5
Aplicações

Neste capítulo são apresentadas duas aplicações do *framework*. A primeira aplicação compreende o contexto de deslizamentos de massas e o segundo está no domínio de agência de viagens.

5.1.
Escorregamento de Massas

O Brasil, por sua diversidade de condições climáticas está suscetível a desastres naturais quando se associam estas condições climáticas à ocupação irregular do terreno e a solos instáveis como ocorre nas encostas. Este conjunto de fatores pode levar ao que Souza *et al* descreve como escorregamentos de massas e os define como "movimentos rápidos, de porções de terreno (solos e rochas), com volumes definidos, deslocando-se sob ação da gravidade, para baixo e para fora do talude ou da vertente" (SOUZA, FARIA, *et al.*, 2009).

No Estado do Rio de Janeiro, nota-se a ocorrência frequente de escorregamentos de massas devido aos fatores naturais como relevo acidentado e constantes precipitações. Tais condições estimularam o governo do estado a investir no monitoramento das áreas propensas a suceder estes escorregamentos através de incentivos a pesquisa e desenvolvimento de sistemas de software capazes de analisar a probabilidade de ocorrência de deslizamentos de terra. Neste contexto, foi desenvolvida a aplicação GeoRisc, um sistema multiagentes capaz de calcular o grau de susceptibilidade de um terreno e seu risco geoambiental, em que o risco pode ser representado por um modelo computacional que inclui características e modelos comportamentais do conjunto de elementos e variáveis que compõem o ambiente real (CERQUEIRA, SANTOS, *et al.*, 2009).

Neste contexto, e no domínio de aplicação do GeoRisc, foi desenvolvida uma nova aplicação estendendo o *framework* elaborado neste trabalho de modo a demonstrar a adaptabilidade do sistema na busca de serviço que forneça o cálculo do risco de escorregamento de massa em uma determinada área.

5.1.1. Ideia Principal

O sistema multiagentes mencionado anteriormente pode receber diversas informações para o cálculo da suscetibilidade de uma área de risco dentre as quais se podem citar: tipo de vegetação, solo, declividade e pluviometria. De posse de algumas destas informações o agente *System* enviará os dados ao Agente Cálculo Risco (ACR) que encontrará um serviço web capaz de realizar o cálculo de risco da determinada área. O cálculo do risco de escorregamento de massas foi disponibilizado em um serviço web, através de combinação qualitativa que utiliza as informações de declividade e vegetação.

Quando o ACR receber as informações de declividade e vegetação ele tentará utilizar o serviço web atual para cálculo do risco de escorregamento de massas da determinada área. Caso o serviço web não esteja disponível, seja por indisponibilidade do serviço web, tempo demasiadamente longo de resposta ou falha de comunicação, o ACR pode adaptar seu comportamento procurando um novo serviço web que forneça as informações procuradas, no caso o cálculo do risco.

Para superar as dificuldades citadas acima, os ACR possuem as seguintes bases de conhecimentos para realizar as adaptações: uma base de regras a qual é possível detectar problemas a partir dos dados coletados no monitoramento dos serviços web e das mensagens trocadas entre o sistema e serviço web; e um conjunto de descrições DF que descrevem os serviços web atualmente ativos. Cada ACR possui um *control loop,* provido pelo *framework,* responsável por realizar a adaptação do sistema, descrito em maiores detalhes a seguir.

5.1.2. Agente Cálculo Risco

Na atividade *Monitor* (Ver Seção 4.1.1) foram utilizados os monitoramentos de serviço web disponibilizado pela classe *ServiceMonitor* e de mensagens disponibilizado pela classe *MessageMonitor*. O primeiro monitora o funcionamento dos serviços web obtendo informações sobre a execução do mesmo e o segundo monitora a troca de mensagens entre o sistema e o serviço web. Também foram criadas as classes *RiskTranslatorSM* estendida da classe

ServiceMonitor e *RiskTranslatorMM* estendida da classe *MessageMonitor*, que são responsáveis por: implementar o método *translator*, que estrutura os dados coletados a partir do monitoramento do serviço web; e implementar o método *translator*, que estrutura os dados coletados a partir da troca de mensagens entre o sistema e o serviço web, respectivamente, de modo que possam ser entendidos pela classe *Analyze*. A Figura 5.1 apresenta a classe *Monitor* do ACR.

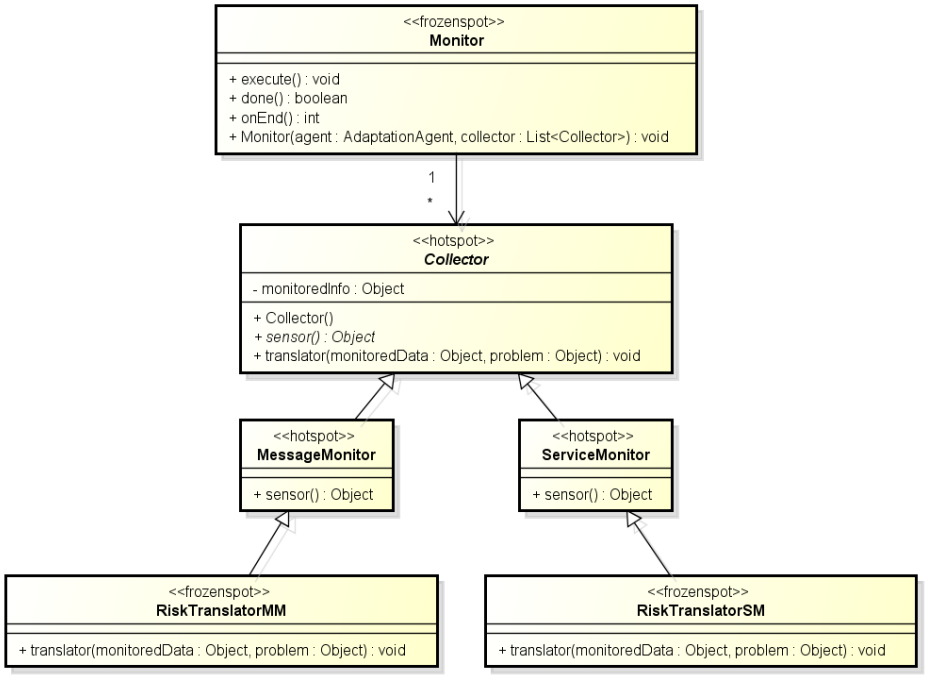

Figura 5.1 – Atividade *Monitor* de um ACR.

A atividade *Analyze* inicia sua execução recebendo o problema captado pela atividade *Monitor*. A partir do RBR (Ver Seção 4.1.2.1) tentará identificar o problema comparando com a base de regras definidas no método *importBase* da classe *RiskBase*, como pode ser observado na Figura 5.2. Para esta aplicação, tem-se definida uma base de regras contendo três regras, descritas a seguir:

Regra 1 : If "*serviceUnavailable*" Then problem = "*serviceUnachievable*"

Regra 2 : If *responseTime* > 30 Then problem = "*timeout*"

Regra 3 : If *serviceFailure > 0* Then problem = "*serviceFail*"

A Regra 1 detecta a indisponibilidade do serviço, onde o parâmetro *serviceUnavailable* é fornecido pelo serviço web informando sua indisponibilidade. Já a Regra 2 verifica se o tempo de resposta do serviço web é excessivamente longo e neste caso temos o problema do *timeout*. Por fim temos a

Regra 3 onde a variável *serviceFailure* representa a falha de comunicação com o serviço web sempre que sua quantidade for maior que zero.

O RBR é formado por um par "problema-solução", portanto, se nenhum problema for detectado a partir da aplicação das regras citadas anteriormente, então nenhuma adaptação é necessária e o sistema continuará sua execução. Se houver algum problema, e este se encaixa com alguma das regras supracitadas, então já está estabelecida a solução, neste caso um serviço substituto. Caso contrário, a atividade *Analyze* continuará sua execução, agora elaborando uma lista de serviços candidatos através da classe *SurrogaServices*. Nesta classe, o método *execute* é responsável por percorrer o registro DF a procura de serviços web que ofereçam o serviço procurado, ou seja, um serviço que faça cálculos de risco para escorregamento de massas. O próximo passo é repassar à atividade *Plan*, o problema atual e a lista de serviços web candidatos a substitutos.

