coleção de asserções de política potencialmente vazia, uma alternativa com zero asserções indica que nenhum comportamento ocorrerá. Já uma alternativa com uma ou mais asserções indica comportamentos explícitos por elas e pelas asserções.

O vocabulário de uma alternativa de política é um conjunto de todos os tipos de asserções de uma alternativa. O vocabulário de uma política é o conjunto de todos os tipos de asserções usados na política.

Uma política é uma coleção potencialmente vazia de alternativas de política. Uma política com nenhuma alternativa não contém nenhuma opção, uma política com uma ou mais alternativas indica escolha dos requisitos ou capacidades no âmbito da política.

Quando aplicadas a *web services*, as políticas são usadas para transmitir condições na interação entre dois *endpoints*. Ao satisfazer uma asserção em uma política, tem-se como resultado o comportamento que satisfaz estas condições. Tipicamente, o provedor de um *web service* expõe a política para transmitir as condições sob as quais ele

provê o serviço. O solicitante do serviço usa esta política para decidir se usa ou não aquele serviço. O solicitante deve escolher uma alternativa, pois ela é uma configuração válida para interação com o serviço, mas deve escolher apenas uma, pois cada uma indica uma configuração.

Uma asserção de política é suportada pelo solicitante se, e somente se, o solicitante satisfaz o requisito (ou acomoda a capacidade) correspondente à asserção. Uma alternativa de política é suportada pelo solicitante se, e somente se, o solicitante suporta todas as asserções em uma alternativa. Uma política é suportada pelo solicitante se, e somente se, o solicitante suporta pelo menos uma das alternativas na política.

O solicitante deve ser capaz de suportar a política mesmo se ele não entende o tipo de cada afirmação no vocabulário da política. O solicitante só precisa entender o tipo de cada afirmação no vocabulário da alternativa da política. Essa característica é crucial para permitir o desenvolvimento de versões incremental das asserções de política, pois permite que o fornecedor da política inclua novas asserções em novas alternativas enquanto continua permitindo que os solicitantes usem as alternativas antigas.

2.5 Expressão de Políticas

Para transmitir uma política de forma interoperável, a expressão de uma política é uma representação utilizando XML *infoset*. A forma normal de expressão de uma política é o *infoset* mais simples. De forma equivalente, *infosets* alternativos permitem a expressão de uma política de forma alternativa através de construtores. A Figura 3 apresenta a forma normal de representar uma política.

```
<wsp:Policy ... >
  <wsp:ExactlyOne>
    [ <wsp:All> [ <Assertion ...> ... </Assertion> ]* </wsp:All> ]*
   </wsp:ExactlyOne>
</wsp:Policy>
```

Figura 3 - Forma normal de representar uma política

Nesta representação fica claro que a política é composta pela enumeração de suas alternativas e cada uma das alternativas enumera suas asserções. A Figura 4 apresenta um exemplo de esquema para representar a expressão de políticas. As linhas de 3 a 7 e de 8 a 12 expressam 2 alternativas na política. Se a primeira alternativa é selecionada, apenas o tipo de *token* Kerberos é suportado. Se a segunda alternativa é selecionada, apenas o tipo de *token* X509 é suportado.

```
01 <wsp:Policy>
02   <wsp:ExactlyOne>
03     <wsp:All>
04       <wsse:SecurityToken>
05         <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
06       </wsse:SecurityToken>
07     </wsp:All>
08     <wsp:All>
09       <wsse:SecurityToken>
10         <wsse:TokenType>wsse:X509v3</wsse:TokenType>
11       </wsse:SecurityToken>
12     </wsp:All>
13   </wsp:ExactlyOne>
14 </wsp:Policy>
```

Figura 4 - Exemplo de esquema para representar a expressão de políticas

2.6 Identificação de Políticas

A expressão de uma política pode ser atribuída a um *namespace*, indicado utilizando-se o atributo `@TargetNamespace` do elemento `wsp:Policy` ou herdando de um elemento que o contenha. Se nenhum *namespace* for especificado, então o *namespace* será "". A expressão de uma política também pode ser um recurso web, portanto identificável por uma URI. Para permitir que expressões de políticas sejam incorporadas a elementos arbitrários que a contenham, o atributo `@wsu:Id` deve ser usado para indicar o ID do fragmento. A Figura 5 é apresenta um esquema com estes atributos.

```
<wsp:Policy xml:base="xs:anyURI" ?

  wsu:Id="xs:ID" ?
  TargetNamespace="xs:anyURI" ? ... >
  ...
</wsp:Policy>
```

Figura 5 – Exemplo de atributos de uma política.

A Figura 6 apresenta um exemplo para ilustrar a expressão de uma política com URI. Neste exemplo, a URI da política é "http://www.w3.org/2000/09/xmldsig#rsa-sha1".

```
<wsp:Policy xml:base="http://fabrikam123.com/policies" wsu:Id="P1" >
  <wsse:SecurityToken>
       <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
  </wsse:SecurityToken>
  <wsse:Integrity>
  <wsse:Algorithm Type="wsse:AlgSignature"
        URI="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
  </wsse:Integrity>
</wsp:Policy>
```

Figura 6 – Exemplo de expressão de uma plítica com URI.

2.7 Interseção de Políticas

A interseção de políticas é útil quando duas ou mais partes expressam políticas e querem limitar as alternativas da política àquelas que são mutuamente compatíveis. Por exemplo, quando um solicitante e um provedor expressam requisitos na troca de mensagens, a interseção identifica alternativas de política compatíveis (se existir) incluídas nas políticas tanto do solicitante quanto do provedor. A interseção é uma função comutativa e associativa que pega duas políticas e retorna apenas uma.

Devido ao conjunto de comportamentos indicados, a alternativa da política depende da semântica específica do domínio das instâncias das asserções coletadas para determinar se duas alternativas de políticas são compatíveis. Isso geralmente envolve processamento específico de domínio. Em uma abordagem, é definido um algoritmo que aproxima a compatibilidade de uma maneira independente de domínio. Para que duas alternativas de políticas sejam compatíveis elas devem, pelo menos, possuir o mesmo vocabulário.

Usando esta abordagem, a interseção entre duas políticas de entrada P1 e P2 é uma política Pi composta por cada alternativa Ai com as seguintes características:

- Uma dada política alternativa Ax em P1 e alguma alternativa Ay em P2, tal que o vocabulário de Ax é igual ao vocabulário de Ay;
- A coleção de instâncias de asserções em Ai é composta por todas as instâncias de asserções de Ax e todas as instâncias de asserções de Ay.

Implicações das definições:

- O vocabulário de Ai é igual ao vocabulário de Ax que é igual ao vocabulário de Ay;
- Quando os vocabulários de duas políticas de entrada se sobrepõem, mas são diferentes, o vocabulário de interseção destas políticas é um subconjunto do vocabulário das políticas de entrada;
- Se o vocabulário de uma política inclui um tipo de afirmação que não está no vocabulário da outra política, então o comportamento associado a este tipo de afirmação é proibido na interseção destas políticas.

As

Figura 7 e Figura 8 apresentam dois exemplos de políticas.

```
<wsp:Policy> // Policy P1
      <wsp:ExactlyOne>
            <wsp:All> // Alternative A1
                  <wsse:Confidentiality>
                        <wsse:Algorithm Type="wsse:AlgEncryption"
                                    URI="http://www.w3.org/2001/04/
                                    xmlenc#3des-cbc" />
                        <MessageParts
                              Dia-
                              lect="http://schemas.xmlsoap.org/2002
                              /12/wsse#part" >
                           wsp:Body
                        </MessageParts>
                  </wsse:Confidentiality>
            </wsp:All>
            <wsp:All> // Alternative A2
                  <wsse:Confidentiality>
                        <wsse:Algorithm Type="wsse:AlgEncryption"
                                    URI="http://www.w3.org/2001/04/
                              xmlenc#3des-cbc" />
                        <MessageParts
                              Dia-
                              lect="http://schemas.xmlsoap.org/2002
                              /12/wsse#part" >
                           wsp:Body
                        </MessageParts>
                  </wsse:Confidentiality>
                  <wsse:SecurityHeader MustPrepend="true"
                                    MustManifestEncryption="true"
                                    />
            </wsp:All>
      </wsp:ExactlyOne>
</wsp:Policy>
```

Figura 7 – Exemplo de política P1

```
<wsp:Policy> // Policy P2
      <wsp:ExactlyOne>
            <wsp:All> // Alternative A3
                  <wsse:Confidentiality>
                        <wsse:Algorithm Type="wsse:AlgEncryption"
                                        URI="http://www.w3.org/20
                                        01/04/xmlenc#3des-cbc" />
                        <MessageParts
                              Dia-
                              lect="http://schemas.xmlsoap.org/2002
                              /12/wsse#part" >
                           wsp:Body wsp:Header(x:AccountNumber)
                        </MessageParts>
                  </wsse:Confidentiality>
                  <wsse:SecurityHeader MustManifestEncryp-
            tion="true"/>
            </wsp:All>
            <wsp:All> //Alternative A4
                  <wsse:Confidentiality>
                        <wsse:Algorithm Type="wsse:AlgEncryption"
                                        URI="http://www.w3.org/20
                                    01/04/xmlenc#3des-cbc" />
                        <MessageParts
                              Dia-
                              lect="http://schemas.xmlsoap.org/2002
                              /12/wsse#part" >
                           wsp:Body wsp:Header(x:AccountNumber)
                        </MessageParts>
                  </wsse:Confidentiality>
                  <wsse:SecurityHeader MustManifestEncryp-
                  tion="true"/>
                  <wsse:MessageAge Age="3600" />
            </wsp:All>
      </wsp:ExactlyOne>
</wsp:Policy>
```

Figura 8 – Exemplo de Política P2

Por existir apenas uma alternativa (A2) na política P1 com o mesmo vocabulário de outra alternativa (A3) na política P2, a interseção é a política com apenas uma alternativa que contém todas as asserções de A2 e A3, representada abaixo.

```
<wsp:Policy> // Interseção de P1 e P2
      <wsp:ExactlyOne>
            <wsp:All>
                  <wsse:Confidentiality>
                        <wsse:Algorithm Type="wsse:AlgEncryption"
                                    URI="http://www.w3.org/2001/04/
                                    xmlenc#3des-cbc" />
                        <MessageParts
                              Dia-
                              lect="http://schemas.xmlsoap.org/2002
                              /12/wsse#part" >
                           wsp:Body
                        </MessageParts>
                  </wsse:Confidentiality>
                  <wsse:SecurityHeader MustPrepend="true"
                                    MustManifestEncryption="true"
                                    />
                  <wsse:Confidentiality>
                        <wsse:Algorithm Type="wsse:AlgEncryption"
                                    URI="http://www.w3.org/2001/04/
                                    xmlenc#3des-cbc" />
                        <MessageParts
                              Dia-
                              lect="http://schemas.xmlsoap.org/2002
                              /12/wsse#part" >
                           wsp:Body wsp:Header(x:AccountNumber)
                        </MessageParts>
                  </wsse:Confidentiality>
                  <wsse:SecurityHeader MustManifestEncryp-
            tion="true"/>
            </wsp:All>
      </wsp:ExactlyOne>
</wsp:Policy>
```

Figura 9 – Interseção das política P1 e P2

Na política P1, o vocabulário é {`wsse:Confidentiality, wsse:SecurityHeader`}, na política P2, o vocabulário é {`wsse:Confidentiality, wsse:SecurityHeader, wsse:MessageAge`} e o vocabulário da interseção é {`wsse:Confidentiality, wsse:SecurityHeader`}. No exemplo, existem duas instâncias do tipo de asserção `wsse: Confidentiality`. Quando o comportamento associado ao `wsse:Confidetiality` é chamado, o conteúdo das duas instâncias da asserção é usado para indicar o comportamento correto. A compatibilidade destes dois exemplos depende da semântica do domínio. Para resolver esta questão, os autores devem pensar que duas instâncias do mesmo tipo, são, pelo menos, compatíveis.

3 Propostas utilizando WS-Policy

Nesta seção são apresentados alguns trabalhos que utilizam o framework WS-Policy.

O trabalho de Mietzner *et al*. [2009] apresenta características de implementação de WS-Policy em um Enterprise Service Bus bem como uma arquitetura para pesquisa automática de serviços a partir de políticas definidas utilizando WS-Policy.

Garcia e Todelo [2006] propõem uma abordagem para dar suporte à interação entre *web services*, pensando na Qualidade do Serviço (*Qualiy of Service* - QoS) e apoiando a descoberta, publicação e especificação de serviços baseando-se também em aspectos não funcionais. Eles propõem a extensão do padrão WS-Policy para a inclusão de semântica (via ontologias) nas políticas para definição de atributos de QoS. Eles também propõem a evolução do padrão UDDI para incluir políticas de QoS.

Hung [2004] discute o uso dos conceitos Separação de Funções (*Separation of Duties* – SoD) e Conflitos de Interesse (*Conflict of Interest* – CIR) para Processo de Correspondência de Serviços Web (*Web Services Matchmaking Process* - WSMP).

Baresi *et al*. [2006] propõem a linguage WS-COL (Web Service Constraint Language), baseada em WS-Policy, para monitoramento da qualidade de serviços em tempo de execução.

