4
Framework

Markiewicz e Lucena (MARKIEWICZ e LUCENA, 2001) descrevem *framework* como “a pedra angular da engenharia de software moderna”. O desenvolvimento de *frameworks* obteve boa aceitação devido as suas habilidades de prover reutilização da arquitetura e código fonte. Desta forma, pode-se dizer que *framework* é uma ferramenta que visa auxiliar a elaboração de uma solução para um problema específico de um domínio, uma vez que, este reutilizará a abstração de classes utilizadas na solução de um problema similar.

Para fornecer a flexibilidade necessária ao *framework*, este é composto de pontos fixos e pontos flexíveis. Segundo Markiewicz e Lucena, “pontos flexíveis são classes e métodos abstratos que devem ser implementados pelo usuário e pontos fixos são classes e métodos imutáveis presentes no cerne do *framework”*. A partir dos pontos flexíveis é possível que o desenvolvedor estenda o *framework* para tratar um problema específico, desde que esteja no mesmo domínio de problemas.

Para facilitar a implementação de aplicações capazes de superar questões de indisponibilidade e interoperabilidade na troca de informações entre dispositivos móveis e serviços web, foi desenvolvido um *framework* baseado no conceito de agentes (Ver Seção 3.4) guiados por um *control loop* conforme descrito pela IBM (BELL, 2004) e pode ser observado na Figura 4.1. Deste modo, o dispositivo móvel contém o sistema que é composto de agentes que se comunicam com os serviços web.

PUC-Rio - Certificação Digital Nº 1012657/CA

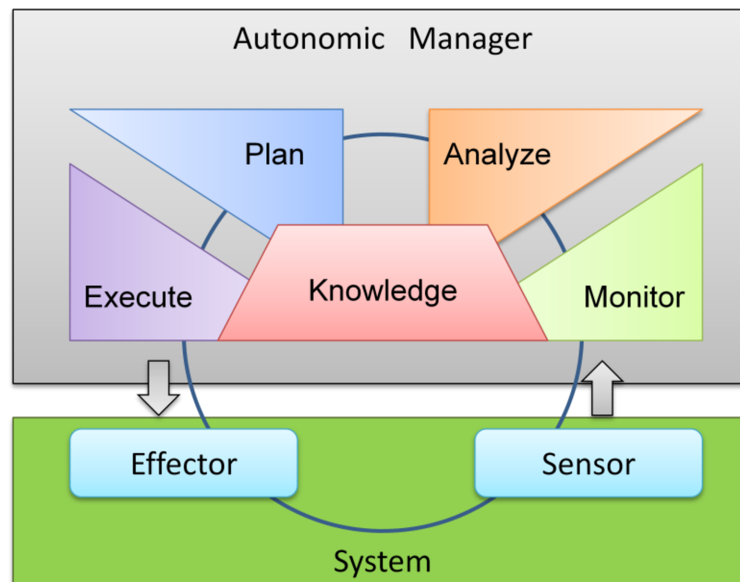

Figura 4.1 – Ilustração do *Control loop* elaborado pela IBM (BELL, 2004).

O *framework* implementado neste trabalho utiliza o *control loop* elaborado pela IBM (BELL, 2004) para controlar os passos na realização da adaptação. O *control loop* é composto de quatro funcionalidades distintas, que implementadas neste trabalho são capazes de realizar as reconfigurações necessárias para que as aplicações continuem sua execução sem interrupções. A seguir temos um resumo das atividades executas em cada etapa do *control loop*.