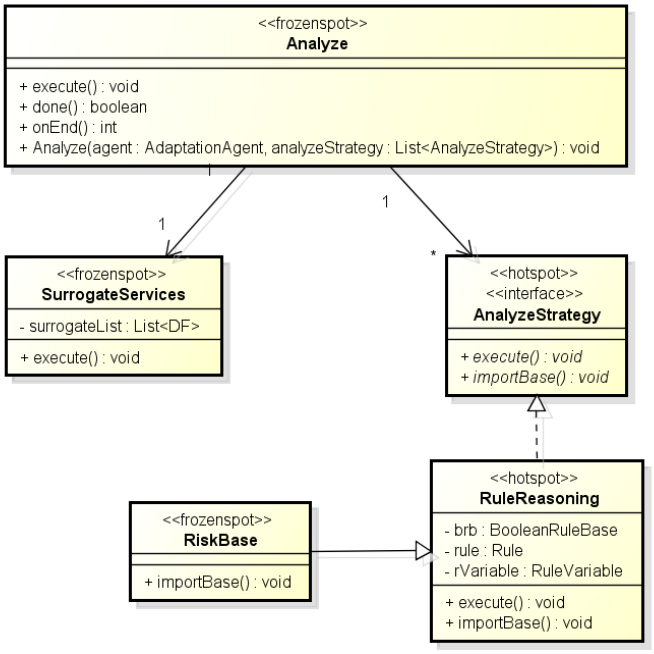

Figura 5.2 – Atividade *Analyze* de um ACR.

Para melhor entendimento do funcionamento do RBR observe os casos a seguir, em que o problema atual é coincidente com uma das regras e neste caso já se tem uma solução pré-estabelecida.

Caso 1

Descrição da Área de Risco:

Declividade: 0,25 porcento

Vegetação: 0,60 porcento

Tipo de Serviço: Cálculo de Risco

Problema: "*serviceUnachievable*"

Solução do Caso 1

Descrição do Serviço: A solução para o Caso 1 é descrita no WSDL da Figura 5.3, que oferece o serviço de cálculo de risco com as seguintes características:

Tipo de Serviço: Cálculo de Risco

Entradas: Parâmetros do tipo *double* que representam os atributos Declividade e Vegetação;

Saída: O cálculo do risco é representeado por um valor do tipo *double*.

Caso 2

Descrição da Área de Risco:

Declividade: 0,25 porcento

Vegetação: 0,60 porcento

Tipo de Serviço: Cálculo de Risco

Problema: "serviceFail"

Solução do Caso 2

Descrição do Serviço: A solução para o Caso 2 é descrita no WSDL da Figura 5.4, que oferece o serviço de cálculo de risco com as seguintes características:

Tipo de Serviço: Cálculo de Risco

Entradas: Parâmetros do tipo *double* que representam os atributos Declividade e Vegetação;

Saída: O cálculo do risco é representeado por um valor do tipo *double*.

Deste modo, ao se identificar o problema "*serviceUnachievable"* o serviço web que será executado é o serviço representado pelo WSDL da Figura 5.3, já no caso de uma falha do tipo "*serviceFail*" o serviço web que será executado está representado pelo WSDL da Figura 5.4. Observe que, para falhas conhecidas têm-se serviços substitutos que atendem plenamente a necessidade do sistema, o que pode não ocorre quando se escolhe um novo serviço web a partir de uma lista de serviços web candidatos elaborada em tempo de execução. Como meio de obter o serviço web que melhor atende as necessidades do sistema, foi elaborado um mecanismo de similaridade, incorporado à atividade *Plan*, de modo a identificar o serviço web, dentre a lista de candidatos, mais similar ao serviço atual.

```
- <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:ns1="http://org.apache.axis2/xsd"
        xmlns:ns="http://servico.georisc.les.puc.rio.br" xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
        targetNamespace="http://servico.georisc.les.puc.rio.br">
        <wsdl:documentation>Calculo de Risco</wsdl:documentation>
- <wsdl:types>
- <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
        targetNamespace="http://servico.georisc.les.puc.rio.br">
- <xs:element name="getRisco">
- <xs:complexType>
- <xs:sequence>
<xs:element minOccurs="0" name="declividade" type="xs:double" />
<xs:element minOccurs="0" name="vegetacao" type="xs:double" />
    </xs:sequence>
    </xs:complexType>
    </xs:element>
- <xs:element name="getRiscoResponse">
- <xs:complexType>
- <xs:sequence>
<xs:element minOccurs="0" name="return" type="xs:double" />
    </xs:sequence>
    </xs:complexType>
    </xs:element>
    </xs:schema>
    </wsdl:types>
- <wsdl:message name="getRiscoRequest">
<wsdl:part name="parameters" element="ns:getRisco" />
    </wsdl:message>
- <wsdl:message name="getRiscoResponse">
<wsdl:part name="parameters" element="ns:getRiscoResponse" />
    </wsdl:message>
- <wsdl:portType name="CalculoRiscoPortType">
- <wsdl:operation name="getRisco">
<wsdl:input message="ns:getRiscoRequest" wsaw:Action="urn:getRisco" />
<wsdl:output message="ns:getRiscoResponse" wsaw:Action="urn:getRiscoResponse" />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="CalculoRiscoSoap11Binding" type="ns:CalculoRiscoPortType">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <wsdl:operation name="getRisco">
<soap:operation soapAction="urn:getRisco" style="document" />
- <wsdl:input>
<soap:body use="literal" />
    </wsdl:input>
- <wsdl:output>
<soap:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:binding name="CalculoRiscoSoap12Binding" type="ns:CalculoRiscoPortType">
<soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <wsdl:operation name="getRisco">
<soap12:operation soapAction="urn:getRisco" style="document" />
- <wsdl:input>
<soap12:body use="literal" />
    </wsdl:input>
- <wsdl:output>
<soap12:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:binding name="CalculoRiscoHttpBinding" type="ns:CalculoRiscoPortType">
<http:binding verb="POST" />
- <wsdl:operation name="getRisco">
<http:operation location="CalculoRisco/getRisco" />
- <wsdl:input>
<mime:content type="text/xml" part="getRisco" />
    </wsdl:input>
- <wsdl:output>
<mime:content type="text/xml" part="getRisco" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:service name="CalculoRisco">
- <wsdl:port name="CalculoRiscoHttpSoap11Endpoint" binding="ns:CalculoRiscoSoap11Binding">
<soap:address location="http://localhost:8080/GeoRiscMobileWS/services/CalculoRisco.CalculoRiscoHttpSoap11Endpoint/" />
    </wsdl:port>
- <wsdl:port name="CalculoRiscoHttpSoap12Endpoint" binding="ns:CalculoRiscoSoap12Binding">
<soap12:address location="http://localhost:8080/GeoRiscMobileWS/services/CalculoRisco.CalculoRiscoHttpSoap12Endpoint/"
        />
    </wsdl:port>
- <wsdl:port name="CalculoRiscoHttpEndpoint" binding="ns:CalculoRiscoHttpBinding">
<http:address location="http://localhost:8080/GeoRiscMobileWS/services/CalculoRisco.CalculoRiscoHttpEndpoint/" />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

Figura 5.3 – WSDL da Solução do Caso 1.