Schulte *et al*. [2009] propõe melhorias ao WS-Re2Policy, que fora proposta em trabalho anterior. O WS-Re2Policy é uma linguagem compatível com o framework WS-Policy 1.1 que tem por objetivo descrever as reações aos desvios do que foi acordado nas *policies*. Dentre as possíveis reações, podem ser citadas: reinicializaçãodo do serviço, renegociação dos parâmetros SLA ou a seleção de um serviço diferente [Schulte *et al*., 2009]. Dentre essas melhorias, detacam-se: ajuste ao padrão W3C-Recommendation 1.5 do WS-Policy; mais construtos; uso de uma ontologia de QoS para definição dos requisitos e criação de uma ontologia para endereçar as reações. Os autores destacam que muitas destas melhorias poderiam ser implementadas através de extensões XML, entretanto ressaltam que uma abordagem semântica pode trazer vários benefícios. Dentre esses benefícios pode-se citar como exemplo o uso de um *reasoner* que possa detectar a inclusão, equivalência, incompatibilidade, incoerência e conformidade das *policies* [Schulte *et al*., 2009].

Sriharee *et al*. [2004] propõem o uso de ontologia para representar a política dos serviços representado-as utilizando WS-Policy e WS-PolicyAttachment.

Fugini *et al*. [2006] apresenta um modelo que "traduza" as *policies* para uma linguagem de mais alto nível para que o usuário possa escolher os serviços que melhor os atendam.

3.1 Virtualizing Services and Resources with ProBus: The WS-Policy-Aware Service and Resource Bus

Atualmente muitas empresas enfrentam desafios relacionados com flexibilidade e a solução, em muitos casos, é a adoção da Arquitetura Orientada a Serviços (*Service Oriented Architecture - SOA*). Esta arquitetura permite que consumidores de serviços sejam desacoplados dos provedores concretos. O ESB (*Enterprise Service Bus*) torna virtual a implementação concreta dos serviços [Mietzner *et al*., 2009]. Segundo [Hewitt, 2009], o ESB é a principal infra-estrutura que apóia uma arquitetura orientada a serviços (SOA).

Um ESB é capaz de conectar diferentes sistemas que se comunicam via mensagens sem forçar a criação de conexões ponto-a-ponto que são visíveis para aplicações- cliente. Clientes se conectam ao barramento que serve como uma camada de mediação que

protege os clientes de diferentes formatos e protocolo de mensagens usados em sistemas *backend*[1].

Sousa *et al.* [2010] apresentam um estudo do *Enterprise Service Bus* apontando suas principais características. Além disso, eles analisam a implementação de ESB da Oracle, o Oracle Service Bus.

Mietzner *et al*. [2009] apresentam a extensão de um *Enterprise Service Bus* (ESB) para permitir que clientes de serviços submetam políticas às quais os provedores de serviços precisam atender, isso na mesma mensagem que invoca o serviço. É mostrado como estas políticas são avaliadas pelo ESB e como são definidas para serviços *statefull* e *stateless*.

A proposta de Mietzner *et al*. [2009] é a apresentação do ProBus, um ESB capaz de realizar a seleção de serviços e recursos com base em políticas (*policies*). Essa abordagem é baseada no *framework* WS-Policy e é adequada para ambientes muito dinâmicos, onde serviços e recursos aparecem e desaparecem dinamicamente, bem como suas propriedades. Nessa abordagem, a escolha do serviço é delegada ao ESB. O ProBus é uma extensão do ESB chamado Apache Service Mix[2].

Clientes de serviços podem selecionar serviços baseados em propriedades funcionais, descritas no WSDL e não-funcionais, descritas utilizando-se o WS-Policy, por exemplo, buscando um registro no UDDI. Para que isso funcione, o cliente do serviço descreve as propriedades não-funcionais (como requisito de segurança, suporte à transação, custo etc) em uma política de solicitante. O WS-Policy então utiliza um algoritmo de interseção para comparar as políticas do solicitante e do provedor, conforme descrito na seção 2.5.

Mietzner *et al*. [2009] apresentam 3 classes de *binding* dinâmico de serviços e recursos, são elas:

1. Seleção manual do serviço: busca manual pela descrição dos serviços e verifica se as políticas do serviço atendem às necessidades.
2. Seleção dinâmica baseada em políticas: o solicitante do serviço envia a política para o repositório que retorna uma lista de políticas efetivas da qual o solicitante pode escolher uma, o solicitante então envia uma requisição para o *endpoint* que escolhido na lista. Esta implementação existe em *middlewares* padrão, como Apache Axis 2.0.
3. Seleção dinâmica baseada no ProBus: é a proposta de Mietzner *et al*. [2009] (Figura 10). Neste tipo de *binding*, a seleção do serviço baseada em WS-Policy é feita em um único passo. Isso significa que apenas uma mensagem é enviada ao ProBus. Esta mensagem é suficiente para selecionar e usar o serviço. Se nenhum serviço adequado for encontrado, um erro será retornado para o solicitante. A especificação da política desejada é feita na mensagem de invocação.

[1] Backend é qualquer sistema responsável por um grupo específico de dados e funcionalidades, por exemplo: um banco de dados em um SGBD, mainframe, SAP, grupo de servidores JEEE, uma conexão com outra empresa. Do ponto de vista do negócio, um backend é um sistema que tem um papel específico e é mantido por um grupo específico. [Josuttis, 2007]

[2] http://servicemix.apache.org/home.html

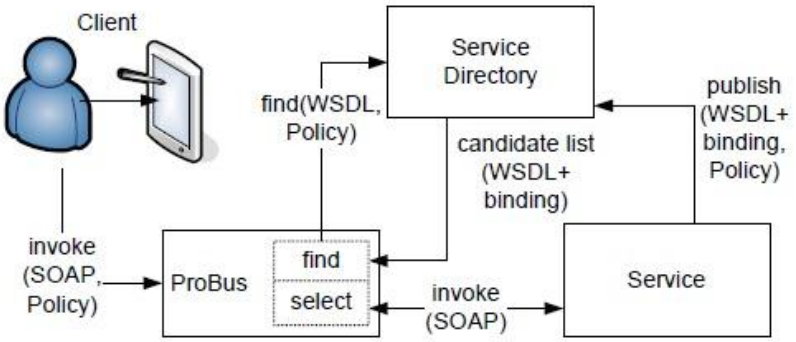

Figura 10 – Fluxo de informações da classe 3 [Mietzner *et al*., 2009].

A Figura 11 mostra a inclusão da política na mensagem SOAP enviada pelo cliente.

```
<soap:Envelope ...>
 <soap:Header>
  <rb:requiredPolicy>
   <wsp:Policy>
    <wsp:ExactlyOne>
     <wsp:All>
      <prn:printer prn:maxCostPerPage="$0.3"/>
     </wsp:All>
    </wsp:ExactlyOne>
   </wsp:Policy>
  </rb:requiredPolicy>
  </soap:Header>
 <soap:Body>
  <!-- Payload of the message -->
 </soap:Body>
</soap:Envelope>
```

Figura 11 – Protocolo SOAP com WS-Policy [MIETZNER et al., 2009].

Mietzner *et al.* [2009] trata em seu trabalho de recursos *stateful*. Um recurso representa um serviço *stateful*, pois seu estado pode estar alterado a cada momento em que é chamado, mesmo quando são passados os mesmos parâmetros de entrada. Um serviço é *stateless* quando sua invocação sempre produz o mesmo resultado, para os mesmos valores de entrada. Essa é uma característica importante, pois permite que os serviços sejam escaláveis. Então, um exemplo de ambiente dinâmico é o caso dos recursos, disponíveis através de serviços *stateful*. Neste caso, o estado atual do recurso precisa ser checado todas as vezes em que for invocado. Suas propriedades são descritas em um documento de propriedade de recurso. Este documento muda freqüentemente e o *binding* do recurso tem que ser dinâmico. Um exemplo seria uma impressora, que possui diversos estados. O framework WS-Resource pode ser usado para prover o documento de propriedades do recurso [Mietzner *et al*., 2009].

A Figura 12 ilustra o funcionamento do ProBus para o caso de uma impressora.

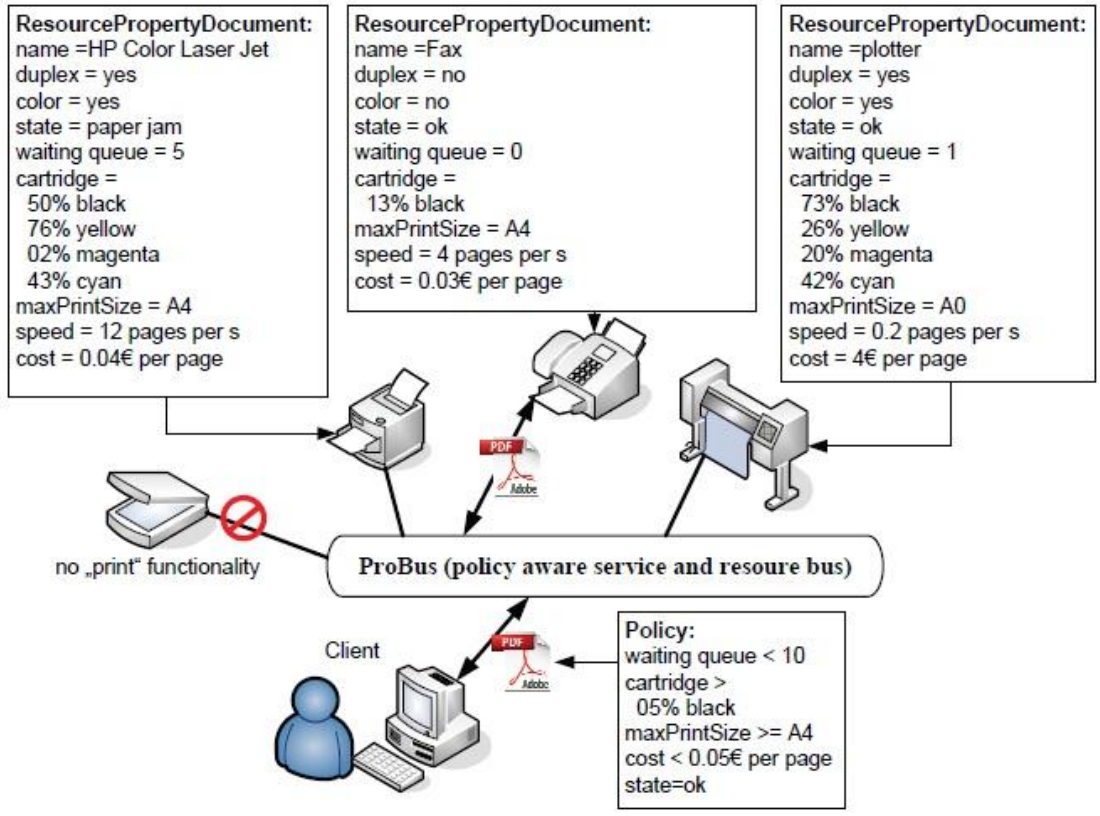

Figura 12 – Funcionamento do exemplo da impressora no ProBus [Mietzner *et al*., 2009].

A abordagem do ProBus para tratar a política dos serviços e recursos apresenta alguns problemas [Mietzner *et al*., 2009]:

- Como o cliente do recurso sabe que um determinado recurso armazenou seus dados para que ele possa recuperar depois?

  Para resolver esta questão, Mietzner *et al*. [2009] apresentam duas abordagens:

  - Solução baseada no EPR (*Endpoint Reference*): o ESB responde a requisição com o EPR do recurso;
  - Solução baseada no ID: é criado um identificador da mensagem, então o ESB sabe qual recurso é responsável por qual conversação (troca de mensagens) armazenando o mapa com o ID da conversação para determinado ERP.

A Figura 13 ilustra estas duas soluções: ERP retornado ao cliente e identificador da requisição retornado ao cliente.

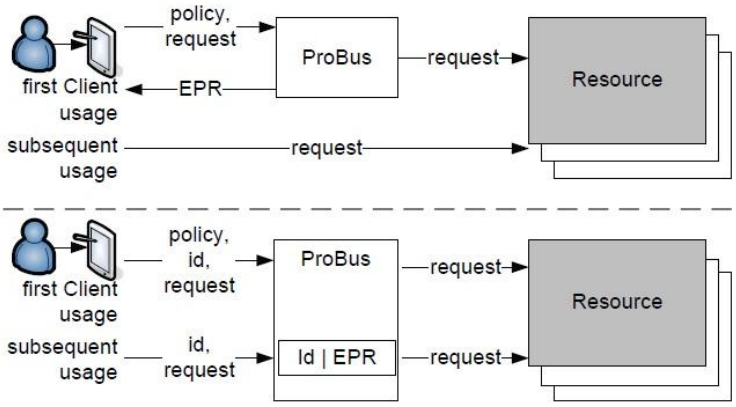

Figura 13 – Soluções para tratar conversações [Mietzner *et al*., 2009].

- Como o cliente pode definir políticas relacionadas às propriedades dos recursos?