Monitor : Esta atividade utiliza mecanismos para monitorar as mensagens trocadas entre o agente e o serviço web a fim de extrair informações que sirvam de base para detectar possíveis falhas de execução. Também é responsável por estruturar estas informações de modo a serem compreendidas pelo *Analyze*;

Analyze : A partir das informações recebidas do *Monitor*, o *Analyze* emprega um mecanismo de raciocínio para detectar o tipo de falha ocorrida na comunicação entre o agente e o serviço web. Adicionalmente, elabora uma lista de serviços substitutos;

Plan : Utiliza de um mecanismo de similaridade para encontrar um serviço substituto, a partir dos novos serviços identificados pela atividade anterior. Avalia também a necessidade de tradução das informações (Ver Seção 4.1.4.1) na comunicação com o novo serviço uma vez que este poderá

PUC-Rio - Certificação Digital Nº 1012657/CA

utilizar um padrão de comunição distinto do anterior (Ver Seções 3.1 e 3.4);

Execute : Realiza as configurações no sistema para que este possa se comunicar com o novo serviço selecionado pela atividade *Plan*.

Os componentes *Sensors* são responsáveis por coletar as informações da fonte de informação e repassar ao *Monitor* e os *Effectors* por aplicar as ações estabelecidas pelo *Execute*. O componente *Knowledge* é a base de conhecimento utilizada para realizar a adaptação, que neste trabalho é representada pela base de regras do mecanismo de raciocínio baseado em regras (Ver Seção 4.1.2.1). Por fim, o componente *System* representa o sistema ao qual o *control loop* monitora e realiza as adaptações.

O *framework* elaborado nesse trabalho é composto de dois agentes, o *AdaptationAgent* e o *JadeGatewayAgent*[1]. As atividades explicadas anteriormente são executadas pelo *AdaptationAgent*. Este agente está constantemente monitorando as trocas de mensagens entre o agente e o serviço web. Caso detecte uma dificuldade de comunicação com o serviço web, inicia-se imediatamente o mecanismo de adaptação a falhas, guiado pelo *control loop*. Caso o novo serviço selecionado utilize um protocolo de dados diferente do padrão FIPA utilizado pelos agentes, então o segundo agente do sistema entrará em ação, neste caso o agente *JadeGatewayAgent*. Este agente fará a intermediação das mensagens trocadas entre o sistema e o serviço web, realizando a tradução das mensagens do padrão dos agentes (FIPA) para o padrão *Web Services* (W3C), como pode ser melhor observado na Seção 4.1.4.1.

4.1. Diagrama de Classes

O diagrama de classes ilustrado pela Figura 4.2 apresenta as principais classes e métodos do *framework*. É possível observar que a autoadaptação realizada pelo sistema é controlada pelo agente *AdaptationAgent*, guiado pelo ciclo *AdaptationControlLoop*. Por ser um agente adaptado para dispositivos móveis, este estende a classe *GatewayAgent* do JADE-LEAP (Ver Seção 3.4.1).

[1] O agente *JadeGatewayAgent* foi incorporado ao trabalho através do *framework* WSIG elaborado pela Telecom Italia SpA (JADE, 2011).

PUC-Rio - Certificação Digital Nº 1012657/CA

Esta classe contém os comportamentos *Monitor* (Ver Seção 4.1.1), *Analyze* (Ver Seção 4.1.2), *Plan* (Ver Seção 4.1.3) e *Decision* (Ver Seção 4.1.4), que serão executados pelo agente, bem como sua lógica de execução, que será executada uma única vez, sendo assim estendida do comportamento *OneShotBehaviour* do JADE, através da classe *Behaviour*. Observe que os comportamentos e a lógica de execução implementada pelo *AdaptationControlLoop* são pontos fixos e não podem ser alterados. O agente *JadeGatewayAgent* somente entrará em execução se for necessário a tradução das mensagens, neste caso, o agente *AdaptationAgente* através do comportamento *Execute* informará ao *JadeGatewayAgent* a necessidade de seu serviço.