```
- <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:ns1="http://org.apache.axis2/xsd"
        xmlns:ns="http://servico.georisc.les.puc.rio.br"
        xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
        targetNamespace="http://servico.georisc.les.puc.rio.br">
        <wsdl:documentation>Calculo de Risco</wsdl:documentation>
- <wsdl:types>
- <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
        targetNamespace="http://servico.georisc.les.puc.rio.br">
- <xs:element name="calculaRisco">
- <xs:complexType>
- <xs:sequence>
<xs:element minOccurs="0" name="declividade" type="xs:double" />
<xs:element minOccurs="0" name="vegetacao" type="xs:double" />
    </xs:sequence>
    </xs:complexType>
    </xs:element>
- <xs:element name="calculaRiscoResponse">
- <xs:complexType>
- <xs:sequence>
<xs:element minOccurs="0" name="return" type="xs:double" />
    </xs:sequence>
    </xs:complexType>
    </xs:element>
    </xs:schema>
    </wsdl:types>
- <wsdl:message name="calculaRiscoRequest">
<wsdl:part name="parameters" element="ns:getRisco" />
    </wsdl:message>
- <wsdl:message name="calculaRiscoResponse">
<wsdl:part name="parameters" element="ns: calculaRiscoResponse" />
    </wsdl:message>
- <wsdl:portType name="CalculoRiscoPortType">
- <wsdl:operation name="calculaRisco">
<wsdl:input message="ns: calculaRiscoRequest" wsaw:Action="urn: calculaRisco" />
<wsdl:output message="ns: calculaRiscoResponse" wsaw:Action="urn: calculaRiscoResponse" />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="CalculoRiscoSoap11Binding" type="ns:CalculoRiscoPortType">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <wsdl:operation name="calculaRisco">
<soap:operation soapAction="urn: calculaRisco" style="document" />
- <wsdl:input>
<soap:body use="literal" />
    </wsdl:input>
- <wsdl:output>
<soap:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:binding name="CalculoRiscoSoap12Binding" type="ns:CalculoRiscoPortType">
<soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <wsdl:operation name="calculaRisco">
<soap12:operation soapAction="urn: calculaRisco" style="document" />
- <wsdl:input>
<soap12:body use="literal" />
    </wsdl:input>
- <wsdl:output>
<soap12:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:binding name="CalculoRiscoHttpBinding" type="ns:CalculoRiscoPortType">
<http:binding verb="POST" />
- <wsdl:operation name="calculaRisco">
<http:operation location="CalculoRisco/calculaRisco" />
- <wsdl:input>
<mime:content type="text/xml" part="calculaRisco" />
    </wsdl:input>
- <wsdl:output>
<mime:content type="text/xml" part="calculaRisco" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:service name="CalculoRisco">
- <wsdl:port name="CalculoRiscoHttpSoap11Endpoint" binding="ns:CalculoRiscoSoap11Binding">
<soap:address
        location="http://localhost:8080/GeoRiscMobileWS2/services/CalculoRisco.CalculoRiscoHttpSoap11Endpoint
        /" />
    </wsdl:port>
- <wsdl:port name="CalculoRiscoHttpSoap12Endpoint" binding="ns:CalculoRiscoSoap12Binding">
<soap12:address
        location="http://localhost:8080/GeoRiscMobileWS2/services/CalculoRisco.CalculoRiscoHttpSoap12Endpoint
        /" />
    </wsdl:port>
- <wsdl:port name="CalculoRiscoHttpEndpoint" binding="ns:CalculoRiscoHttpBinding">
<http:address location="http://localhost:8080/GeoRiscMobileWS2/services/CalculoRisco.CalculoRiscoHttpEndpoint/"
        />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

Figura 5.4 – WSDL da Solução do Caso 2.

Já na atividade *Plan*, representada na Figura 5.5, foram utilizadas as classes base de modo a aproveitar o algoritmo de similaridade elaborado por Resnik (Ver Seção 4.1.3.1) já incorporado ao *framework*.

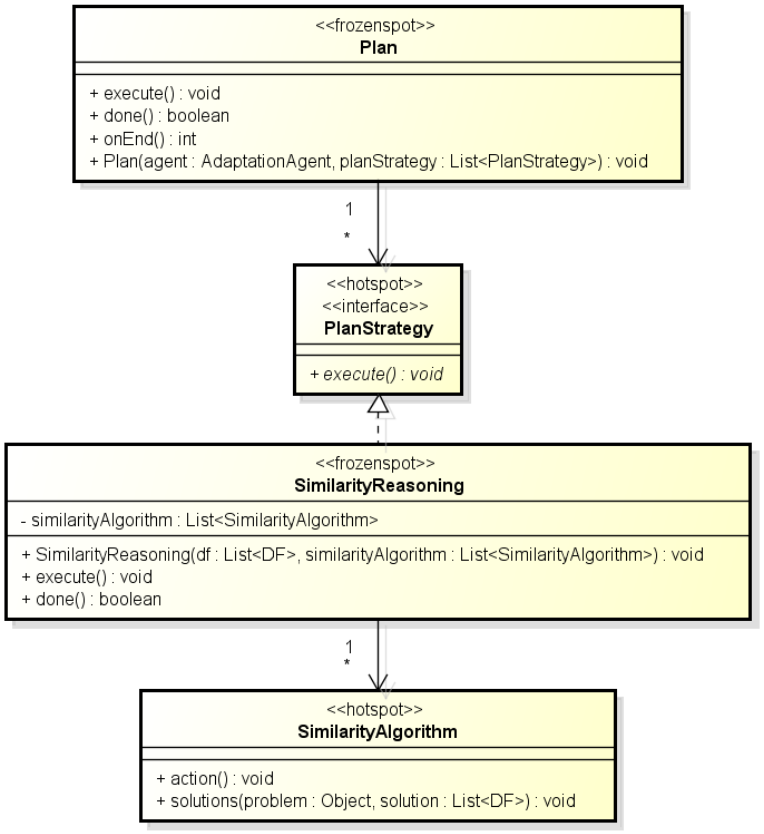

Figura 5.5 – Atividade *Plan* de um ACR.

Considerando que a solução não foi encontrada na atividade *Analyze*, através do RBR, então a tarefa de identificar um novo serviço web cabe à atividade *Plan*, que recebe a lista de serviços candidatos e o problema atual. O método *execute* da classe *SimilarityReasoning* será executado chamando o método *solutions* para o cálculo da similaridade.

Note que na seleção do novo serviço web, este pode utilizar um protocolo de dados, comunicação e descrição diferente do padrão FIPA utilizado pelos agentes. Caso este fato se confirme, então será necessária a transcrição das mensagens do padrão dos agentes (FIPA) para o padrão de serviços web (W3C). Esta atividade de transcrição se configura na atividade *Execute*, detalhada mais a frente. Por hora, observe os casos a seguir exemplificando o funcionamento do mecanismo de similaridade, em que a Solução 1 é está representada na Figura 5.6 e a Solução 2 na Figura 5.7.

Problema Atual

Serviços: O serviço solicitado deve conter os parâmetros: nome do serviço (Escorregamento de Massa) e tipo de serviço (Cálculo de Risco);

Ontologia: Calcula Risco;

Linguagem: FIPA ACL;

Conteúdo:

Entrada: As informações repassadas ao serviço web são: valor da declividade e vegetação;

Saída: O retorno esperado é grau do risco (representado por um valor).

Solução 1

Serviço: O serviço contêm os parâmetros: nome do serviço (Deslizamento de Massa) e tipo de serviço (Cálculo de Risco);

Ontologia: Calcula o Risco;

Linguagem: FIPA ACL

Conteúdo:

Entrada: As informações solicitadas pelo serviço web são: valor da declividade, vegetação e pluviometria;

Saída: O retorno esperado é grau do risco (representado por um valor).

Solução 2

Serviço: O serviço contêm os parâmetros: nome do serviço (Deslizamento de Massa) e tipo de serviço (Cálculo de Risco);

Ontologia: Calcula Risco;

Linguagem: W3C SOAP

Conteúdo:

Entrada: As informações solicitadas pelo serviço web são: valor da declividade e vegetação;

Saída: O retorno esperado é grau do risco (representado por um valor).

```
(inform
 :sender
  (agent-identifier
   :name df@125.125.125.103:1099
   :addresses (sequence http://service:7778/acc))
 :receiver (set
  (agent-identifier
   :name riskAgent1@125.125.125.103:1099
   :addresses (sequence http://service:7778/acc)))
 :language FIPA-SL0
 :protocol FIPA-Request
 :ontology FIPA-Agent-Management
 :content
  (done
   (action
    (agent-identifier
     :name df@125.125.125.103:1099
     :addresses (sequence http://service:7778/acc))
    (register
     (df-agent-description
      :name
       (agent-identifier
        :name riskAgent1@125.125.125.103:1099
        :addresses (sequence http://service:7778/acc))
      :protocol (set FIPA-Request Application-Protocol)
      :ontology (set Calcula o Risco)
      :language (set FIPA ACL)
      :services (set
       (service-description
        :name Deslizamento de Massa
        :type Cálculo de Risco
        :ontology Calcula o Risco
        :properties (set
        (property
          :name Declividade
          :value double)
        (property
          :name Vegetacao
          :value double)
        (property
          :name Pluviometria
          :value double)))))))))
```