  Para resolver este problema é introduzido um novo tipo de asserção de política, chamado condição da propriedade do recurso (*resource property condition*). Um exemplo é apresentado na Figura 14.

```
<wsp:Policy ...>
  <wsp:ExactlyOne>
    <wsp:All>
      <!-- some assertions -->
      <rb:resourcePropertyCondition
          rb:type="prn:printer">
        <rb:XPath
            rb:propertyQName="prn:waitingQueue">
          number(prn:waitingQueue/text()) <10
        </rb:XPath>
      </rb:resourcePropertyCondition>
      <!-- more assertions -->
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Figura 14 – Exemplo de *policy* utilizando Resource Property Type [Mietzner *et al*., 2009].

- Como o ESB seleciona recursos concretos com base nas políticas?

  Para endereçar esta questão, o processador deve executar o seguinte algoritmo: avaliar a política com a política do provedor associado ao *web service* que manipula o recurso; se uma ou mais alternativas forem encontradas, então a política é entregue ao "pós-processamento"; se o "pós-processador" detecta dois ou mais elementos `rb:resourcePropertyCondition` em uma das alternativas da política então é passado ao *WSRFAssertionProcessor*, que executa a requisição do `GetResourceProperty` no próximo recurso do tipo solicitado. A requisição contém o valor do atributo `rb:type` da afirmação do solicitante para indicar qual propriedade do recurso é solicitada. Então é avaliado o resultado do `GetResourceProperty`. Isso é feito até que o *WSRFAssertionProcessor* encontre o recurso apropriado.

- Como o ESB trata a aparição e o desaparecimento dinâmicos do recurso?

Para poder tratar esta questão, o ESB precisa saber quando o recurso é criado ou removido. Quando um novo recurso é criado, a abordagem é permitir que a definição de quais operações serão acessadas no *endpoint* sejam feitas através do ESB, então ele atua como um fábrica de recursos e cria novos recursos. Na remoção de recursos, o ESB sabe que o recurso não existe quando recebe uma exceção do tipo `resourceUnvailable` após invocar uma operação e então remove o recurso de seu registro interno de recursos.

- Como o ESB trata exceções?

  Existem 2 tipos de exceção: exceção de aplicação e exceção de *middleware*. Quando acontece uma exceção de aplicação, o erro é transmitido ao solicitante. Quando ocorre algum erro no *middleware*, o ProBus introduz um novo tipo de exceção chamado `noSuitableResourceFound`. Essa exceção é lançada quando o ProBus não consegue encontrar o recurso que corresponde à política solicitada.

  O protótipo criado é um extensão do Apache ServiceMix, pois ele provê virtualização de *endpoint* e um sofisticado modelo de *deployment*. Após fazer o roteamento de *endpoint* baseado em políticas, outros artefatos relacionados às políticas poderão facilmente ser desenvolvidos. O protótipo permite o registro de políticas para serviços. O protótipo também permite que as operações de fábrica de recursos sejam executadas. Os clientes de serviços podem incluir informações de políticas no cabeçalho da mensagem SOAP e o *middleware* avalia os documentos de propriedades de recursos para encontrar o mais adequado. A Figura 15 apresenta a arquitetura do ProBus.

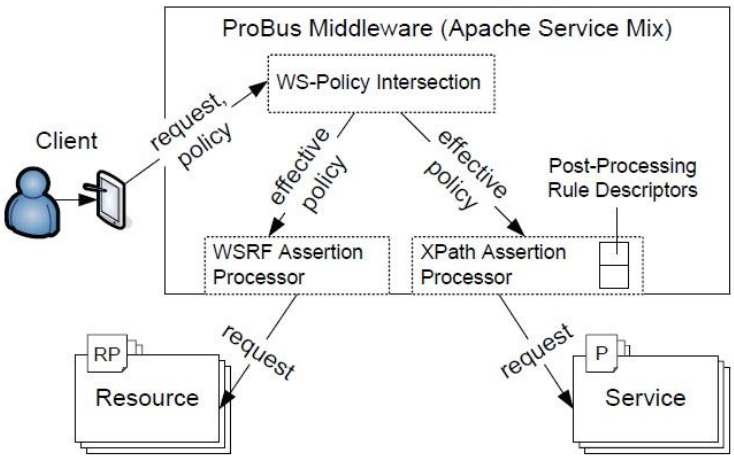

Figura 15 – Arquitetura do ProBus [Mietzner *et al*., 2009].

3.2 Semantics-enriched QoS Policies for Web Service Interactions

Garcia e Todelo [2006] propõem uma abordagem para dar suporte à interação entre *web services*, pensando na Qualidade do Serviço (*Qualiy of Service* - QoS) e apoiando a descoberta, publicação e especificação de serviços baseando-se também em aspectos não funcionais. O resultado do trabalho é uma extensão do WS-Policy para complementar as descrições do WSDL com políticas de QoS enriquecidas de semântica utili-

zando *Ontology Web Language* (OWL) [Patel-Schneider, 2004], padrões da linguagem *ABLE Rule Language* (ARL) [IBM, 2005] e uma extensão do padrão UDDI para incluir políticas de QoS.

A contribuição dos autores é a extensão do WS-Policy para complementar as descrições do WSDL com políticas de qualidade de serviço enriquecidas com semântica e a extensão do UDDI. A proposta permite que sejam selecionados/descobertos serviços que atendam determinados requisitos não funcionais.

Um ambiente típico de *web services* inclui o desacoplamento de serviços e a gestão de políticas que é importante em áreas de QoS. Neste tipo de ambiente, os serviços de diversas indústrias estão envolvidos. Nas abordagens atuais falta uma forma de gerenciar conceitos de diferentes domínios. O framework WS-Policy pode ser utilizado para especificar informações de QoS dos *web services* e complementar a descrição do WSDL. Além disso, o UDDI também precisa ser estendido para permitir a inclusão destas novas informações. Entretanto, o WS-Policy não resolve todos os problemas, pois a semântica não é tratada [Garcia e Todelo, 2006].

Considere, por exemplo, que um provedor especifica que seu serviço suporta um algoritmo particular para ajuste de intervalo de transmissão e o consumidor define uma política solicitando um algoritmo diferente. Se existirem algoritmos compatíveis, mas diferentes, então o algoritmo solicitado poderia ser substituído pelo algoritmo disponível, pois as políticas são compatíveis. Entretanto, o padrão WS-Policy não suporta este tipo de identificação de relacionamento. Para endereçar esse tipo de questão, as políticas de QoS devem ser criadas utilizando conceitos ontológicos [Garcia e Todelo, 2006].

Ontologias podem ser usadas para definir a informação semântica utilizada pelas políticas de QoS. A inclusão de semântica permite que a descrição das asserções de políticas de QoS sejam mais precisas [Garcia e Todelo, 2006].

Os *web services* possuem diversos padrões a serem utilizados no seu desenvolvimento, como WSDL, UDDI e SOAP. Outros padrões têm sido criados, como o WS-Policy para especificar políticas de serviços web, tais políticas podem estar relacionadas com operações de serviços, gestão de *Service Level Agreement* ou regulamentações. Com WS-Policy podemos descrever políticas solicitadas e suportadas, mas não é possível especificar seu significado [Garcia e Todelo, 2006].

OWL é uma linguagem que facilita a interpretação de uma máquina de conteúdo interpretável. Ela pode ser usada para representar o significado de termos de vocabulários e seus relacionamentos. Essa representação é chamada de ontologia. A linguagem de regras é introduzida para dar à OWL habilidade adicional para dedução. A linguagem de regras utilizada é a ARL. Esta linguagem oferece a habilidade de externalizar regras de negócio de maneira formal. Uma regra é uma expressão de conhecimento ou uma indicação declarativa. As regras em ARL podem ser representadas em XML [Garcia e Todelo, 2006].

QoS é a combinação de diversas características não funcionais. A publicação de QoS ajuda a selecionar um serviço dentre vários com a mesma funcionalidade, permite a composição de serviços baseada em QoS e a avaliação de caminhos de execução para adaptação de processos. QoS também pode ser usada como base para custos de modelos que direciona a otimização de processos. O monitoramento da QoS ajuda a realização de níveis de qualidade esperados pelos consumidores e permite que os provedores detectem problemas, acomodem modificações e mantenham o equilíbrio entre custo e performance [Garcia e Todelo, 2006]. A Figura 16 apresenta os atributos de QoS.

Atributos de QOS	Descrição
Tempo de resposta	O tempo que o serviço demora para completar sua tarefa
Latência	O tempo necessário para iniciar o atendimento de uma requisição de serviço
Taxa de transferência	A taxa de processamento de requisição que um serviço suporta
Escalabilidade	O aumento da taxa de transferência em um intervalo de tempo
Capacidade	O número de requisições concorrentes que um serviço permite
Disponibilidade	O percentual de tempo em que o serviço está operando.
Confiabilidade	O tempo de continuidade do serviço correto e para transição para o estado correto
Acurácia	A taxa de erro do serviço em um intervalo de tempo
Robustez	O nível de elasticidade do serviço para entrada incorretas e invocação de sequencias
Estabilidade	A taxa de modificação da interface do serviço
Custo	A medida do custo envolvido na utilização do serviço
Segurança	Define se o serviço oferece mecanismos de confidencialidade, integridade e autenticação
Mensagem confiável	Determina se o serviço oferece mecanismos de garantia de entrega de mensagem confiável
Integridade	Define se o serviço suporta propriedades transacionais
Interoperabilidade	Determina se o serviço é complacente com perfis de interoperabilidade

Figura 16 – Atributos de QoS [Garcia e Toledo, 2006].

As características de QoS podem ser representadas pelas asserções de políticas, já que são críticas para a seleção e uso dos *web services*. As políticas podem ser usadas em diferentes fases do ciclo de vida de um serviço. Na fase de projeto e desenvolvimento, os provedores podem criar políticas especificando como os serviços devem trabalhar. Estas políticas podem combinar políticas independentes de servidor e políticas que especificam características que um serviço disponibilizado em um determinado servidor deve suportar. Na fase de execução, os consumidores podem criar políticas associadas aos serviços dos parceiros. Tais políticas especificam capacidades do consumidor e propriedades que devem ser oferecidas pelos *web services* [Garcia e Todelo, 2006].

O WS-Policy permite a especificação de políticas em diferentes áreas, como privacidade e operação. Entretanto, sem o conhecimento do domínio, potenciais parceiros podem ser rejeitados durante a interseção de políticas, mesmo com políticas compatíveis [Garcia e Todelo, 2006].

A inclusão de semântica no WS-Policy facilita a negociação de serviços. Asserções de política especificadas de forma heterogênea são associadas a conceitos ontológicos utilizando regras de domínio e políticas semânticas herdam o monitoramento de políticas. Políticas enriquecidas com semântica permitem interseções mais precisas que a abordagem sintática. O uso de ontologias permite o *reasoning* [Garcia e Todelo, 2006].

Na abordagem proposta, os provedores precisam especificar capacidades e requisitos de QoS de seus *web services* usando WS-Policy. Tais especificações são extensões da interface do *web service* descritas em WSDL. Além disso, os consumidores podem selecionar *web services* considerando suas capacidades e requisitos também especificados via WS-Policy [Garcia e Todelo, 2006].

Para possibilitar que as políticas de QoS sejam especificadas, as asserções de políticas do WS-Policy são estendidas e a ontologia será utilizada para obter informações semânticas da política. A política de QoS define conceitos independentes de domínio e conceitos dependentes de domínio. A ontologia de QoS permite extensão e possui duas camadas. A camada inferior abrange conceitos genéricos de QoS, uma unidade de medida ontológica para permitir a especificação de métricas, uma unidade de conversão de regras e um operador ontológico de comparação para comparar métricas de atributos particulares de serviços. Na camada superior está a ontologia independente de domínio que define atributos de QoS de propósito geral e sua métrica correspondente. Estes atributos podem ser medidos objetivamente ou subjetivamente, automaticamente ou com intervenção humana. Além disso, ela ainda possui uma ontologia específica de domínio para definir outros atributos de qualidade além daqueles aplicáveis em múltiplos domínios. Por exemplo, um atributo de dedução pode ser adicionado a uma ontologia de seguros [Garcia e Todelo, 2006].

Abaixo são apresentados os componentes da política de QoS [Garcia e Todelo, 2006]:

- Política: é o elemento raiz. Indica a expressão da política, que é uma representação da política em XML;
- Nome, ID: a identificação da política pode ser feita utilizando o atributo Nome, quando a política é associada a uma URI absoluta. O atributo ID é usado quando a política é associada a uma referência no documento XML;
- TipoPolítica: cada política possui um atributo TipoPolítica associado. Ele indica se a política é do consumidor ou do provedor;
- ReferênciaPolítica: permite o reuso de asserções de política entre políticas. Pode ser usado para incluir o conteúdo de uma política em outra.
- Serviço: a política de QoS de um provedor inclui um elemento Serviço para descrever detalhes da implementação do Serviço para cada política especificada. A política de QoS de um consumidor utiliza este elemento para especificar detalhes do tipo de política na qual o serviço se aplica.
- RegraSeleção: o consumidor da política de QoS especifica uma regra para a seleção do serviço. Esta regra define o número máximo de serviços a serem recuperados e o tipo de seleção necessário. Na regra também se indica se os serviços que satisfaçam apenas requisitos funcionais podem ser selecionados se nenhum serviço satisfizer os requisitos de QoS.
- Operadores da política de QoS: os elementos *ExactlyOne* e *All* são os operadores da política de QoS. O primeiro é uma coleção de alternativas de polí-

ticas e o segundo agrupa asserções de políticas em alternativas. Estes operadores podem estar aninhados recursivamente.

Para cada atributo de QoS, a política de asserções de QoS coleta os seguintes componentes:

- Tipo de asserção de política de QoS: o tipo identifica unicamente a asserção de política de QoS.
- OperaçãoServiço: este atributo define a operação de serviço na qual a asserção se aplica;
- Opção: este atributo, definido no WS-Policy, indica se a asserção é opcional;
- ValorDesejado, ValorAceito: cada requisito da asserção deve possuir estes atributos para especificar o valor desejado e o aceito para a afirmação da política de QoS;
- ValorSuportado: este atributo deve ser usado em asserções de capacidade para definir valores da asserção, estes valores são definidos com base nas métricas especificadas para os atributos de QoS.
- Unidade: corresponde a uma medida da unidade de métrica associada ao atributo relacionado à asserção;
- TipoInterseção: este atributo indica se *reasoning* de semântica pode ser usado para determinar compatibilidade da asserção;
- Operador: os seguintes operadores podem ser utilizados – *qoso:less, qoso:greater, qoso:subclass* e *qoso:instance.*

O exemplo da Figura 17 mostra uma política de QoS utilizando WS-Policy estendido.