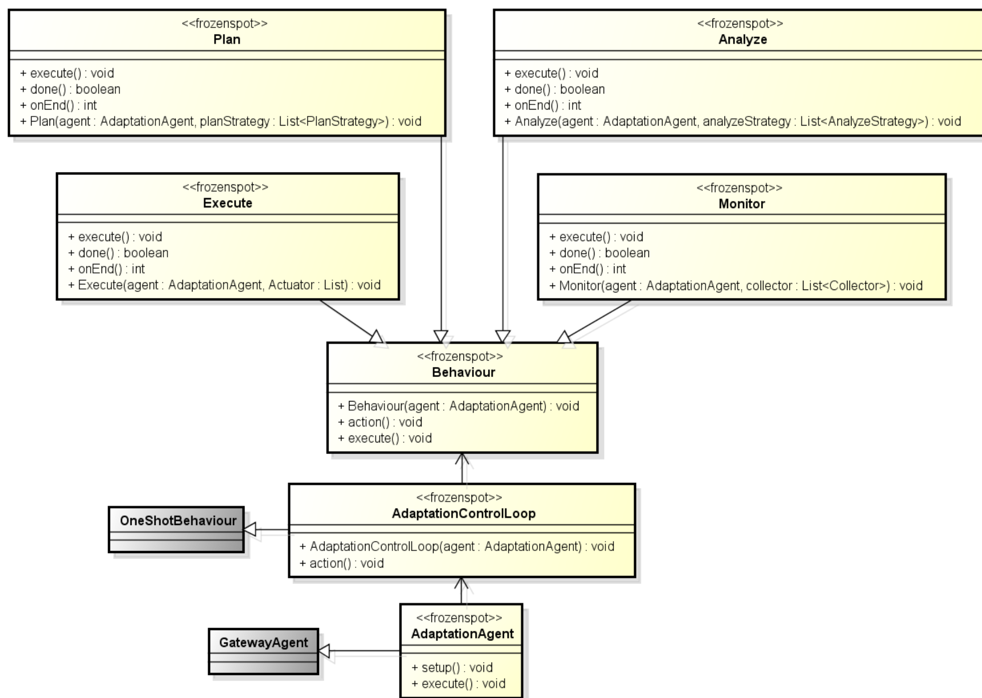

Figura 4.2 – Visão geral do *framework* proposto.

Para facilitar a compreensão das atividades executadas pelo *control loop*, apresenta-se a seguir os diagramas de classes separadamente para cada um de suas quatro etapas: *Monitor, Analyze, Plan* e *Execute*.

PUC-Rio - Certificação Digital Nº 1012657/CA

4.1.1. Monitor

Esta é a primeira atividade executada pelo *control loop* implementado neste trabalho. O *Monitor* é responsável por supervisionar as mensagens trocadas entre o agente e o serviço web, coletar estas mensagens e estruturá-las de forma que possam ser compreendidas pela próxima atividade. A Figura 4.3 apresenta o diagrama de classes para a atividade *Monitor*, onde esta é uma extensão da classe *Behaviour*, representando assim o primeiro comportamento a ser executado pelo *AdaptationAgent*.