Figura 5.6 – Ilustração do descritor do serviço prestado pela Solução 1.

```
- <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:ns1="http://org.apache.axis2/xsd"
        xmlns:ns="http://servico.georisc.les.puc.rio.br" xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
        targetNamespace="http://servico.georisc.les.puc.rio.br">
        <wsdl:documentation>Calculo Risco</wsdl:documentation>
- <wsdl:types>
- <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
        targetNamespace="http://servico.georisc.les.puc.rio.br">
- <xs:element name="calculaRisco">
- <xs:complexType>
- <xs:sequence>
<xs:element minOccurs="0" name="declividade" type="xs:double" />
<xs:element minOccurs="0" name="vegetacao" type="xs:double" />
    </xs:sequence>
    </xs:complexType>
    </xs:element>
- <xs:element name="calculaRiscoResponse">
- <xs:complexType>
- <xs:sequence>
<xs:element minOccurs="0" name="return" type="xs:double" />
    </xs:sequence>
    </xs:complexType>
    </xs:element>
    </xs:schema>
    </wsdl:types>
- <wsdl:message name="calculaRiscoRequest">
<wsdl:part name="parameters" element="ns:getRisco" />
    </wsdl:message>
- <wsdl:message name="calculaRiscoResponse">
<wsdl:part name="parameters" element="ns: calculaRiscoResponse" />
    </wsdl:message>
- <wsdl:portType name="CalculoRiscoPortType">
- <wsdl:operation name="calculaRisco">
<wsdl:input message="ns: calculaRiscoRequest" wsaw:Action="urn: calculaRisco" />
<wsdl:output message="ns: calculaRiscoResponse" wsaw:Action="urn: calculaRiscoResponse" />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="CalculoRiscoSoap11Binding" type="ns:CalculoRiscoPortType">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <wsdl:operation name="calculaRisco">
<soap:operation soapAction="urn: calculaRisco" style="document" />
- <wsdl:input>
<soap:body use="literal" />
    </wsdl:input>
- <wsdl:output>
<soap:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:binding name="CalculoRiscoSoap12Binding" type="ns:CalculoRiscoPortType">
<soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <wsdl:operation name="calculaRisco">
<soap12:operation soapAction="urn: calculaRisco" style="document" />
- <wsdl:input>
<soap12:body use="literal" />
    </wsdl:input>
- <wsdl:output>
<soap12:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:binding name="CalculoRiscoHttpBinding" type="ns:CalculoRiscoPortType">
<http:binding verb="POST" />
- <wsdl:operation name="calculaRisco">
<http:operation location="CalculoRisco/calculaRisco" />
- <wsdl:input>
<mime:content type="text/xml" part="calculaRisco" />
    </wsdl:input>
- <wsdl:output>
<mime:content type="text/xml" part="calculaRisco" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:service name="DeslizamentodeMassa">
- <wsdl:port name="CalculoRiscoHttpSoap11Endpoint" binding="ns:CalculoRiscoSoap11Binding">
<soap:address location="http://localhost:8080/GeoRiscMobileWS3/services/CalculoRisco.CalculoRiscoHttpSoap11Endpoint/"
        />
    </wsdl:port>
- <wsdl:port name="CalculoRiscoHttpSoap12Endpoint" binding="ns:CalculoRiscoSoap12Binding">
<soap12:address
        location="http://localhost:8080/GeoRiscMobileWS3/services/CalculoRisco.CalculoRiscoHttpSoap12Endpoint/" />
    </wsdl:port>
- <wsdl:port name="CalculoRiscoHttpEndpoint" binding="ns:CalculoRiscoHttpBinding">
<http:address location="http://localhost:8080/GeoRiscMobileWS3/services/CalculoRisco.CalculoRiscoHttpEndpoint/" />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

Figura 5.7 – Ilustração do descritor do serviço prestado pela Solução 2.

A Figura 5.8 apresenta o resultado do cálculo da similaridade entre o Problema Atual e a Solução 1. Nota-se que a coluna similaridade é calculada para cada termo de cada atributos e desta forma similaridade do atributo ontologia não recebeu a classificação zero. No caso do atributo de entrada pluviometria, que está presente em um serviço candidato e não é solicitado pelo cliente, o resultado da semelhança é zero. A partir da similaridade calculada para cada atributo, é feita uma média simples para o resultado final, neste caso o a similaridade entre o Problema Atual e a Solução 1 é 0.73.

Atributos		Problema Atual	Solução 1	Similaridade
Serviço	Nome	Escorregamento de Massa	Deslizamento de Terra	0.3
	Tipo	Cálculo de Risco	Cálculo de Risco	1
Ontologia		Calcula Risco	Calcula o Risco	0.6
Linguagem		FIPA ACL	FIPA ACL	1
Conteúdo	Entrada	Declividade	Declividade	1
		Vegetação	Vegetação	1
			Pluviometria	0
	Saída	Risco	Risco	1
Similaridade Total				0.73

Figura 5.8 – Ilustração da similaridade entre o Problema Atual e a Solução 1.

A Figura 5.9 apresenta o resultado do cálculo da similaridade entre o Problema Atual e a Solução 2. Nota-se que a similaridade entre os atributos nome do serviço recebeu a classificação 0.6. A similaridade entre o serviço atual, representado pelo Problema Atual, e o serviço candidato, representado pela Solução 2, é 0.74 e portanto, a Solução 2 é a mais adequada para substituir o serviço web que falhou.

Atributos		Problema Atual	Solução 2	Similaridade
Serviço	Nome	Escorregamento de Massa	Deslizamento de Massa	0.6
	Tipo	Cálculo de Risco	Cálculo de Risco	1
Ontologia		Calcula Risco	Calcula o Risco	0.6
Linguagem		FIPA ACL	W3C SOAP	0
Conteúdo	Entrada	Declividade	Declividade	1
		Vegetação	Vegetação	1
	Saída	Risco	Risco	1
Similaridade Total				0.74

Figura 5.9 - Ilustração da similaridade entre o Problema Atual e a Solução 2.

Ao término da execução do cálculo da similaridade, tem-se o serviço web escolhido para substituir o serviço falho. O serviço escolhido será enviado à atividade *Execute* para que esta realize as reconfigurações necessárias ao sistema completando assim a adaptação ao novo serviço web.

Continuado o exemplo anterior, tem-se que o serviço escolhido foi a Solução 2 por possuir maior similaridade ao serviço falho. A Solução 2, como pode ser observado na Figura 5.7 é provida por um serviço web que utiliza o padrão elaborado pela W3C. Neste caso, faz-se necessária a tradução das mensagens trocadas entre o sistema e o serviço web, uma vez que o sistema utilizado está implementado no padrão gentes (FIPA). Esta tradução de mensagens é executada pelo agente *JadeGatewayAgent* (Ver Seção 4.1.4.1), que é acionado na atividade *Execute*.

Por fim, a última atividade a ser executada é a *Execute*. Para a implementação desta atividade foram utilizadas as classes base do *framework*, como pode ser observado na Seção 4.1.4. Através da classe *Notification*, o sistema é informado que será necessária a tradução das mensagens, pois o novo serviço selecionado utiliza um padrão diferente do padrão FIPA utilizado pelos agentes. Deste modo, o agente *System*, que representa o sistema, passa a trocar mensagens com o agente *JadeGatewayAgent*, como se estes fosse o provedor do serviço. Já o agente *JadeGatewayAgent*, que é o responsável por executar a transcrição das mensagens, é acionado pela classe *ServiceConfiguration*, onde é informado que suas mensagens deverão ser encaminhas a um determinado serviço web e o retorno deve ser encaminhado ao agente *System*.

Observado que a Solução 2 foi escolhido como novo serviço, apresenta-se a seguir uma ilustração da tradução das mensagens realizada pelo agente *JadeGatewayAgent* a ser executada em todas as trocas de mensagens entre o sistema e o serviço web.

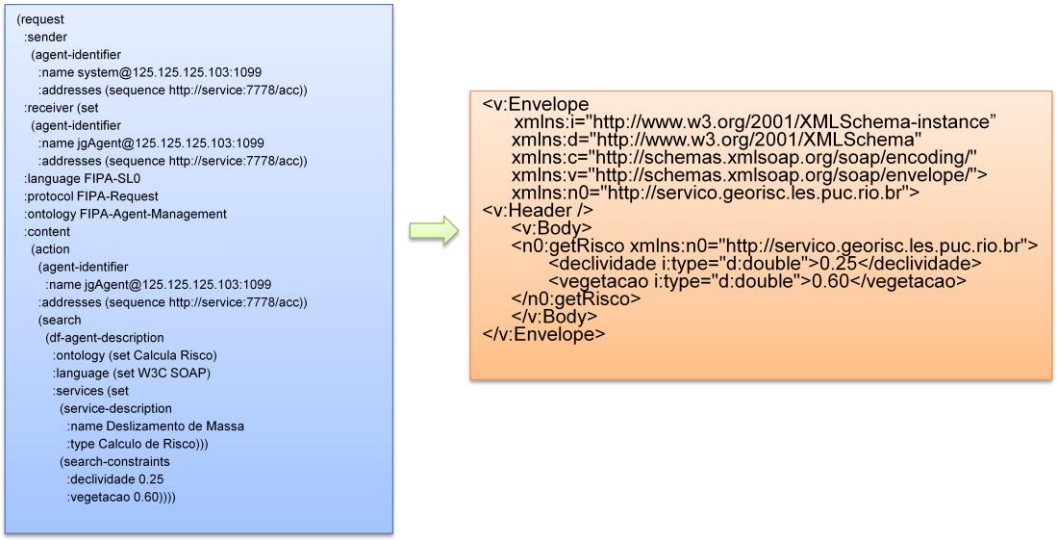

Figura 5.10 – Ilustração da transcrição da mensagem ACL para SOAP.

A Figura 5.10 apresenta a tradução da solicitação enviado pelo agente do sistema ao agente *JadeGatewayAgent*, em que a mensagem ACL foi transcrita para um envelope SOAP que será enviado ao serviço web. Ao receber o retorno do serviço web, o agente fará a transcrição inversa, SOAP para ACL, e encaminhará para o agente do sistema, Figura 5.11.

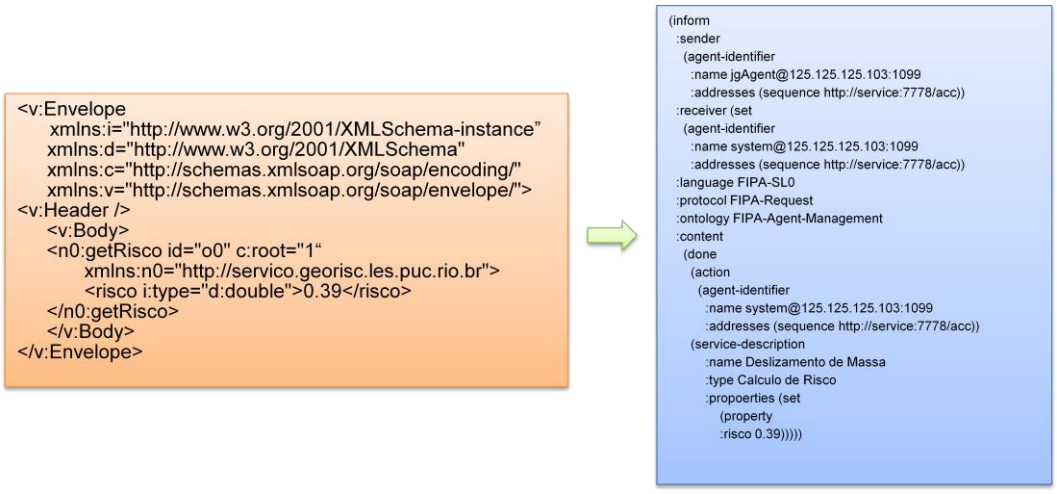

Figura 5.11 - Ilustração da transcrição da mensagem SOAP para ACL.

Assim, finaliza-se o clico de adaptação do sistema ao qual após a execução das atividades *Monitor*, *Analyze*, *Plan* e *Execute*, tem-se o sistema apto a executar

o novo serviço, sendo que o ciclo de autoadaptação foi realizado em tempo de execução e sem intervenção externa.

5.2. Agência de Viagens

Um grande desafio enfrentado por pessoas que querem viajar é encontrar um pacote de viagens que satisfaça plenamente suas necessidades. Muitas das vezes a procura por um pacote ideal se torna tão desgastante que o usuário simplesmente desiste de viajar ou aceita um pacote que atenda parcialmente seu desejo inicial. Além da dificuldade de encontrar o pacote de viagens ideal, em algumas vezes há a necessidade do usuário visitar diversos sites de modo a contemplar todos os passos da viagem como: sites de companhias aéreas, aluguel de veículos, hotéis, museus, dentre outros.

O desafio de encontrar um pacote de viagens que melhor satisfaça as necessidades do usuário é notadamente um problema clássico da ciência da computação e a pesquisa sobre este tema é incentivado pela *Carnegie Mellon University* e *Swedish Institute of Computer Science* através de campeonatos denominados *Trading Agent Competition* (TAC) (CMU; SICS, 2001). Neste contexto, foi desenvolvida uma aplicação para dispositivo móvel baseada no conceito de sistemas multiagentes e estendida do *framework* elaborado neste trabalho que tem o objetivo de encontrar o pacote de viagens de melhor atende as preferências do usuário, observando que se o serviço web utilizado atualmente para elabora o pacote de viagem apresentar falha de comunicação então o sistema será capaz de se adaptar a um novo serviço web.

5.2.1. Ideia Principal

O sistema multiagentes mencionado anteriormente recebe diversas informações do usuário como: nome, idade, profissão, meio de transporte, cidade de origem, cidade de destino, dia da partida, dia de retorno, tipo de acomodação, tipo de veículo, forma de pagamento e eventos. A partir dos dados fornecidos pelo usuário o sistema pode retornar pacotes com informações sobre reserva de hotel, passagem aérea, locação de veículo, ingresso para eventos e valor do pacote. Por exemplo, se o usuário fornece informações sobre cidade de origem e destino, dia

de partida e retorno e meio de locomoção marítimo, então o sistema retornará informações sobre pacotes de cruzeiros marítimos.

O sistema é composto de um agente *System* e um conjunto de Agente Agência de Viagens (AAV) que conhecem serviços por continentes: América do Norte e Europa. Deste modo, ao receber as informações do usuário, o agente *System* identifica a partir da cidade de destino qual AAV deverá ser consultado para obter o serviço que satisfaça as necessidades do usuário. Cada AAV está apto a executar o serviço que atenda as necessidades do usuário utilizando os dados enviados pelo agente *System*, no entanto, caso ocorra alguma falha de comunicação, falha de autenticação ou tempo demasiadamente longo de resposta com o serviço web, o AAV está apto a se adaptar a um novo serviço web que satisfaça o usuário.

Para superar as dificuldades citadas acima, os AAV possuem as seguintes bases de conhecimentos para realizar as adaptações: uma base de regras a qual é possível detectar problemas a partir dos dados coletados no monitoramento dos serviços web e das mensagens trocadas entre o sistema e serviço web; e um conjunto de descrições DF que descrevem os serviços web atualmente ativos. Cada AAV possui um *control loop*, provido pelo *framework*, responsável por realizar a adaptação do sistema, descrito em maiores detalhes a seguir.

5.2.2. Agente Agência de Viagens

Na atividade *Monitor* (Ver Seção 4.1.1) foram utilizados os monitoramentos de serviço web disponibilizado pela classe *ServiceMonitor* e de mensagens disponibilizado pela classe *MessageMonitor*. O primeiro monitora o funcionamento dos serviços web obtendo informações sobre a execução do mesmo e o segundo monitora a troca de mensagens entre o sistema e o serviço web. Também foram criadas as classes *TripTranslatorSM* estendida da classe *ServiceMonitor* e *TripTranslatorMM* estendida da classe *MessageMonitor*, que são responsáveis por: implementar o método *translator*, que estrutura os dados coletados a partir do monitoramento do serviço web; e implementar o método *translator*, que estrutura os dados coletados a partir da troca de mensagens entre o sistema e o serviço web, de modo que possam ser entendidos pela classe *Analyze*. A Figura 5.12 apresenta a classe *Monitor* do AAV.

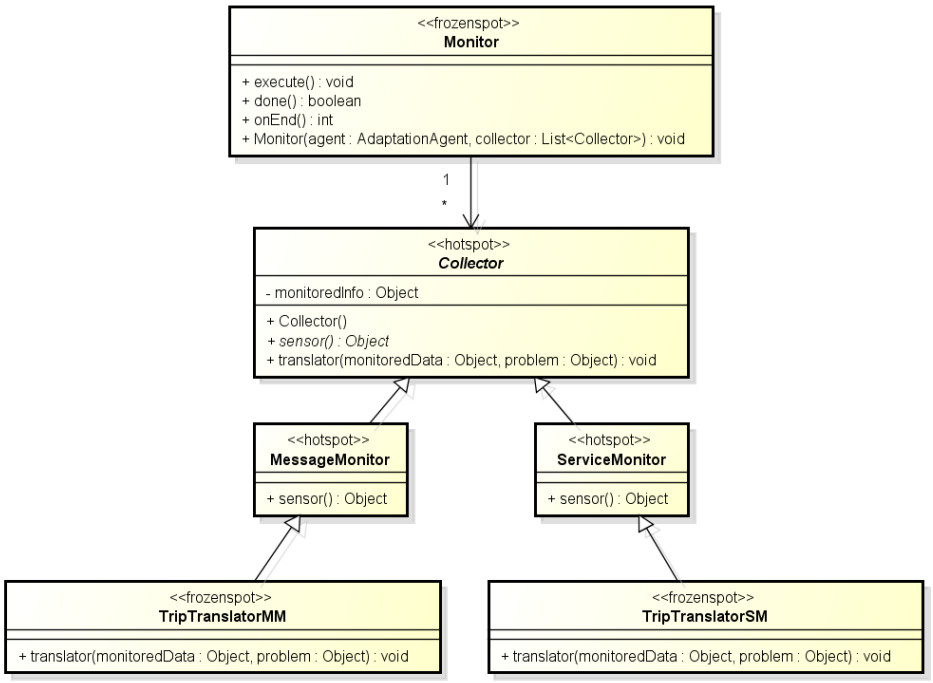

Figura 5.12 – Atividade *Monitor* de um AAV.

Logo após a finalização da atividade *Monitor*, a atividade *Analyze* entra em execução. Nesta atividade, é utilizado o mecanismo de raciocínio baseado em regras (Ver Seção 4.1.2.1) para identificar o problema detectado, onde se tem as regras R1, R2 e R3, importadas pelo método *importBase* da classe *TripBase*, como pode ser observado na Figura 5.13. Para esta aplicação, tem-se definida uma base de regras contendo três regras, descritas a seguir:

Regra 1 : If *serviceFailure > 0* Then problem = "*serviceFail*"

Regra 2 : If "*authenticationFailure*" Then problem = "*authenticationFail*"

Regra 3 : If *responseTime* > 30 Then problem = "*timeout*"

A Regra 1 detecta a falha de comunicação com o serviço web sempre que sua quantidade for maior que zero para a variável *serviceFailure*. Já a Regra 2 constata a ocorrência de falha na autenticação do usuário através da variável *authenticationFailure*. Por fim temos a Regra 3 em que verifica se tempo de resposta do serviço web é excessivamente longo e neste caso temos o problema do *timeout*, conferido através da variável *responseTime*.

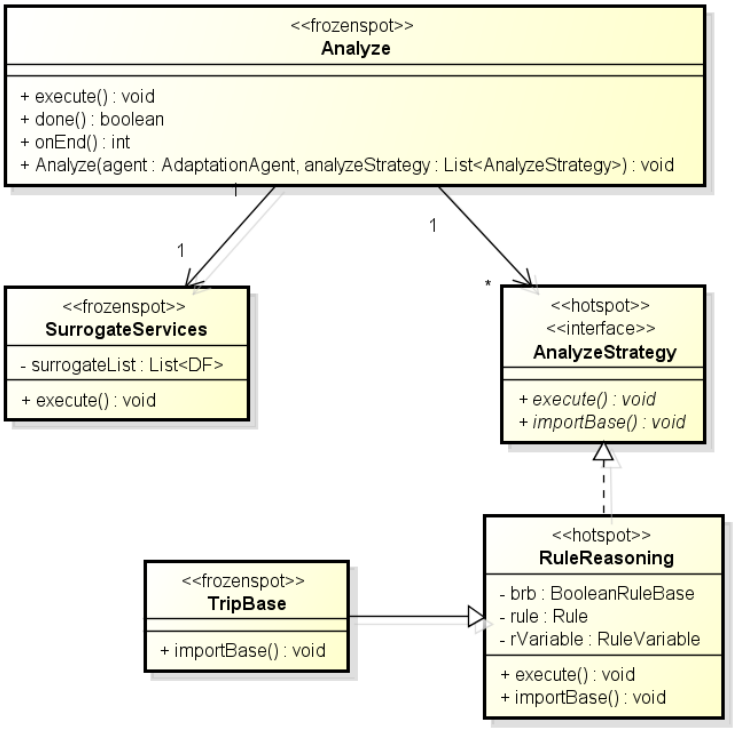

Figura 5.13 – Atividade *Analyze* de um AAV.

O RBR é formado por um par "problema-solução", portanto, se nenhum problema for detectado a partir da aplicação das regras citadas anteriormente, então nenhuma adaptação é necessária e o sistema continuará sua execução. Se houver algum problema, e este se encaixa com alguma das regras supracitadas, então já está estabelecida a solução, neste caso um serviço substituto. Caso contrário, a atividade *Analyze* continuará sua execução, agora elaborando uma lista de serviços candidatos através da classe *SurrogaServices*. Nesta classe, o método *execute* é responsável por percorrer o registro DF a procura de serviços web que ofereçam o serviço procurado, ou seja, um serviço que encontre o pacote de viagem que atenda as necessidades do usuário. O próximo passo é repassar à atividade *Plan*, o problema atual e a lista de serviços web candidatos a substitutos.

Para melhor entendimento do funcionamento do RBR observe os casos a seguir, em que o problema atual é coincidente com uma das regras e neste caso já se tem uma solução pré-estabelecida.

Caso 1

Descrição das preferências do usuário:

Idade: 26

Meio de Transporte: Avião

Cidade de Origem: Rio de Janeiro

Cidade de Destino: Fernando de Noronha

Dia da Partida: 05/12/2012

Dia de Retorno: 20/12/2012

Tipo de Acomodação: Hotel 4 estrelas

Tipo de Serviço: Reserva de Pacote de Viagem

Problema: "*timeout*"

Solução do Caso 1

Descrição do Serviço: A solução para o Caso 1 é descrita no WSDL da Figura 5.14, que oferece o serviço reserva de pacotes de viagens com as seguintes características:

Tipo de Serviço: Reserva de Pacote de Viagem.

Entradas: Parâmetros do tipo *string* que representam os atributos idade, meio de transporte, cidade de origem e destino, dia da partida e retorno e o tipo de acomodação.

Saída: A confirmação da reserva das passagens aéreas e do hotel.

Caso 2

Descrição das preferências do usuário:

Idade: 45

Meio de Transporte: Marítimo

Cidade de Origem: Rio de Janeiro

Cidade de Destino: Ibiza

Dia da Partida: 05/06/2012

Dia de Retorno: 05/07/2012

Tipo de Acomodação: Cabine Externa - Luxo

Forma de Pagamento: Cartão de Crédito

Tipo de Serviço: Reserva de Pacote de Viagem

Problema: "*serviceFail*"

Solução do Caso 2

Descrição do Serviço: A solução para o Caso 2 está no descritor da Figura 5.15, que oferece o serviço reserva de pacotes de viagens com as seguintes características:

Tipo de Serviço: Reserva de Pacote de Viagem.

Entradas: Parâmetros do tipo *string* que representam os atributos idade, meio de transporte, cidade de origem e destino, dia da partida e retorno, tipo de acomodação e forma de pagamento.

Saída: A confirmação da reserva da cabine no cruzeiro marítimo.

Deste modo, ao se identificar o problema "*timeout"* o serviço web que será executado é o serviço representado pelo WSDL da Figura 5.14, já no caso de uma falha do tipo "*serviceFail*" o serviço web que será executado está representado pelo descritor da Figura 5.15. Observe que, para falhas conhecidas têm-se serviços substitutos que atendem plenamente a necessidade do sistema, o que pode não ocorre quando se escolhe um novo serviço web a partir de uma lista de serviços web candidatos elaborada em tempo de execução. Como meio de obter o serviço web que melhor atende as necessidades do sistema, foi elaborado um mecanismo de similaridade, incorporado à atividade *Plan*, de modo a identificar o serviço web, dentre a lista de candidatos, mais similar ao serviço atual.