```
01  <wsps:Policy …
02    xmlns:wsps="…/ontology/wspolicyschema"
03    xmlns:qosi="…/ontology/domainindependent"
04    xmlns:qoso="…/ontology/comparisonoperator"
05    xmlns:qosu="…/ontology/measureunit"
06      <wsps:ExactlyOne>
07        <wsps:All> …
08            <qosi:RMAssertion
09              IntersectionType="wsps:xml"
10              Operator="qoso:equal">
11              <qosi:InactivityTimeout
12                DesirableValue="400000"
13                Unit="qosu:millisecond"
14                IntersectionType="wsps:xml"
15                Operator="qoso:equal"/>
16              <qosi:BaseRetransmissionInterval
17                DesirableValue="1000"
18                Unit="qosu:millisecond"
19                IntersectionType="wsps:owl"
20                Operator="qoso:equal"/>
21              <qosi:ExponentialBackoff
22                IntersectionType="wsps:owl"
23                Operator="qoso:equal"/>
24            </qosi:RMAssertion>
25        </wsps:All>
26      </wsps:ExactlyOne>
27  </wsps:Policy>
```

Figura 17 – Exemplo de política de QoS [Garcia e Toledo, 2006].

O exemplo mostrado na Figura 17 usa uma afirmação que define propriedades de mensagem confiável (linhas 8 até 24) de um *web service* consumidor. Um prefixo de namespace é usado para indicar o namespace onde o componente está sendo definido, no exemplo, os prefixos wsps, qosi, qoso e qosu são associados às URIs dos namespaces das ontologias do framework WS-Policy, independente de domínio, do operador de comparação e da unidade de medidas, respectivamente. Na linha 08, o tipo *RMAssertion* indica uma afirmação que especifica que o protocolo de mensagem Confiável do Serviço Web tem que ser usado para gerenciar a entrega de uma sequência de mensagens. A afirmação aninhada *InactivityTimeout*, na linha 11, define o perído de inatividade para a sequência de mensagens. A afirmação *BaseRetransmissionInterval*, na linha 16, é usada para especificar por quanto tempo o serviço deve esperar após transmitir uma mensagem e antes de retransmitir a mensagem. A afirmação *ExponentialBackoff* define que um algoritmo de *backoff* exponencial tem que ser usado para ajustar o intervalo de retransmissão (linha 21).

Uma asserção de uma capacidade de uma política de QoS (CA - *capability QoS policy assertion*) satisfaz os requisitos de uma asserção de política de QoS (*RA - requirement QoS policy assertion*) se o tipo de CA satisfaz o tipo de RA e se o valor de CA satisfaz o valor desejado ou aceito, se RA possuir pelo menos 1 valor [Garcia e Todelo, 2006].

A lista de potenciais provedores contém serviços com pelo menos um alternativa de política de QoS que pode ser satisfeita por completo pelas capacidades do consumidor e pode satisfazer completamente os requisitos do consumidor [Garcia e Todelo, 2006].

O conhecimento de domínio nas ontologias e regras de negócio ajuda a identificar relacionamentos entre asserções que não poderiam ser identificadas com base apenas

na sintaxe das políticas [Garcia e Todelo, 2006]. A Figura 18 mostra um exemplo de ontologia com regras de negócio para permitir essa identificação.

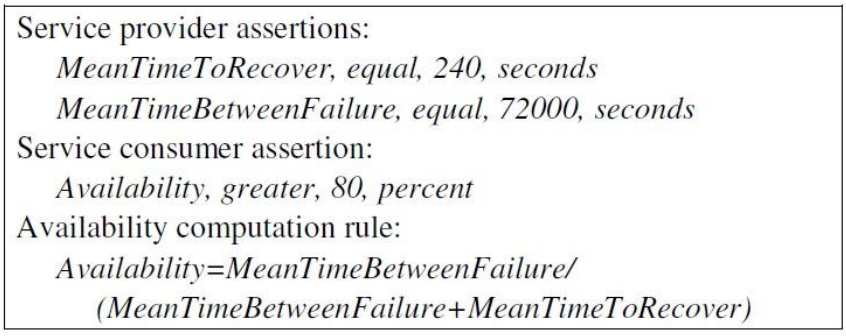

Figura 18 - Exemplo de políticas de QoS com conhecimento do domínio [Garcia e Toledo, 2006].

Na Figura 18, três asserções de políticas são apresentadas. Duas delas são definidas pelo provedor e uma pelo consumidor. Além disso, uma regra de domínio é apresentada [Garcia e Todelo, 2006].

No UDDI padrão faltam facilidades para descrição de QoS. O WS-Policy oferece um meio de especificar QoS. Os provedores devem especificar seus serviços utilizando WSDL e WS-Policy e publicar seus serviços. Na abordagem de Garcia e Toledo [2006], o UDDI é estendido para incluir políticas de QoS. Estas extensões nos mecanismos de busca e registro realçam a flexibilidade da busca. O modelo de informação do UDDI é composto por estruturas de dados expressas através de XML, que são armazenadas nos registros do UDDI [Garcia e Todelo, 2006]. A Figura 19 mostra o modelo de informação do UDDI estendido, com a inclusão da estrutura *qosPolicy* e seus relacionamentos.

Os tipos de estruturas de dados que compõem o modelo de informação do UDDI são:

- *businessEntity*: estrutura de nível mais alto no UDDI que contém informações descritivas sobre a organização provedora, tais como contato e classificação. Cada *businessEntity* pode prover inúmeros serviços.
- *businessService*: representa um serviço web que pode ter inúmeras implementações, inclui informações descritivas do serviço, como nome e classificação;
- *bindingTemplate*: represente a implementação de um serviço e provê informações necessárias para fazer o "bind" com o serviço, permitindo sua invocação. Contém informações de ponto de acesso e protocolo de transporte.
- *tModel* (*Technical Model* – Modelo técnico): representa um conceito único no UDDI, tais como namespaces e sistemas de categorização. Exemplos incluem *tModels* baseados em WSDL e outros documentos que especificam as interfaces do serviço;
- *publisherAssertion*: define uma comunidade de *businessEntities*. Pode ser usado por organizações para exportar seus relacionamentos, como empresas subsidiárias e consórcios;
- *subscription*: descreve uma requisição para acompanhar as atividades no repositório UDDI de acordo com as preferências do requisitante. Podem ser

registradas subscrições para receber informações sobre mudança em quaisquer estruturas do UDDI;

- *qosPolicy*: representa uma política de QoS para um serviço web. As informações disponibilizadas descrevem atributos do serviço web representado pelo *bindingTemplate*. Referências ao tModels podem ser usadas para indicar que a política de QoS está em conformidade com uma especificação particular.

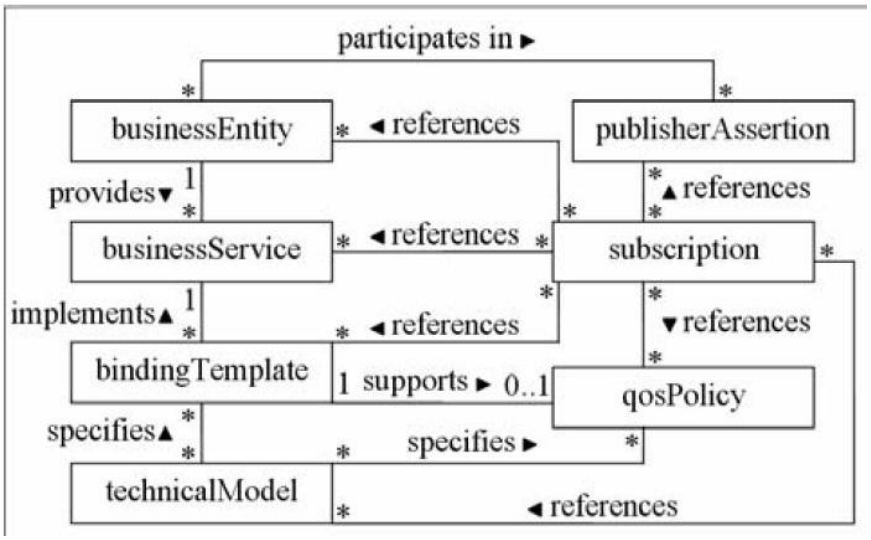

Figura 19 – Modelo de informação do UDDI [Garcia e Toledo, 2006].

Para facilitar o acesso ao registro dos serviços é utilizado o *Broker* para prover operações de gestão da QoS. Ele utiliza API do UDDI estendida para recuperar dados dos registros e para publicar e atualizar dados do UDDI. O *Broker* em si é um *web service* [Garcia e Todelo, 2006].

O *Broker* obtém políticas de QoS dos provedores e publica nos registro do UDDI. Isso ajuda os consumidores a selecionar serviços de acordo com seus requisitos funcionais e de QoS. O *Broker* mapeia as políticas de QoS dos provedores expressas em WS-Policy em estruturas de *qosPolicy* do UDDI estendido. Os consumidores fornecem políticas de QoS ao *Broker*, que as utiliza para selecionar os *web services* registrados no UDDI [Garcia e Todelo, 2006].

Para implementar a descoberta de serviços enriquecida com QoS, o *Broker* executa dois passos: primeiro ele descobre os serviços que satisfazem os requisitos funcionais, da forma tradicional; depois, com base na lista de serviços retornados pelo primeiro passo, ele executa a interseção de políticas de QoS. Para fazer isso, o *Broker* utiliza uma *procedure* para determinar a compatibilidade das políticas. A partir de então são realizados outros dois passos. Primeiro, é determinado se as políticas de QoS possuem o mesmo vocabulário e depois são utilizadas informações de semântica para descobrir a compatibilidade das políticas. Para isso, o *Broker* decompõe as políticas de QoS em asserções. Para que esta decomposição seja feita, as políticas precisam estar normalizadas. Existe uma operação no WS-Policy para converter as políticas para o formato padrão. Quando o serviço com a QoS adequada é encontrado, essa informação é passada ao consumidor [Garcia e Todelo, 2006].

O *Broker* foi implementado utilizando JDK e Apache Axis. A extensão do WS-Policy foi implementada utilizado Apache WS-Commom/Policy. A extensão do UDDI foi

feita utilizando-se Apache jUDDI. Para implementar a base de dados do UDDI foi utilizado o MySQL AB MySQL. O servidor Apache Tomcat foi utilizado para *deploy* dos serviços e para execução de experimentos [Garcia e Todelo, 2006].

Os experimentos avaliaram se as extensões propostas permitiam que os consumidores selecionassem provedores adequados, com relação às políticas de QoS, além dos requisitos funcionais. Os resultados mostraram que as extensões podem ser incluídas na arquitetura dos *web services* para refinar a descoberta dos serviços sem comprometer o desempenho. Para medir os impactos, em termos de tempo para seleção dos serviços, o tempo de processamento das requisições de serviços considerando QoS foi comparado com o tempo de processamento destes mesmos serviços considerando apenas a funcionalidade básica de busca. Um cenário com 500 descrições de *web services* publicadas no UDDI foi simulado [Garcia e Todelo, 2006]. Como esperado, a avaliação dos resultados mostra um aumento no tempo médio de processamento, conforme apresentado na Figura 20.

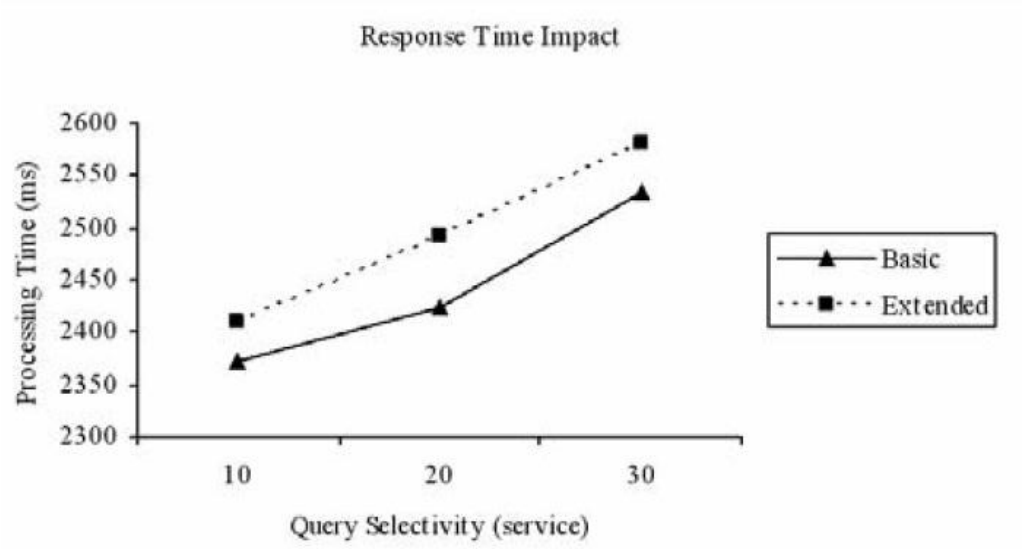

Figura 20 – Tempo de resposta com QoS e sem QoS [Garcia e Toledo, 2006].

3.3 From Conflict of Interest to Separation of Duties in WS-Policy for Web Services Matchmaking Process

Hung [2004] apresenta o conceito de segurança chamado Separação de Funções (*Separation of Duties* – SoD) para o Processo de Correspondência de Serviços Web (*Web Services Matchmaking Process* - WSMP). Este trabalho discute a relação entre SoD e outro conceito chamado Conflitos de Interesse (*Conflict of Interest* – CIR), pensando em WSMP. Estes dois conceitos são implementados em WS-Policy. É apresentado um algoritmo de correspondência (*matchmaking*) para maximizar o nível de SoD. Por fim, é apresentado um protótipo com os serviços "CIRService" e "SoDService" para suportar CIR e SoD na correspondência (*matchmaking*) de processos.

3.4 WS-Policy for Service Monitoring

Este trabalho apresenta um *framework* de monitoramento de processos especificados utilizando-se BPEL (Business Process Execution Language) [Baresi *et al.*, 2006]. Para especificar os requisitos dos usuários na execução de composição de *web services* é

proposta a WS-Col (Web Service Constraint Language), uma linguagem independente de domínio e em conformidade com o WS-Policy. O trabalho se concentra em requisitos não funcionais, mesmo com a WS-Policy e a WS-Col permitindo tratamento tanto para requisitos funcionais como para não funcionais. O motivo é o foco em segurança, apresentado pelos autores como uma das dimensões mais desafiadoras quando se pensa em QoS para aplicações desenvolvidas utilizando *serviços web*.

Baresi *et al*. [2006] afirmam que WS-Policy se tornou um *framework* amplamente utilizado. Devido a isso, várias linguagens foram propostas no sentido de prover um conjunto de assertivas dependentes de domínio, tudo em conformidade com o *framework* WS-Policy. WS-SecurityPolicy e WS-ReliableMessaging Policy são exemplos dessas linguagens [Baresi *et al*., 2006].

A linguagem WS-CoL, proposta por Baresi *et al*. [2006], é similar às linguagens citadas anteriormente no sentido que também é um conjunto de assertivas em conformidade com o *framework* WS-Policy. Entretanto, diferencia-se por ser independente de domínio. O principal objetivo dessa linguagem é expresar limitações de monitoramento [Baresi *et al*., 2006].

Baresi *et al*. [2006] não abordam a declaração das políticas dos consumidores e dos provedores. Tão pouco detalham como é feita a interseção entre essas políticas. O foco da proposta dos autores está no monitoramento da política que foi acordada, ou seja, no resultado da interseção entre as políticas. Desta forma, os processos especificados em BPEL são monitorados em tempo de execução, para que seja verificado a todo instante se os níveis de qualidade de serviço acordados nas políticas estão sendo cumpridos.

A WS-CoL não é uma linguagem totalmente nova. Baresi *et al*. [2006] argumentam que para aumentar a difusão e a aceitação da linguagem, seu desenvolvimento foi baseado em duas linguagens de assertivas já existentes: Annotated ADA (Anna) e Java Modelling Language (JML).

Uma das preocupações de Baresi *et al*. [2006] foi em separar a lógica de controle da lógica do negócio. Além disso, os autores se preocuparam também com o *tradeoff* entre monitoramento e desempenho. Por isso, as diretivas de monitoramento foram propostas como *policies* de monitoramento *stand alone*, renderizadas pelo *framework* WS-Policy. Desta forma as *constraints* não são incorporadas ao processso BPEL, sendo que é o processo BPEL que escolhe a *policy* de monitoramento a ser utilizada. Esta escolha é feita através de uma solicitação ao *monitoring manager*. Esta é a única modificação no processo BPEL: chamadas ao *monitoring* manager. Esta independência garante que as *policies* possam ser modificadas sem necessidade de alteração do processo BPEL. Além disso, as *policies* de monitoramento podem ser ligadas e desligadas em tempo de execução, assim como ter suas prioridades alteradas. Isso garante controle por parte do desenvolvedor no impacto que o monitoramento tem na execução do processo.

3.4.1 Notação

Esta seção descreve a notação da WS-CoL, utilizando como exemplo um serviço de compra de livros *on-line*, utilizado no trabalho de Baresi *et al*. [2006].

Os dados a serem analisados podem vir do próprio processo ou de fontes externas. A instrução BPEL "`$<nome da variável>\<parte da variável>`" permite acesso a variáveis internas do processo. Como essas variáveis podem ter várias partes,

por serem instâncias de esquemas XML, a instrução permite que diferentes partes da variável sejam acessadas [Baresi *et al.*, 2006].

Quando os dados a serem analisados encontram-se em uma fonte externa, a seguinte instrução é utilizada: `\return[Int|String|Boolean](WSDL, OpName, <parameters>)`. Os parâmetros `WSDL`, `OpName` e `<parameters>` representam, respectivamente, a URL do WSDL relacionado ao *data collector*, a operação suportada pelo *data collector* e os valores requisitados pela operação.

A Figura 21 apresenta um exemplo de *policy attachment* e de *policy definition*, respectivamente.