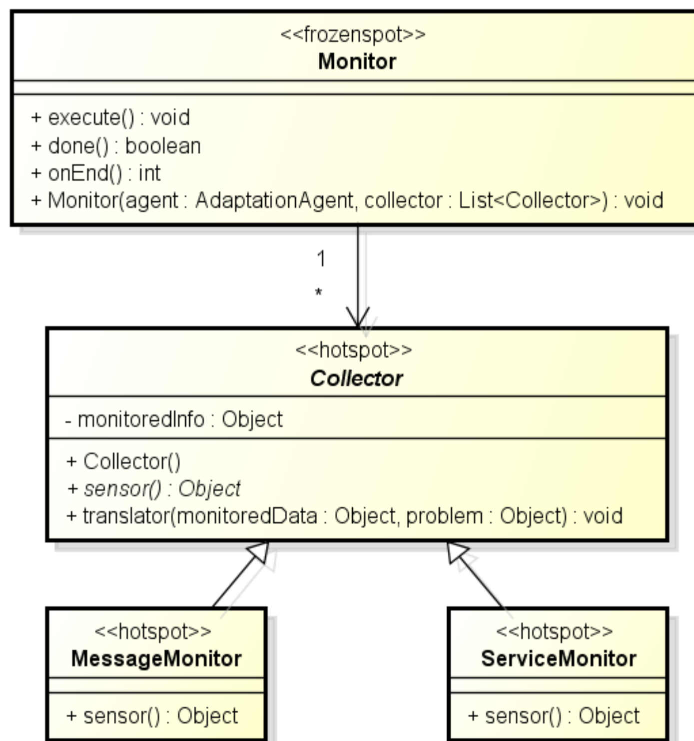

Figura 4.3 – Atividade *Monitor*.

Dentre suas responsabilidades, a classe *Monitor* é responsável por gerenciar um ou mais coletores, representados pela classe *Collector*, sendo estes os encarregados de monitorar e estruturar as mensagens trocadas entre o agente e o serviço web. O *Collector* realiza suas ações através do *sensor* e *translator*, descritos abaixo:

sensor : No *sensor* devem ser especificados os atributos que serão monitorados e coletados. Neste trabalho oferece-se o mecanismo *MessageMonitor* que capta a última mensagem trocada entre o agente e o serviço web. Podem-

PUC-Rio - Certificação Digital Nº 1012657/CA

se coletar informações como nome, tipo do serviço, propriedades, ontologias, linguagem, dentre outras. No caso de serviços web construídos com a tecnologia de *Web Services,* pode-se utilizar a classe *ServiceMonitor,* que utiliza a API elaborada por Perry (PERRY, 2002), para captar informações como tempo de resposta, segurança, escalabilidade, dentre outras.

translator : No *translator* devem ser implementados os atributos que definem os dados que serão coletados, representado pelo parâmetro *monitoredData* do tipo *Object* e o atributo que manipulará a última mensagem trocada entre os sistemas ou os dados coletados do serviço web, representado pelo parâmetro *problem* do tipo *Object*. A partir destas informações, o *translator* estruturará os dados de forma a serem compreendidos pela próxima atividade.

Ao final da execução do *Monitor*, teremos a mensagem de falha estruturada de forma que possa ser compreendida pela atividade *Analyze*. O próximo passo é a execução da atividade *Analyze*, sendo esta responsável por identificar o tipo de falha ocorrido e elaborar uma lista de possíveis serviços substitutos.

4.1.2. Analyze

A segunda atividade executada pelo *control loop* é o *Analyze*. Esta atividade é responsável por identificar o tipo de falha ocorrida na troca de mensagens entre o agente e o serviço web, e elaborar uma lista de possíveis serviços substitutos e enviá-la à próxima atividade chamada *Plan*. Esta atividade também é um comportamento estendido da classe *Behaviour* e está representada pelo diagrama de classes da Figura 4.4.