```
- <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:ns1="http://org.apache.axis2/xsd"
        xmlns:ns="http://servico.av.lfg.br" xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
        xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
        targetNamespace="http://servico.av.lfg.br">
        <wsdl:documentation>Reserva de Pacote de Viagem</wsdl:documentation>
- <wsdl:types>
- <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNamespace="http://servico.av.lfg.br ">
- <xs:element name="getReserva">
- <xs:complexType>
- <xs:sequence>
<xs:element minOccurs="0" name="idade" type="xs:string" />
<xs:element minOccurs="0" name="meiodetransporte" type="xs:string" />
<xs:element minOccurs="0" name="cidadeorigem" type="xs:string " />
<xs:element minOccurs="0" name="cidadedestino" type="xs:string " />
<xs:element minOccurs="0" name="diapartida" type="xs:string " />
<xs:element minOccurs="0" name="diaretorno" type="xs:string " />
<xs:element minOccurs="0" name="tipoacomodacao" type="xs:string " />
    </xs:sequence>
    </xs:complexType>
    </xs:element>
- <xs:element name="getReservaResponse">
- <xs:complexType>
- <xs:sequence>
<xs:element minOccurs="0" name="return" type="xs:boolean" />
    </xs:sequence>
    </xs:complexType>
    </xs:element>
    </xs:schema>
    </wsdl:types>
- <wsdl:message name="getReservaRequest">
<wsdl:part name="parameters" element="ns:getReserva" />
    </wsdl:message>
- <wsdl:message name="getReservaResponse">
<wsdl:part name="parameters" element="ns:getReservaResponse" />
    </wsdl:message>
- <wsdl:portType name="AgenciaViagensPortType">
- <wsdl:operation name="getReserva">
<wsdl:input message="ns:getReservaRequest" wsaw:Action="urn:getReserva" />
<wsdl:output message="ns:getReservaResponse" wsaw:Action="urn:getReservaResponse" />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="AgenciaViagensSoap11Binding" type="ns: AgenciaViagensPortType">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <wsdl:operation name="getReserva">
<soap:operation soapAction="urn:getReserva" style="document" />
- <wsdl:input>
<soap:body use="literal" />
    </wsdl:input>
- <wsdl:output>
<soap:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:binding name="AgenciaViagensSoap12Binding" type="ns: AgenciaViagensPortType">
<soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <wsdl:operation name="getReserva">
<soap12:operation soapAction="urn:getReserva" style="document" />
- <wsdl:input>
<soap12:body use="literal" />
    </wsdl:input>
- <wsdl:output>
<soap12:body use="literal" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:binding name="AgenciaViagensHttpBinding" type="ns: AgenciaViagensPortType">
<http:binding verb="POST" />
- <wsdl:operation name="getReserva">
<http:operation location="AgenciaViagens/getReserva" />
- <wsdl:input>
<mime:content type="text/xml" part="getReserva" />
    </wsdl:input>
- <wsdl:output>
<mime:content type="text/xml" part="getReserva" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:service name="AgenciaViagens">
- <wsdl:port name="AgenciaViagensHttpSoap11Endpoint" binding="ns: AgenciaViagensSoap11Binding">
<soap:address location="http://localhost:8080/AgenciaViagensWS/services/AgenciaViagens.
        AgenciaViagensHttpSoap11Endpoint/" />
    </wsdl:port>
- <wsdl:port name="AgenciaViagensHttpSoap12Endpoint" binding="ns: AgenciaViagensSoap12Binding">
<soap12:address location="http://localhost:8080/AgenciaViagensWS/services/AgenciaViagens.
        AgenciaViagensHttpSoap12Endpoint/" />
    </wsdl:port>
- <wsdl:port name="AgenciaViagensHttpEndpoint" binding="ns: AgenciaViagensHttpBinding">
<http:address location="http://localhost:8080/AgenciaViagensWS/services/AgenciaViagens. AgenciaViagensHttpEndpoint/"
        />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