```
<wsp:PolicyAttachment xmlns:wsp="...">
    <wsp:AppliesTo xmlns:wsal="...">
        <wscol:MonitoredItems xmlns:wscol="...">
            <wscol:MonitoredItem type="precondition"
            path='XPATH expression to WS-BPEL invoked
            activity'/>
        </wscol:MonitoredItems>
    </wsp:AppliesTo>
    <wsp:PolicyReference
        URI="http://www.bookshop.it/policies#BookShopPoli
    cy/>
</wsp:PolicyAttachment>

<wsp:Policy xml:base="http://www.bookshop.it/policies"
    wsu:Id="BookShopPolicy"
    xmlns:wsp="..."
    xmlns:wsu="...">
    <wsp:All xmlns:wsse="..."
        xmlns:wscol="...">
        <wsse:Confidentiality>
            <wsse:Algorithm type="wsse:AlgSignature"
            URI="http://www.w3.org/2000/09/xmlenc#3des-
        cbc"/>
        </wsse:Confidentiality>
        <wscol:Expression>
            ($ChargeRequest\amount) <=
            \returnInt(WSDL_XPATH, applyXPATH,
            '\\userpref\moneyCap', up.xml)
        </wscol:Expression>
    </wsp:All>
</wsp:Policy>
```

Figura 21 - Exemplo de policy attachment e de policy definition

No exemplo apresentado na Figura 21, a *policy* `BookShopPolicy` da *policy definition* é aplicada de acordo com o resultado da expressão XPath da tag `MonitoredItem` da *policy attachment*, definindo, desta forma, o que vai ser monitorado. Além disso, todas as mensagens devem ser criptografadas usando "3DES". E a tag `<wscol:Expression>` indica que a compra só pode ser feita se o valor da compra (`$Chargerequest`) for igual ou menor ou igual `moneyCap` das preferências do usuário [Baresi *et al.*, 2006].

O exemplo apresentado anteriormente também permite observar o uso de variáveis internas e de fontes de dados externas. A expressão `$ChargeRequest\amount` se refere a uma variável interna do processo BPEL. Já a expres-

são`\returnInt(WSDL_XPATH, applyXPATH,'\\userpref\moneyCap', up.xml)` retorna um valor obtido no arquivo `up.xml` [Baresi *et al.*, 2006].

3.4.2 Monitor Manager

Esta seção tem por objetivo apresentar o *monitor manager*¸ referenciado na seção anterior.

O *monitor manager* é composto de quatro partes: *rules manager*, *configuration manager*, *external monitors* e *invoker*. Na Figura 22 é possível observar uma representação do *monitor manager*.

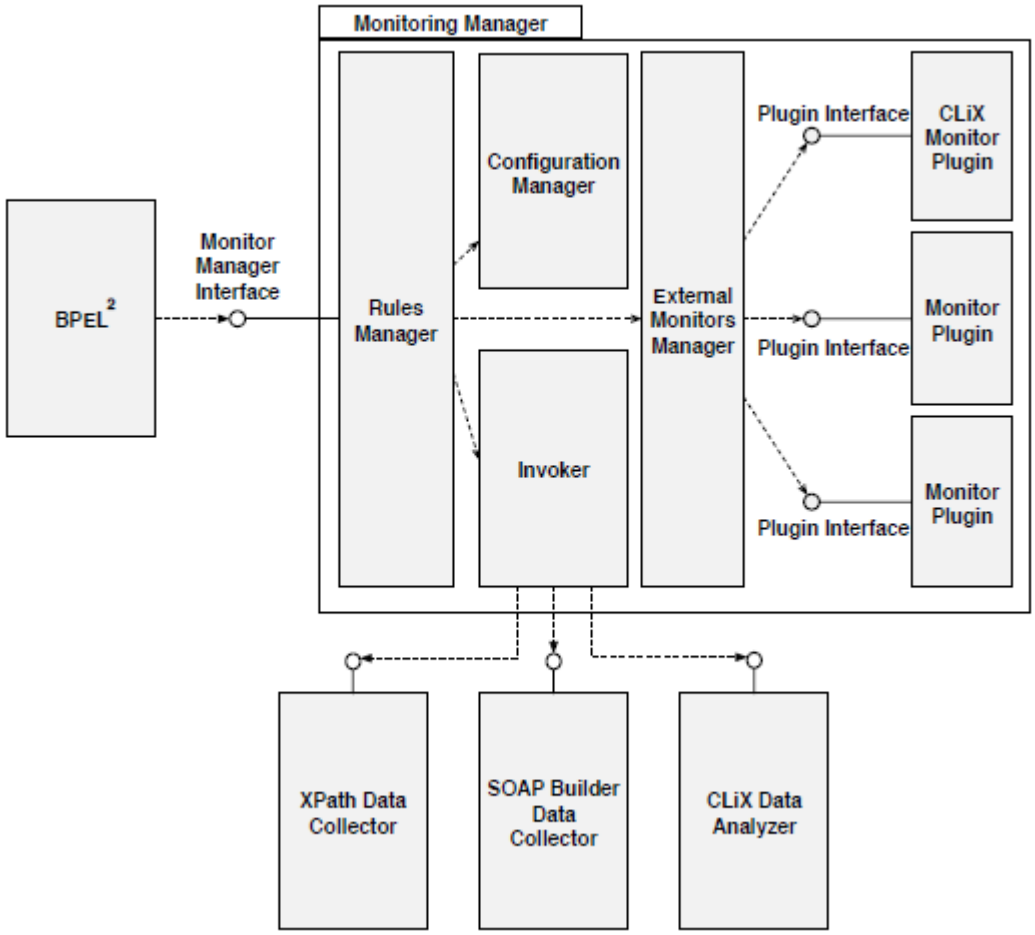

Figura 22 – Componentes do monitor manager [Baresi *et al*., 2006]

Na Figura 23 é apresentado o diagrama de colaboração UML. Através desta figura é possível observar como os componentes interagem durante a execução de um processo BPEL [Baresi *et al.*, 2006].

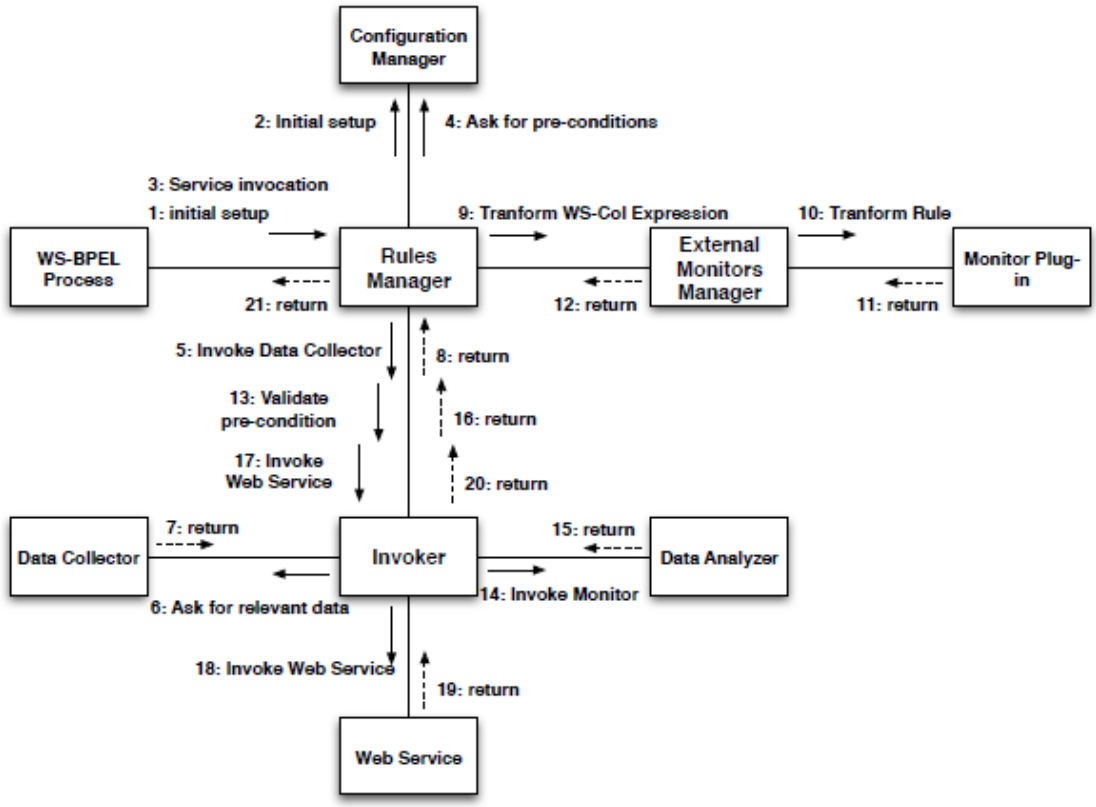

Figura 23 – Interação entre os elementos do monitor manager [Baresi *et al*., 2006]

A interação entre o processo BPEL e o *monitor manager* tem início quando o processo BPEL invoca o *monitor manager*, conforme Figura 23 (1). Desta forma, tem-se o acionamento da atividade de monitoramento através da criação de uma configuração no *configuration manager*, que contém todas as *policies* selecionadas para o processo, conforme Figura 23 (2). Se o processo desejar invocar um serviço fazendo uso de monitoramento, ele o faz através do *monitor manager*, conforme Figura 23 (3). O componente *rules manager* extrai as expressões associadas com a invocação do serviço através do *configuration manager*, conforme Figura 23 (4). Os passos (5,8) são acionados quando dados adicionais precisam ser obtidos. A interação com os *data collectors* adicionais pode ser vista em (6,7). Os passos (9, 10, 11 e 12) só são executados depois que todos os dados são obtidos. É neste momento que o *rules manager* traduz a expressão WS-CoL num formato que o monitor externo compreenda. O *data analyzer* é invocado em (13, 14, 15 e 16) e o *rules manager* aguarda por uma resposta. Em (17, 18, 19, 20, 21) o serviço é invocado, caso a resposta seja positiva (em caso negativo, poderia ter sido retornado um erro). A partir de então o monitoramento é iniciado. Durante o monitoramento, verifica-se, por exemplo, se as mensagens SOAP enviadas ao serviço OnlineBank estão criptografadas conforme estabelecido (no exemplo da seção anterior a criptografia especificada era 3DES).

Como trabalho futuro, Baresi *et al*. [2006] destaca a tradução automática das asserções WS-Policy em expressões WS-CoL. Ainda assim, os autores destacam que estas atividades são facilitadas pelo uso do *framework* [Baresi *et al*., 2006].

3.5 From Web Service Policies to Automatic Deviation Handling - Supporting semantic description of reactions to policy violations

Schulte *et al*. [2009] destacam a ampla utilização da arquitetura orientada a serviços (SOA) na atualidade. Segundo os autores, o uso desse tipo de arquitetura conduz a uma necessidade dos consumidores de serviços conhecerem as funcionalidades e o nível de qualidade de serviços (QoS) dos serviços que eles desejam consumir. Comumente essas informações são publicadas em um acordo de nível de serviço (*Service Level Agreement* - SLA) [Schulte *et al*., 2009].

Apesar dos consumidores de serviços poderem fazer uso dos SLAs, Schulte *et al*. [2009] destacam um problema: "o que fazer quando um provedor de serviço não atender aos requisitos do consumidor?".

Os autores defendem que apesar dos requisitos de consumo de serviços serem bem endereçados pelas *policies*, há poucos trabalhos que endereçam o que fazer quando ocorrem desvios [Schulte *et al*., 2009].

De forma semelhante a Baresi *et al*. [2006], Schulte *et al*. [2009] já haviam proposto em trabalhos anteriores uma forma de se monitorar a qualidade do serviço que está sendo oferecido. O WS-Re2Policy é uma linguagem compatível com o framework WS-Policy 1.1 que tem por objetivo descrever as reações aos desvios do que foi acordado nas *policies*. Dentre as possíveis reações, podem ser citadas: reinicializaçãodo do serviço, renegociação dos parâmetros SLA ou a seleção de um serviço diferente [Schulte *et al*., 2009].

Neste trabalho Schulte *et al*. [2009] propõe melhorias ao WS-Re2Policy. Dentre essas melhorias, detacam-se: ajuste ao padrão W3C-Recommendation 1.5 do WS-Policy; mais construtos; uso de uma ontologia de QoS para definição dos requisitos e criação de uma ontologia para endereçar as reações. Os autores destacam que muitas destas melhorias poderiam ser implementadas através de extensões XML, entretanto ressaltam que uma abordagem semântica pode trazer vários benefícios. Dentre esses benefícios pode-se citar como exemplo o uso de um *reasoner* que possa detectar a inclusão, equivalência, incompatibilidade, incoerência e conformidade das *policies* [Schulte *et al*., 2009].

3.5.1 Notação

Esta seção tem por objetivo apresentar a linguagem WS-Re2Policy 2.0.

A linguagem WS-Re2Policy 2.0, uma melhoria da WS-Re2Policy, faz uso da linguagem *Semantic Annotations for WSDL and XML Schema* (SAWSDL) [SAWSDL, 2011] para representação semântica das QoS *policies*. A SAWSDL provê o atributo *modelReference*, utilizado para modelar referências descritas em WSDL. O *modelReference* pode ser utilizado para descrever assertivas ou uma reação, e provendo uma referência aos respectivos conceitos relacionados de forma semântica [Schulte *et al*., 2009]. Schulte *et al*. [2009] ressaltam, entretanto, que o WS-Re2Policy 2.0 é apenas uma linguagem descritiva, sendo que os mecanismos de comparação (*match*) dos requisitos e o início das reações devem ser implementado nas aplicações que fazem uso do WS-Re2Policy.

A abordagem adotada por Schulte *et al*. [2009] culminou com a necessidade de se definir uma ontologia, assim como os elementos do WS-Policy que seriam utilizados. Segundo os autores, há uma grande quantidade de ontologias que descreve QoS, entretanto nenhuma endereça as possíveis reações quando um requisito não é atendido

[Schulte *et al*., 2009]. Schulte *et al*. [2009] destacam que a ontologia proposta não teve por objetivo ser independente de domínio, sendo restrita à taxonomia usada na OWL DL [Patel-Schneider, 2004].

A principal classe da proposta é a *Reaction*. Esta classe possui as subclasses *AutomaticReaction* e *ManualReaction*. As reações automáticas possuem um valor padrão que indica por quanto tempo a reação deve ocorrer ou com que frequência, sendo que o tipo *unit* define a unidade de medida. As instâncias da subclasse *AutomaticReaction* podem ser: *Delegate*, *Restart*, *Sleep* e *Logging* [Schulte *et al*., 2009]. A instância *Delegate* indica que a invocação do serviço deve ser delegada a outro *host* de serviços, enquanto a instância *Restart* indica que o serviço deve ser reiniciado. *Sleep* indica que o serviço deve aguardar antes da próxima etapa e *Logging* indica como deve ser feito *log* da reação [Schulte *et al*., 2009].

Na **Erro! Fonte de referência não encontrada.** é possível observar alguns contrutores da WS-Re2Policy 2.0.

Tabela 1 – Construtures do WS-Re2Policy 2.0 [Schulte *et al*., 2009]

Construtor	Uso e comentários
re2:RequirementsReactionsSuite	Conjunto de requisitos e reações obtidos de re2:Requirements e re2:Reactions
re2:Requirements wsp:Policy qos:XXXXXXX	Conjunto de requisitos (*policy assertions*). Coleção de *policy assertions*. *Policy assertion* relacionada a QoS, sendo que o atributo sawsdl:modelReference pode apontar para um conceito na ontologia QoS.
re2:Reactions re2:Sleep, re2:StartService, re2:DelegateControl re2:LogReactions re2:Usage	Coleção de reações que descrevem o que deve ser feito se uma *policy assertion* não for atendida. Exemplos de reações. Podem ter o atributo re2:Usage ou um apontamento para a ontologia QoS (sawsdl:modelReference). Pode assumir os valores required, optional e final.

As reações são executadas na sequência em que são encontradas em `re2:RequirementsReactionsSuite`. Por padrão, o processamento de reações é interrompido caso uma reação tenha sido executada com sucesso. Esse comportamento pode mudar em duas circunstâncias: quando a reação possui os valores `final` ou `required` . Caso a reação tenha o valor `final` , nenhuma reação é executada posteriormente. Entretanto, caso o valor da reação seja `required` , ela será executada independente da anterior ter sido executada com sucesso ou não [Schulte *et al*., 2009].

A Figura 24 apresenta um exemplo de reação na linguagem WS-Re2Policy 2.0.