PUC-Rio - Certificação Digital Nº 1012657/CA

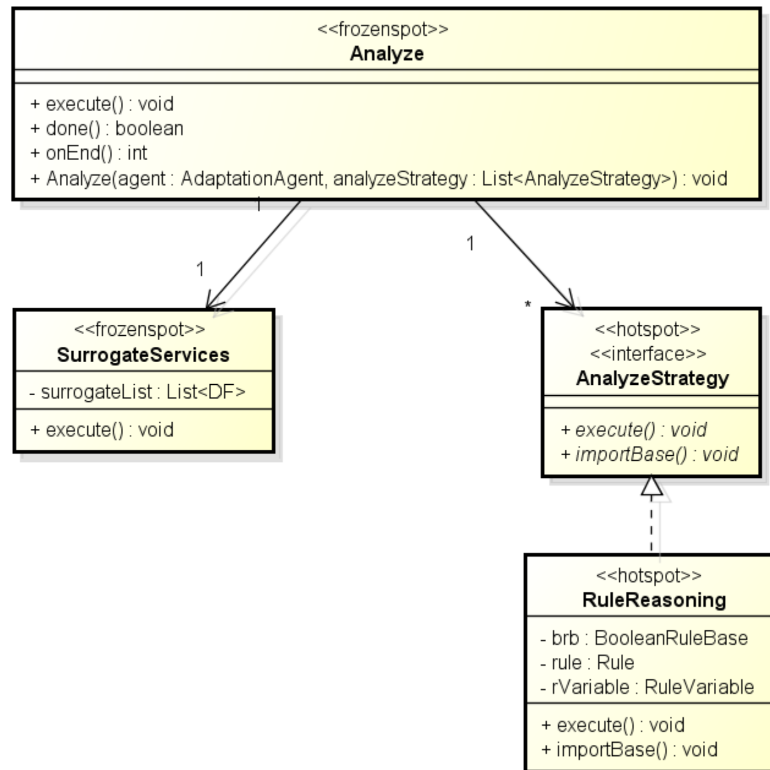

Figura 4.4 – Atividade *Analyze*.

Dentre suas responsabilidades, a classe *Analyze* é responsável por gerenciar as estratégias de raciocínios, representados pela interface *AnalyzeStrategy*, onde dois métodos devem ser implementados para utilização do raciocínio baseado em regras. Devem ser definidas uma base de regras através do método *importBase* e a lógica de execução no método *execute*. A classe *Analyze* também é responsável por elaborar uma lista de serviços substitutos caso seja identificada uma falha de comunicação entre o sistema e o serviço web, o mecanismo para identificação e preparação da lista de substitutos é implementado através do *SuggorateService*, que faz uma varredura do registro DF (Ver Seção 3.4.1) a procura de serviços substitutos.

4.1.2.1.
Raciocínio Baseado em Regras

Checkland e Scholes (CHECKLAND e SCHOLES, 1990) descrevem o raciocínio baseado em regras como uma estrutura de regras organizada que guiam as ações de modo a tentar gerenciar os problemas do mundo real. Este conjunto de regras normalmente segue um ciclo que consiste basicamente de dois

PUC-Rio - Certificação Digital Nº 1012657/CA

componentes: uma base de conhecimento e uma máquina de inferência. Deste modo, o raciocínio baseado em regras (RBR) procura resolver o problema atual baseado na reutilização de uma solução utilizada num problema passado. Para tal, a máquina de inferência procura na base de conhecimento um problema que seja igual ao problema atual. A base de conhecimento é formada de registros compostos de pares problema-solução. Assim, ao encontrar um problema passado igual ao atual tem-se a solução do problema atual senão, o problema não tem solução.

Um tipo particular para implementar o raciocínio baseado em regras é utilizando os comandos "*if-then*". Conforme mencionado acima, o RBR é formado pelas regras, que são um conjunto de padrões de problemas; um mecanismo de inferência que procura um problema no conjunto de padrões de problemas; e a solução daquele problema. Portanto, o *if* infere se determinado problema é condizente com o primeiro problema do conjunto senão procura no item seguinte do conjunto, caso o problema seja identificado então a ação é executada pelo comando *then*. A seguir temos uma simplificação do funcionamento do mecanismo de raciocínio baseado em regras onde é possível detectar problemas como: falha de colapso[2] (Regra 1), falha com parada segura (Regra 2) e falha de temporização (Regra 3).

Regra 1 : If "*serviceFailure*" Then problem = "*serviceFail*";

Regra 2 : If "*serviceUnavailable*" Then problem = "*serviceUnachievable*";

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

Portanto, ao receber a mensagem da classe *Monitor*, a primeira ação é verificar se a mensagem contém algum indício de falha de comunicação, utilizando o mecanismo de raciocínio baseado em regras. Se nenhuma falha for detectada, o sistema continua sua execução normalmente. Se for identificada alguma falha corresponde à lista de regras, então é executado o mecanismo que elabora a relação de serviços substitutos, executado pela classe *SurrogateServices*,

[2] Segundo Birman (BIRMAN, 1996), falha de colapso – ocorre quando o sistema simplesmente para sua execução sem tomar ações indevidas; falha com parada segura – similar à falha anterior, porém a falha é passível de detecção e falha de temporização – ocorre quando o tempo de resposta é excessivamente longo.