Figura 5.14 – WSDL da Solução do Caso 1.

```
(inform
 :sender
  (agent-identifier
   :name df@125.125.125.103:1099
   :addresses (sequence http://service:7778/acc))
 :receiver (set
  (agent-identifier
   :name system@125.125.125.103:1099
   :addresses (sequence http://service:7778/acc)))
 :language FIPA-SL0
 :protocol FIPA-Request
 :ontology FIPA-Agent-Management
:content
  (done
   (action
    (agent-identifier
     :name df@125.125.125.103:1099
     :addresses (sequence http://service:7778/acc))
    (register
     (df-agent-description
      :name
       (agent-identifier
        :name avAgent@125.125.125.103:1099
        :addresses (sequence http://service:7778/acc))
        :services (set
         (service-description
          :name Agencia de Viagens
          :type Reserva de Pacote de Viagem
          :ontology Reserva
          :properties (set
           (property
            :name idade
            :value string)
          (property
            :name meiotransporte
            :value string)
          (property
            :name cidadeorigem
            :value string)
          (property
            :name cidadedestino
            :value string)
          (property
            :name diapartida
            :value string)
          (property
            :name diaretorno
            :value string)
          (property
            :name tipoacomodacao
            :value string)
          (property
            :name formapagamento
            :value string)))))))))
```