```
<re2:Reactions RequirementsID="3428">
  <re2:Sleep re2:usage="required"
   sawsdl:modelReference="http://www.kom.tu-
   darmstadt.de/~schulte/reactions20090715.owl#Sleep">10000
  </re2:Sleep>
  <re2:RestartService re2:usage="required"
   sawsdl:modelReference="http://www.kom.tu-
   darmstadt.de/~schulte/reactions20090715.owl#Restart">4
  </re2:RestartService>
  <re2:DelegateControl re2:usage="optional"
   sawsdl:modelReference="http://www.kom.tu-
   darmstadt.de/~schulte/reactions20090715.owll#Delegate">peer
  </re2:DelegateControl>
  <re2:LogReactions re2:usage="final"
   sawsdl:modelReference="http://www.kom.tu-
   darm-
   stadt.de/~schulte/reactions20090715.owl#Logging">Mail_Admin
  </re2:LogReactions>
</re2:Reactions>
```

Figura 24 - Exemplo de reação na linguagem WS-Re2Policy 2.0 [Schulte *et al.*, 2009].

No exemplo é possível observar que a primeira reação vai aguardar 10000 unidades de tempo antes de executar a próxima etapa. A próxima etapa, por sua vez, tentará reiniciar o serviço por quatro vezes. Como a segunda reação é do tipo `required`, entende-se que ela será executada independente da anterior ter sido feita com sucesso ou não. Já a terceira reação só será executada caso a segunda não tenha tido sucesso, uma vez que ela é do tipo `optional`. A função da terceira reação é repassar o controle para outro *peer*. A última ação descreve o *log*, que enviará email para o administrador após os passos terem sido executados.

Diferentemente de Baresi *et al*. [2006], Schulte et al. [2009] não chegaram a implementar a aplicação que faz a monitoria em si.

3.6 On Using WS-Policy, Ontology, and Rule Reasoning to Discover Web Services

Sriharee *et al*. [2004] propõem uma abordagem baseada em comportamento para a descoberta dos *web services*. As regras de negócio que governam o comportamento do *web service* são descritas como políticas. Nessa abordagem, a política é representada através de informação ontológica e é baseada nas ações relativas ao serviço e nas condições para realizá-las. Para associar a política ao serviço é usado o WS-Policy e o WS-PolicyAttachment.

No UDDI convencional é feita a correspondência dos valores dos atributos dos consumidores com os valores publicados pelo provedor do serviço. Neste trabalho, é considerada a adição de regras de negócio à busca convencional baseada nos atributos publicados no UDDI.

Sriharee *et al*. [2004] apresentam alguns exemplos de políticas, relacionadas com o universo da cadeia de suprimentos. Um varejista pode ter uma política para tempo de entrega de aparelhos com base no tipo de aparelho e na área de destino da compra. O cliente pode ter a política de comprar um aparelho de um varejista que possa realizar a entrega em determinado número de dias.

As políticas associadas ao *web service* podem ser ligadas (*bound*) a entidades do *web service* como operação, mensagem, "*part*", etc. As assertivas de políticas serão modeladas pelas ações relacionadas ao serviço e as condições para executá-lo.

Sriharee *et al*. [2004] listam alguns trabalhos relacionados e indicam que a maioria deles possui foco em políticas administrativas, relacionadas com segurança e controle de recurso. A abordagem proposta por Sriharee *et al*. [2004] expressa a política baseada em regras na perspectiva das funções de negócio e tais regras serão utilizadas na descoberta de serviço para restringir as consultas e ser avaliado no momento da consulta. A política será implantada para o padrão UDDI e o framework WS-Policy será utilizado. Segundo os autores, até a data de publicação do artigo, nenhum outro trabalho propôs uma política com base neste *framework* para apoiar a descoberta de serviços.

Sriharee *et al*. [2004] apresentam a motivação para uso de políticas no domínio da cadeia de suprimento com o exemplo a seguir:

1) Um varejista precisa fazer um pedido de compra para um distribuidor que entregue 100 peças em 5 dias;
2) O distribuidor recebe o pedido de compra citado acima e precisa selecionar candidatos a fornecedor, através do UDDI, considerando a existência de contrato ou fornecedores daquelas peças;
3) Os candidatos a fornecedor podem publicar uma política referente a sua função empresarial. Exemplos de ações a serem tomadas: entregar peças ou verificar inventário;
4) Os fornecedores que atendem ao varejista serão retornados.

O foco do trabalho proposto por Sriharee *et al*. [2004] está no item 3 e as políticas de regras de negócio são incorporadas aos serviços individuais. A Figura 25 apresenta as políticas das regras de negócio para entrega de peças descritas acima.