Figura 5.15 – Descritor DF da Solução do Caso 2.

Já na atividade *Plan*, representada na Figura 5.16, foram utilizadas as classes base de modo a aproveitar o algoritmo de similaridade elaborado por Resnik (Ver Seção 4.1.3.1) já incorporado ao *framework*.

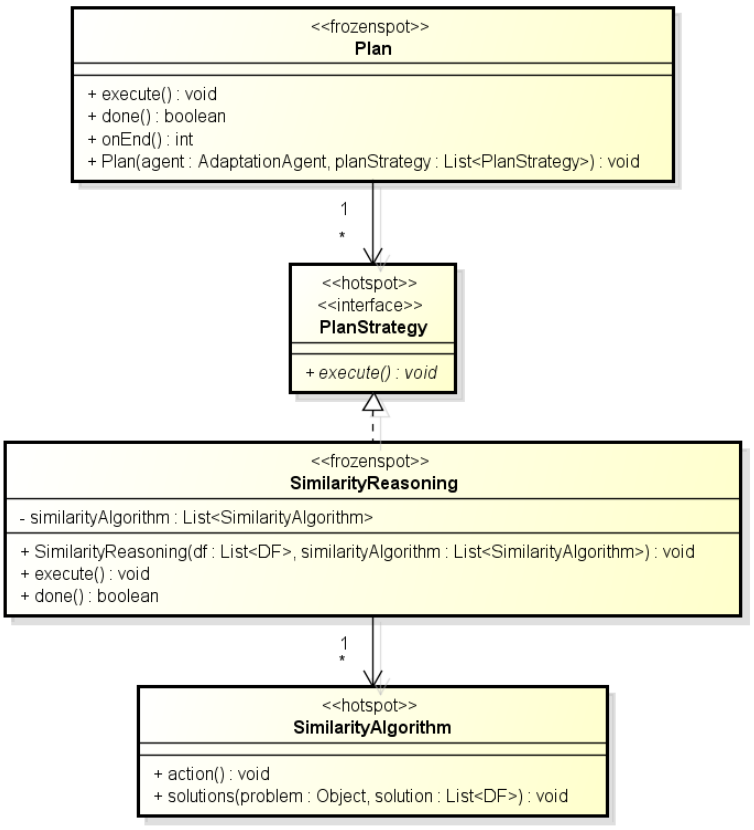

Figura 5.16 – Atividade *Plan* de um AAV.

Considerando que a solução não foi encontrada na atividade *Analyze*, através do RBR, então a tarefa de identificar um novo serviço web cabe à atividade *Plan*, que recebe a lista de serviços candidatos e o problema atual. O método *execute* da classe *SimilarityReasoning* será executado chamando o método *solutions* para o cálculo da similaridade.