```
Perform DeliverParts(days ≤  3)
       IF (numberOfParts  ≤ 100)
Perform DeliverParts(days ≤  10)
       IF (100  <  numberOfParts  ≤ 500)
Perform ContactInventoryBeforeConfirm
       IF (numberOfParts  > 500)
```

Figura 25 - Políticas de regras de negóciopara entrega de peças [Sriharee *et al*., 2004]

Para representar as políticas de regras de negócio é proposta uma ontologia de alto nível (*upper ontology*). Nessa ontologia são definidos os conceitos e as relações entre os conceitos para representar tais políticas. A ontologia proposta é apresentada na Figura 26.

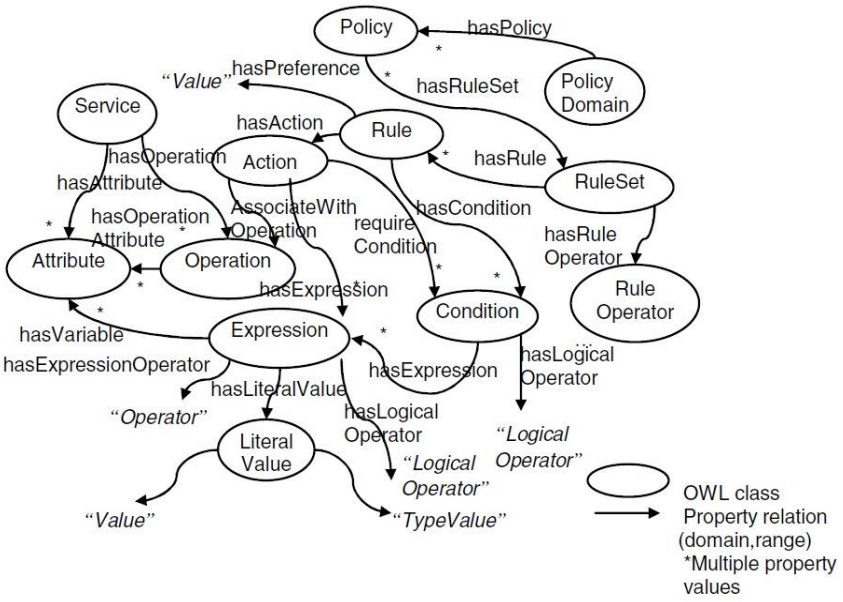

Figura 26 - Ontologia de alto nível para Políticas de regras de negócio [Sriharee *et al*., 2004].

Os principais componentes da ontologia são descritos a seguir:

- *Policy Domain*: é o domínio do serviço no qual a política é aplicada. Por exemplo, cadeia de suprimentos. O domínio da política pode ter diversas políticas. Por exemplo, no domínio da cadeia de suprimentos, pode haver uma política para entrega de peças e outra política relacionada com pagamentos;
- *Policy*: define restrições para uma tarefa particular de um domínio. Uma política pode ter diversos *Rule Sets*. Por exemplo, a política de entrega de peças pode ter um conjunto de regras para tempo de entrega e outro para meios de entrega;
- *Rule Set*: define um conjunto de regras relacionadas a um determinado aspecto;
- *Rule Operator*: é definido para um conjunto de regras para determinar o nível de cumprimento (*enforcement*) da regra dentro de um conjunto de regras. O operador da regra pode ser "All" para indicar que todas as regras do conjunto serão aplicadas, "ExactlyOne" para indicar que apenas uma regra será aplicada em determinado momento ou "OneOrMore" para indicar que pelo menos uma das regras é aplicável.
- *Rule*: é uma declaração de condição-e-ação que diz que ação será tomada quando uma determinada condição for satisfeita. Na Figura 25, três regras são definidas. Uma regra pode ser *tagged* com uma preferência que indica o grau de satisfação quando uma regra é *matched*. A preferência pode ajudar na elaboração do *ranking* de políticas com regras *matched*.
- *Action*: é um termo abstrato que pode ser associado a uma operação de um serviço e expressões que podem ser ligadas a operadores lógicos. Por exemplo, a ação de entrega de partes da peça pode ser associda a operação de compra de partes de um serviço web *fornecedor de partes da peça* e na Figura 26, na primeira regra, é associada à expressão `(days ≤ 3)`.

- *Condition*: é um termo abstrato que pode ser associado com expressões que podem ser *bound* a operadores lógicos.
- *Expression*: especifica uma expressão composta por *Attribute*, *Expression Operator* e *Literal Value*. Na primeira regra da Figura 25, `(numberOfParts ≤ 100)` é uma expressão para a condição da regra onde `numberOfParts` é o atributo, ≤ é o operador da expressão e `100` é o valor literal.

A partir da ontologia de alto nível, deve-se derivar a ontologia específica para determinado domínio. Os provedores de serviços podem criar suas próprias políticas derivadas da ontologia específica do domínio, ou simplesmente utilizando um template de política específico para um domínio, preenchendo os detalhes de sua política e obtendo a especificação OWL baseada na política gerada automaticamente. A seguir é apresentada a especificação da política chamada `DeliverPartsToDistributorSupplierA`, do fornecedor chamado `supplierA`, representada em OWL, que corresponde a primeira regra da Figura 25.

```
<xmlns:sp= "http://supplychain.com/policy.owl#"
xmlns:po= "http://samplepolicy.com/policy.owl#"
xmlns= "http://supplierA.com/policy.owl#"
<!-----Policy -
<sp:DeliverPartsToDistributorPolicy
rdf:ID="DeliverPartsToDistributorSupplierA">
  <po:hasRuleSet rdf:ID="RuleSet1">
    <po:hasRuleOperator rdf:resource="po:ExactlyOne"/>
    <po:hasRule>
        <po:Rule rdf:ID="Rule1">
          <hasAction rdf:resource="#DeliverPartsRule1"/>
          <hasCondition rdf:resource="#CheckQuantity1"/>
        </po:Rule>
    </po:hasRule>
  </po:hasRuleSet>
</sp:DeliverPartsToDistributorPolicy>

<sp:CheckQuantity rdf:ID="CheckQuantity1">
  <po:hasExpression>
    <po:Expression rdf:ID="ExpressionCondition1">
       <po:hasVariable>
         <po:Attribute rdf:resource="sp:NumberOfParts"/>
       </po:hasVariable>
     <po:hasExpressionOperator rdf:resource="po:isLessThanOrEqual"/>
     <po:hasLiteralValue>
        <po:LiteralValue rdf:ID="LiteralValue2">
          <po:hasValue
             rdf:datatype="http://www.w3.org/2001/XMLSchema#int">100
          </hasValue>
          <po:hasType rdf:resource="po:Integer"/>
```

```
          </po:LiteralValue>
        </po:hasLiteralValue>
      </po:Expression>
    </po:hasExpression>
  </sp:CheckQuantity>

  <sp:DeliverParts rdf:ID="DeliverPartsRule1">
    <po:associateWithOperation>
       <sp:PurchaseParts rdf:ID="PurchasePartsSupplierA"/>
    </po:associateWithOperation>
    <po:hasExpression rdf:ID="ExpressionAction1">
       <po:hasVariable>
          <po:Attribute rdf:ID="sp:DeliveryDay"/>
       </po:hasVariable>
       <po:hasExpressionOperator rdf:resource="po:isLessThanOrEqual"/>
         <po:hasLiteralValue rdf:ID="LiteralValue1">
           <po:hasValue
rdf:datatype="http://www.w3.org/2001/XMLSchema#int">3
           </po:hasValue>
           <po:hasType rdf:resource="po:Integer"/>
         </po:hasLiteralValue>
     </po:hasExpression>
     <po:requireCondition rdf:resource="#CheckQuantity1"/>
  </sp:DeliverParts>
```

A Figura 27 apresenta a adição da política de regras de negócio chamada `PartsTo-DistributorPolicy` referente à política representada em OWL `DeliverPartsToDistri-butorSupplierA` (apresentada anteriormente) à especificação do arquivo policy.xml de todas as políticas do *web service* do `supplierA`.

```
base = "http://supplierA.com/policy.xml"
wspsp = "http://supplychainschema.com/policyspec"
…
<wsp:Policy Name="PurchasingProcess">
    <wsp:All>
      <wspsp:DeliverPartsToDistributorPolicy>
http://supplierA.com/policy.owl#DeliverPartsToDistributorSupplierA
      </wspsp:DeliverPartsToDistributorPolicy>

<!---Other policy for PurchasingProcess, e.g. WS-Security -→
…
    </wsp:All>
</wsp:Policy>
```

Figura 27 – Adição da Política de regras de negócio ao arquivo Policy.xml [Sriharee *et al.*, 2004].

A Figura 28 apresenta anexação da política aos WSDL do serviço.

```
base = "http://supplierA.com/purchaseparts.wsdl"
…
<wsp:PolicyAttachment>
  <wsp:AppliesTo>
    <wsp:EndpointReference>
      <wsp:ServiceName
            Name="PurchasePartsService"/>
         <wsp:PortType Name="PurchasePartsPortType/>
           <wsp:Address URI="http://supplierA.com/policy.xml" />
     </wsp:EndpointReference>
   </wsp:AppliesTo>
  <wsp:PolicyReference Ref="http://supplierA.com/policy.xml"/>
</wsp:PolicyAttachment>
```

Figura 28 - Anexando a Política ao WSDL do serviço [Sriharee *et al*., 2004].

Ao ter a política no WSDL do serviço publicada no UDDI, é realizada a busca padrão no UDDI, onde se obtém os possíveis serviços que atendem às políticas, depois disso é feita a busca baseada em regras de negócio e a avaliação das políticas é feita com base no *reasoning* de regras. A Figura 29 apresenta o framework proposto por Sriharee *et al*. [2004] que permitirá a descoberta de serviços com base nas políticas de regras de negócio.

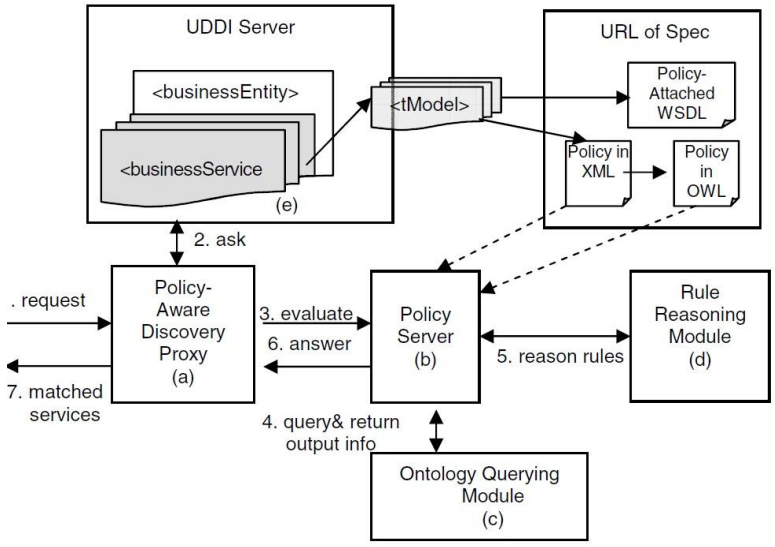

Figura 29 - Framework de descoberta baseada em Políticas [Sriharee *et al*., 2004].

Alguns serviços candidatos que satisfaçam algumas características são obtidos através do UDDI padrão. O PADP (Policy Aware Discovery Proxy) consultará o servidor UDDI para obter os tModels que se referem a arquivos WSDL de todos os serviços candidatos (2). Do WSDL, o PADP pode obter a referência para o arquivo de todas as políticas de cada *web service* e envia a referência do arquivo para o servidor de política (3). Ao receber a referência, o servidor de políticas carrega o arquivo de toda a política correspondente, extrai a referência para o arquivo OWL e recupera o OWL baseado em regras de negócio da política. O Policy Server irá interagir com o módulo para consultar o Ontology Querying Module para fazer o *match* dos conceitos ontológicos da especificação da política da requisição (4), do domínio e da política. O Policy Server ainda analisa a política OWL em uma especificação baseada em regras e contacta o Rule Reasoning Module para avaliar a especificação (5). Finalmente, o Policy Server relata os resultados encontrados para PADP (6) que por sua vez classifica os resultados antes de retornar para o distribuidor (7).

Sriharee *et al*. [2004] apresentam a proposta para descoberta de serviços com base nas políticas que descrevem as regras de negócio do serviço. Esta proposta está relacionada com as questões levantadas na seção 4.1, especialmente relacionada com a questão de encontrar os melhores serviços para um determinado contexto e até mesmo para encontrar os serviços mais adequados para criar uma composição.

3.7 A User Driven Policy Selection Model

Diferentemente de Schulte *et al*. [2009] e Baresi *et al*. [2006], Fugini *et al*. [2006] não abordam a monitoria dos serviços em execução. A proposta de Fugini *et al*. [2006] está na etapa anterior à execução, ou seja, está na escolha do melhor serviço.

O principal objetivo do trabalho de Fugini *et al*. [2006] é o de apresentar um modelo que "traduza" as *policies* para o usuário. Fugini *et al*. [2006] consideram que é necessário haver uma abordagem que permita ao usuário, através de uma linguagem de alto nível, escolher os serviços mais adequados de acordo com seu contexto. Segundo os autores, as propostas encontradas na literatura não endereçam essa diferença entre a perspectiva do usuário e a perspectiva da aplicação. Além disso, eles ressaltam que nem sempre os requisitos de qualidade expressos nas *policies* têm a mesma importância, dependendo do domínio e do usuário.

A proposta de Fugini *et al*. [2006] é dividida em três modelos principais: o modelo de dimensões da qualidade, o modelo de definição da qualidade do *web service* e o modelo de avaliação da qualidade.

3.7.1 Modelo de dimensões da qualidade

Fugini *et al*. [2006] definem dimensão da qualidade como sendo "um aspecto não funcional relacionado a uma entidade que se deseja descrever". A dimensão da qualidade, por sua vez, pode ser dividida entre primitiva e derivada. A dimensão da qualidade primitiva pode ser expressa da seguinte forma:

```
pqd = <nome, valores>
```

O parâmetro `nome` identifica univocamente a dimensão da qualidade primitiva, enquanto `valores` definem os valores que essa dimensão da qualidade pode assumir no domínio. Como exemplo de dimensão de qualidade primitiva, pode-se citar a largura de banda de uma rede. Já os valores podem ser, por exemplo, de 0 a 100, quando se diz que são contínuos, ou então "banda larga", quando se diz que o valor é discreto [Fugini *et al*., 2006].

Da dimensão da qualidade primitiva (*pqd*) obtém-se a dimensão da qualidade derivada (*dqd*). A dimensão da qualidade derivada pode ser expressa da seguinte maneira [Fugini *et al*., 2006]:

```
dqd =< nome,f(pqdi, dqdj )> i= 0..n, j = 0..m
```

Assim como *pqd*, a *dqd* é identificada por um nome. Entretanto, seu valor é calculado em função de uma ou mais *pqd´*s. Além disso, outras *dqd´*s também podem influenciar no cálculo do valor da *dqd*. Essa é a dimensão da qualidade que o usuário irá avaliar.

3.7.2 Modelo de definição da qualidade do *web service*

O modelo de definição da qualidade do *web service* nada mais é do que uma árvore, denominada árvore de qualidade [Fugini *et al*., 2006].

Devido à forte dependência do domínio, este modelo precisa ser desenvolvido por um *expert* no domínio. É esse profissional que irá compor a árvore de qualidade, fazendo uso das dimensões de qualidade primitivas e derivadas. Também é esse profissional que será capaz de saber quando uma dimensão de qualidade é primitiva ou derivada.

A Figura 30 apresenta um exemplo de uma árvore de dimensões da qualidade. Segundo Fugini *et al*. [2006], esta árvore é utilizada tanto pelos consumidores quanto pelos provedores dos serviços para descrever a qualidade.

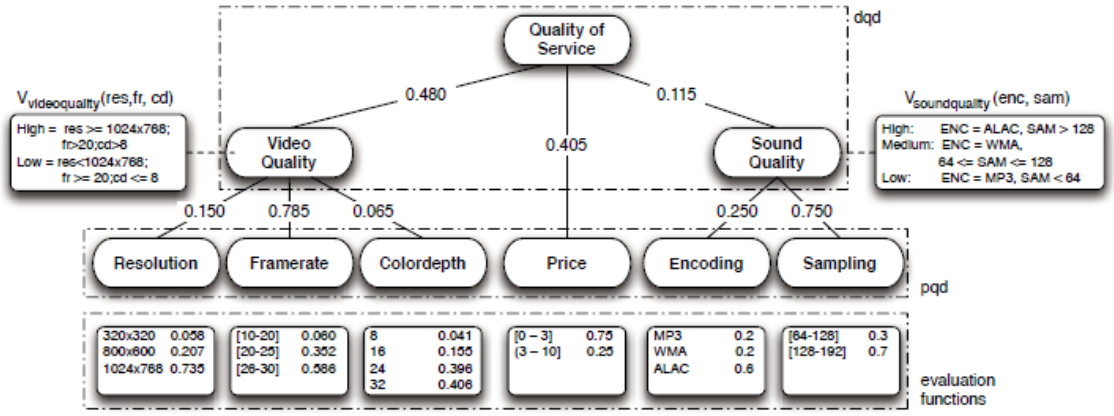

Figura 30 – Árvore de dimensões da qualidade [Fugini *et al*., 2006]

Na Figura 30 é possível observar que a qualidade do serviço (*quality of service*) é uma dimensão da qualidade derivada que depende de outras duas dimensões: qualidade do vídeo (*video quality*) e qualidade do som (*sound quality*). A qualidade do vídeo e a qualidade do som, por sua vez, também são dimensões de qualidade derivadas. A qualidade do vídeo depende da resolução (*resolution*), do *framerate* e da paleta de cores (*colordepth*). A qualidade do som depende da codificação (*encoding*) e do *sampling*. Todas essas dimensões da qualidade (resolução, *framerate*, paleta de cores, codificação e *sampling*) são do tipo primitiva.

Além de calcular as dimensões derivadas da qualidade, pode acontecer de algumas dimensões terem mais ou menos importância do que as demais. Desta forma, faz parte da elaboração da árvore a atribuição de pesos para cada dimensão da qualidade derivada. Para isso Fugini *et al*. [2006] determinam o uso da abordagem AHP (*Analitic Hierarchy Process*). Os valores representados na Figura 30 já estão ajustados.

3.7.3 Modelo de avaliação da qualidade

Uma função de avaliação precisa ser criada, pois nem sempre valores mais altos representam alta qualidade. No caso da largura de banda, por exemplo, quanto mais alto o valor, mais qualidade. Entretanto, no caso da latência, quanto mais alto o valor, pior é a qualidade [Fugini *et al*., 2006].

A dimensão de qualidade primitiva possui uma função relativamente simples, que basicamente leva em consideração o ajuste proposto pela abordagem AHP. Já a dimensão de qualidade derivada precisa ser calculada em função das dimensões de qualidade primitivas e de outras dimensões de qualidade derivadas, além do fator de ajuste [Fugini *et al*., 2006].

Na Figura 30 é possível observar que a qualidade do serviço está diretamente ligada a duas dimensões de qualidade derivadas (*video quality* e *sound quality*) e uma única dimensão da qualidade primitiva (*price*). Desta forma, a qualidade do serviço é o somatório dessas três dimensões (*video quality*, *sound quality* e *price*). Cada uma dessas dimensões, entretanto, precisa ser calculada. A dimensão *price*, por ser primitiva, é facilmente calculada, bastando que se leve em consideração o fator de ajuste. Já as dimensões *video quality* e *sound quality* são derivadas. Sendo assim, primeiro é necessário definir um fator de ajuste para cada dimensão da qualidade primitiva associada às dimensões de qualidade derivadas. Depois é feito um somatório das dimensões de qualidade primitivas (devidamente ajustadas) de cada dimensão de qualidade derivada. Em seguida, é feito o ajuste final, de acordo com o fator de ajuste da dimensão da qualidade derivada [Fugini *et al*., 2006].

3.7.4 Modelo de política

Fugini *et al*. [2006] descrevem que uma política pode ser definida com base em elementos da árvore de qualidade. Uma política precisa conter alternativas, sendo que cada alternativa contém assertivas. Essas assertivas nada mais são do que especializações das dimensões de qualidade [Fugini *et al*., 2006].

Dessa forma, o provedor do serviço pode ter seu documento de política do serviço. Já o usuário pode ter o documento de política do usuário e também a árvore de qualidade do usuário. Essa árvore de qualidade do usuário nada mais é do que uma árvore de qualidade customizada pelo usuário [Fugini *et al*., 2006].

3.7.5 Modelo de seleção de política

Definidos os modelos, chega o momento de se definir como as políticas são selecionadas.

Segundo Fugini *et al*. [2006], o processo de seleção recebe como insumos a política do serviço, a política do usuário e a árvore de qualidade do usuário. A partir daí, duas atividades são executadas: 1) avaliação da satisfabilidade; 2) *ranking* de alternativas. Na Figura 31 é possível observar um exemplo do modelo.

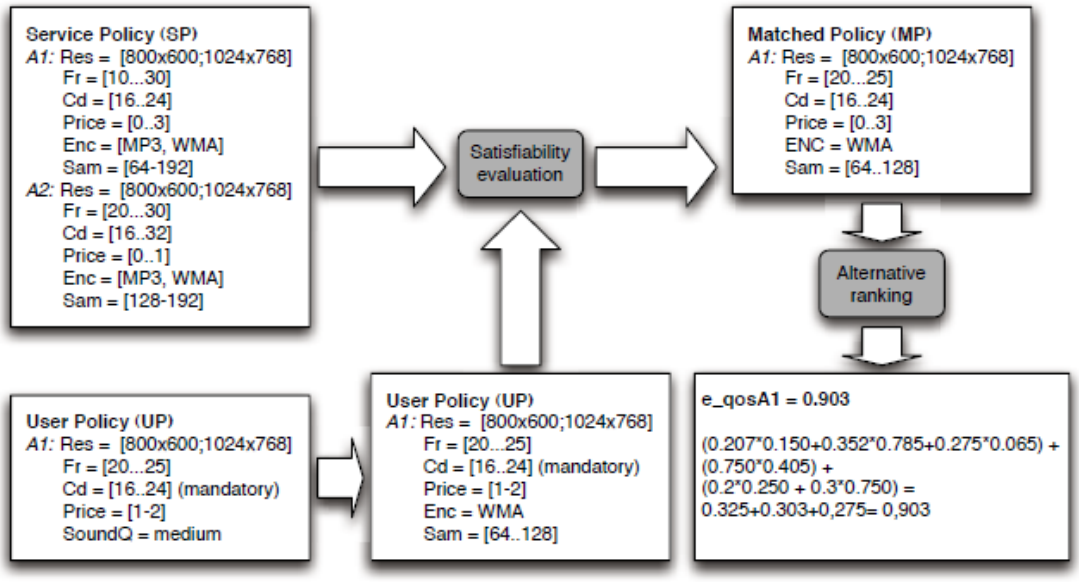

Figura 31 – Exemplo de avaliação da qualidade [Fugini *et al*., 2006]

Durante a avaliação da satisfabilidade procura-se encontrar políticas de serviços que ofereçam todas as dimensões incluídas na política do usuário. Daí resulta a política chamada de *matched policy* [Fugini *et al*., 2006].

Em seguida, é feito o *ranking* das alternativas incluídas na *matched policy*. Essa etapa é crucial para que as alternativas sejam avaliadas de acordo com as preferências do usuário. O resultado desta atividade é o *ranked policy*.

Diferentemente das demais propostas, Fugini *et al*. [2006] endereçam a facilitação do usuário na escolha de políticas. Entretanto, os autores não apresentam uma implementação da proposta.

4 Conclusão

Os trabalhos apresentados podem ser divididos em dois grupos: um primeiro grupo que trata a descoberta de serviços e um outro grupo para monitoramento de serviços. Em ambos os grupos a questão das políticas é relacionada com requisitos não funcionais.

Um dos principais diferenciais entre esses dois grupos é que o primeiro faz uso das políticas para identificar o serviço mais adequado antes de sua execução. Já o segundo grupo endereça o uso de políticas para identificar se o serviço, já em execução, continua a atender as políticas acordadas.

Na descoberta de serviços, a proposta da virtualização de serviços e recursos permite que o serviço mais adequado, de acordo com a política especificada, seja selecionado. Na descoberta de serviços com base em políticas de QoS enriquecidas com semântica foi proposta a especificação da política juntamente com a ontologia, sendo que é esta ontologia que fornece semântica às políticas, permitindo que políticas relacionadas ou semelhantes sejam selecionadas, mesmo que não sejam idênticas. Também foi proposta uma especificação de políticas utilizando-se ontologias com foco em regras de negócio. Por fim foi apresentada uma proposta na qual os próprios usuários, com base em sua árvore de políticas, faça a escolha dos serviços de forma transparente.

Em relação ao monitoramento de serviços, foi proposto um monitor que acompanha a qualidade do serviço. Foi abordado o balanceamento entre desempenho e monitoriamento, sendo possível que o consumidor, a qualquer instante, possa aumenta ou diminui a carga de monitoramento, equilibrando monitoria e desempenho. Também foi endereçado o problema de o que fazer quando um serviço não mais atende às políticas acordadas.

Referências

ALONSO, G. CASATI, F. KUNO, H. MACHIRAJU, V. **Web Services Concepts, Architecture and Applications**. ISBN 3-540-44008-9. Springer, 2004. 328 p.

BARESI, L. GUINEA, S. PLEBANI, P. **WS-Policy for Service Monitoring**. In: 6th VLDB Intl. Workshop on Technologies for E-Services, volume 3811 of Lect. Notes in Computer Science, 2006.

BOOTH, D., HAAS, H., McCABE, F., NEWCOMER, E., CHAMPION, M., FERRIS, C., ORCHARD, D., 2004, **Web Services Architecture**, W3C Working Group Note 11 February 2004. Disponível em < http://www.w3.org/TR/ws-arch/#engaging>. Acesso em: 10 jun. 2008.

BOX, D et al. **Web Services Policy Framework (WS-POLICY)**. BEA, IBM, Microsoft, SAP, Sonic, VeriSign, Setembro 2004. Disponível em: ttp://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf. Acesso em 8 de jun 2010.

CHAPPELL, D. **Introducing SCA**. Chappell & Associates., 2007. Disponível em <http://www.davidchappell.com/articles/introducing_sca.pdf>. Acessado em 16 Jun. 2010.

FUGINI, M. PLEBANI, P. RAMONI, F. **A User Driven Policy Selection Model**. Proceedings of the Service Oriented Computing – ISOC, 2006.

HEWITT, E., **Java SOA Cookbook**, O'Reilly, 2009.

GARCIA, D. TOLEDO, B. **Semantics-enriched QoS Policies for Web Service Interactions**. WebMedia'06. Novembro 19-22. Natal, RN, Brasil. 2006.

HEINZL, S., SCHMELING, B. **Using XML Schema to Improve Writing, Validation, and Structure of WS-Policies** Anais do ACM Symposium On Applied Computing. ISBN: 978-1-605580639. 2010.

IBM. ABLE Rule Language User's Guide and Reference 2.3.0, 2005. Disponível em < http://www.research.ibm.com/able/doc/reference/com/ibm/able/rules/docfiles/ABLERuleLanguage.pdf>. Acessado em 24 jan. 2011.

HUNG, Patrick C. K. **From Conflict of Interest to Separation of Duties in WS-Policy for Web Services Matchmaking Process**. Proceedings of the 37th Hawaii International Conference on System Sciences, 2004.

MIETZNER, R. LESSEN, T. WIESE, A. WIELAND, M. KARASTOYANOVA, D. LEYMANN, F. **Virtualizing Services and Resources with ProBus: The WS-Policy-Aware Service and Resource Bus**. IEEE International Conference on Web Services. 2009.

PATEL-SCHNEIDER, P. F., HAYES, P., HORROCKS, I. OWL Web Ontology Language Semantics and Abstract Syntax, 2004. Disponível em <w3.org/TR/owl-semantics/>. Acessado em 24 jan. 2011.

PAVLIK, G. HYNES, D e GRALL, T. **Infra-estrutura SOA de Próxima Geração**. 2007. Disponível em: http://www.scribd.com/doc/4005278/-ref-Infraestrutura-SOA-de-Proxima-Geracao.

SCHULTE et al. **From Web Service Policies to Automatic Deviation Handling - Supporting semantic description of reactions to policy violations**. IEEE International Conference on Semantic Computing. 2009.

SOUSA, H.P., AZEVEDO, L., SANTORO, F., BAIAO, F. **Estudos de ESB e OSB**. Relatórios Técnicos do NP2TEC/UNIRIO, RT-0007/2010, 2010.

SRIHAREE, N. SENIVONGSE, T. VERMA, K. SHETH, A. **On Using WS-Policy, Ontology, and Rule Reasoning to Discover Web Services**. In: Proceedings of INTELLCOMM, 2004.

UDDI, **Universal Description, Discovery & Integration Specification version 3, 2004. Disponível em <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3>. Acessado em 25 Nov. 2009.**

SAWSDL, **Semantic Annotations for WSDL and XML Schema, 2007.** Disponívem em: http://www.w3.org/TR/sawsdl/. Acesso em 26/04/2011.

WS-Policy. Web Services Policy 1.5 - Primer. W3C Standard, 2006. Disponível em: http://www.w3.org/TR/2006/WD-ws-policy-primer-20061221/. Acesso em 15/06/2010.