Note que na seleção do novo serviço web, este pode utilizar um protocolo de dados, comunicação e descrição diferente do padrão FIPA utilizado pelos agentes. Caso este fato se confirme, então será necessária a transcrição das mensagens do padrão dos agentes (FIPA) para o padrão de serviços web (W3C). Esta atividade de transcrição se configura na atividade *Execute*, detalhada mais a frente. Por hora, observe os casos a seguir exemplificando o funcionamento do mecanismo de similaridade, em que a Solução 1 é está representada na Figura 5.17 e a Solução 2 na Figura 5.18:

Problema Atual

Serviços: O serviço solicitado deve conter os parâmetros: nome do serviço (Agência de Viagem) e tipo de serviço (Reserva de Pacote de Viagem);

Ontologia: Reserva Pacote;

Conteúdo:

Entrada: As informações repassadas ao serviço web são: cidade de origem e destino, dia de partida e retorno, meio de transporte e tipo de acomodação;

Saída: O retorno esperado é a reserva do pacote (representado por um valor booleano[1]).

Solução 1

Serviço: O serviço contêm os parâmetros: nome do serviço (Agência de Viagens) e tipo de serviço (Reserva de Pacote de Viagem);

Ontologia: Reserva Pacote Turístico;

Conteúdo:

Entrada: As informações solicitadas pelo serviço web são: cidade de origem e destino, dia de partida e retorno, meio de transporte, tipo de acomodação e forma de pagamento;

Saída: O retorno é a reserva do pacote (representado por um valor booleano).

Solução 2

Serviço: O serviço contêm os parâmetros: nome do serviço (Agência de Viagem) e tipo de serviço (Reserva de Pacote de Viagem);

Ontologia: Reserva Pacote;

Conteúdo:

Entrada: As informações solicitadas pelo serviço web são: cidade de origem e destino, dia de partida e retorno, meio de transporte, tipo de acomodação;

Saída: O retorno é a reserva do pacote (representado por um valor booleano) e as formas de pagamento.

[1] Em ciência da computação, booleano é um tipo de dado primitivo que pode assumir dois valores, podendo ser representado por 0 ou 1 e também por verdadeiro ou falso.

```
(inform
 :sender
  (agent-identifier
   :name df@125.125.125.103:1099
   :addresses (sequence http://service:7778/acc))
 :receiver (set
  (agent-identifier
   :name system@125.125.125.103:1099
   :addresses (sequence http://service:7778/acc)))
 :language FIPA-SL0
 :protocol FIPA-Request
 :ontology FIPA-Agent-Management
 :content
  (done
   (action
    (agent-identifier
     :name df@125.125.125.103:1099
     :addresses (sequence http://service:7778/acc))
    (register
     (df-agent-description
      :name
       (agent-identifier
        :name avAgent1@125.125.125.103:1099
        :addresses (sequence http://service:7778/acc))
        :services (set
         (service-description
          :name Agencia de Viagens
          :type Reserva de Pacote de Viagem
          :ontology Reserva de Pacote Turistico
          :properties (set
           (property
            :name cidadeorigem
            :value string)
           (property
            :name cidadedestino
            :value string)
           (property
            :name diapartida
            :value string)
           (property
            :name diaretorno
            :value string)
           (property
            :name meiotransporte
            :value string)
           (property
            :name tipoacomodacao
            :value string)
           (property
            :name formapagamento
            :value string)))))))))
```

Figura 5.17 – Ilustração do descritor do serviço prestado pela Solução 1.

```
(inform
 :sender
  (agent-identifier
   :name df@125.125.125.103:1099
   :addresses (sequence http://service:7778/acc))
 :receiver (set
  (agent-identifier
   :name system@125.125.125.103:1099
   :addresses (sequence http://service:7778/acc)))
 :language FIPA-SL0
 :protocol FIPA-Request
 :ontology FIPA-Agent-Management
:content
  (done
   (action
    (agent-identifier
     :name df@125.125.125.103:1099
     :addresses (sequence http://service:7778/acc))
    (register
     (df-agent-description
      :name
       (agent-identifier
        :name avAgent2@125.125.125.103:1099
        :addresses (sequence http://service:7778/acc))
        :services (set
         (service-description
          :name Agencia de Viagem
          :type Reserva de Pacote de Viagem
          :ontology Reserva Pacote
          :properties (set
           (property
            :name cidadeorigem
            :value string)
          (property
            :name cidadedestino
            :value string)
          (property
            :name diapartida
            :value string)
          (property
            :name diaretorno
            :value string)
          (property
            :name meiotransporte
            :value string)
          (property
            :name tipoacomodacao
            :value string)))))))))
```

Figura 5.18 – Ilustração do descritor do serviço prestado pela Solução 2.

A Figura 5.19 apresenta o resultado do cálculo da similaridade entre o Problema Atual e a Solução 1. Nota-se que a coluna similaridade é calculada para cada termo de cada atributos e desta forma similaridade dos atributos nome do serviço e ontologia não receberam a classificação zero. No caso do atributo de

entrada forma de pagamento, que está presente na Solução 1 e não é solicitado pelo cliente, o resultado da semelhança é zero. A partir da similaridade calculada para cada atributo, é feita uma média simples para o resultado final, neste caso o a similaridade entre o Problema Atual e a Solução 1 é 0.84.

Atributos		Problema Atual	Solução 1	Similaridade
Serviço	Nome	Agência de Viagem	Agência de Viagens	0.6
	Tipo	Reserva de Pacote de Viagem	Reserva de Pacote de Viagem	1
Ontologia		Reserva Pacote	Reserva Pacote Turístico	0.6
Conteúdo	Entrada	Cidade de Origem	Cidade de Origem	1
		Cidade de Destino	Cidade de Destino	1
		Dia de Partida	Dia de Partida	1
		Dia de Retorno	Dia de Retorno	1
		Meio de Transporte	Meio de Transporte	1
		Tipo de Acomodação	Tipo de Acomodação	1
			Forma de Pagamento	0
	Saída	Reserva	Reserva	1
Similaridade Total				0.84

Figura 5.19 – Ilustração da similaridade entre o Problema Atual e a Solução 1.

A Figura 5.20 apresenta o resultado do cálculo da similaridade entre o Problema Atual e a Solução 2. Nota-se que a similaridade entre os atributos de entrada receberam a nota máxima 1. Já o atributo de saída "forma de pagamento" que não foi solicitado pelo usuário recebe classificação zero. A similaridade entre os serviço atual, representado pelo Problema Atual, e o serviço candidato, representado pela Solução 2, é 0.91 e portanto, a Solução 2 é a mais adequada para substituir o serviço web que falhou.

Atributos		Problema Atual	Solução 2	Similaridade
Serviço	Nome	Agência de Viagem	Agência de Viagem	1
	Tipo	Reserva de Pacote de Viagem	Reserva de Pacote de Viagem	1
Ontologia		Reserva Pacote	Reserva Pacote	1
Conteúdo	Entrada	Cidade de Origem	Cidade de Origem	1
		Cidade de Destino	Cidade de Destino	1
		Dia de Partida	Dia de Partida	1
		Dia de Retorno	Dia de Retorno	1
		Meio de Transporte	Meio de Transporte	1
		Tipo de Acomodação	Tipo de Acomodação	1
	Saída	Reserva	Reserva	1
			Formas de Pagamento	0
Similaridade Total				0.91

Figura 5.20 - Ilustração da similaridade entre o Problema Atual e a Solução 2.

Ao término da execução do cálculo da similaridade, tem-se o serviço web escolhido para substituir o serviço falho. O serviço escolhido será enviado à atividade *Execute* para que esta realize as reconfigurações necessárias ao sistema completando assim a adaptação ao novo serviço web.

Por fim, a última atividade a ser executada é a *Execute*. Para a implementação desta atividade foram utilizadas as classes base do *framework*, como pode ser observado na Seção 4.1.4. Através da classe *Notification*, o sistema é informado das novas configurações do serviço web escolhido. Continuado o exemplo anterior, tem-se que o serviço escolhido foi a Solução 2 por possuir maior similaridade ao serviço falho. A Solução 2, como pode ser observado na Figura 5.18 é provida por um serviço web que utiliza o padrão FIPA dos agentes. Neste caso, não se faz necessária a tradução das mensagens trocadas entre o sistema e o serviço web e, portanto o agente tradutor de mensagens, *JadeGatewayAgent*, não será solicitado pela classe *ServiceConfiguration*.

Assim, finaliza-se o clico de adaptação do sistema ao qual após a execução das atividades *Monitor*, *Analyze*, *Plan* e *Execute*, tem-se o sistema apto a executar o novo serviço, sendo que o ciclo de autoadaptação foi realizado em tempo de execução e sem intervenção externa.