PUC-Rio - Certificação Digital Nº 1012657/CA

que será repassada à atividade *Plan*. Este mecanismo é bem simples, o qual define a lista de serviços candidatos a partir da leitura do registro DF dos serviços ativos.

Ao final da execução do *Analyze*, teremos falha identificada uma lista de possíveis serviços substitutos. O próximo passo é a execução da atividade *Plan*, sendo esta responsável por selecionar o novo serviço e identificar se haverá necessidade de tradução das mensagens, configurando assim a abordagem da interoperabilidade na comunicação entre o sistema e o serviço web.

4.1.3. Plan

A atividade *Plan*, apresentada no diagrama de classes da Figura 4.5, é responsável por eleger o novo serviço, a partir da lista de serviços candidatos elaborada pela atividade *Analyze,* e verificar a necessidade de um tradutor de mensagens para a troca de informações entre o agente e o novo serviço web.

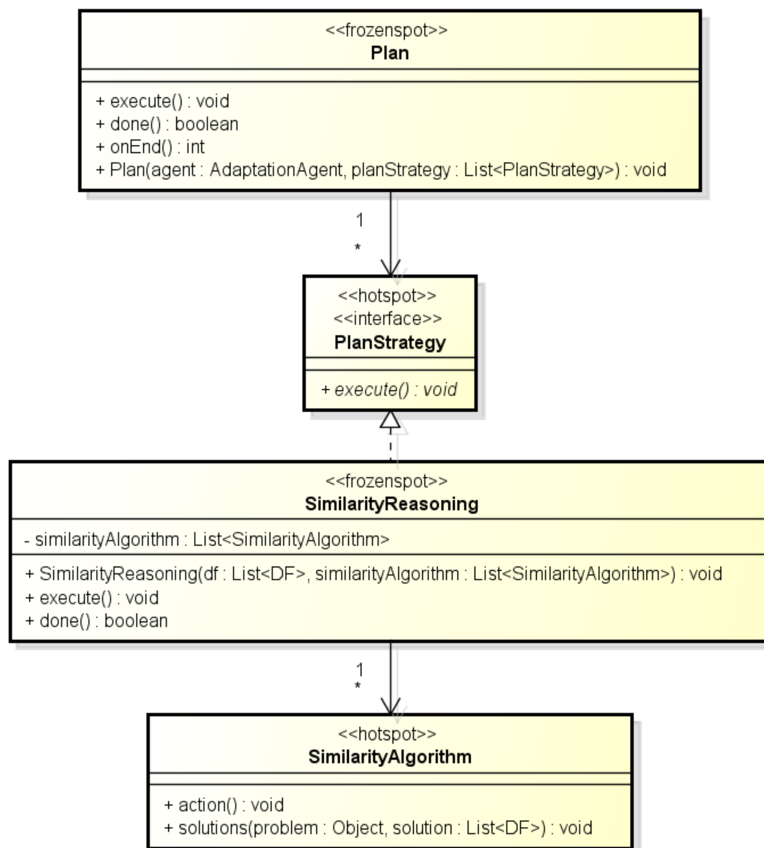

Figura 4.5 – Atividade *Plan*.

Dentre suas responsabilidades, a classe *Plan* é responsável por gerenciar as estratégias de similaridades, representados pela interface *PlanStrategy*, onde o

PUC-Rio - Certificação Digital Nº 1012657/CA

método *execute* deve ser implementado definindo a lógica de execução dos algoritmos de similaridade. A classe *SimilarityReasoning* é que executa o cálculo de similaridade entre o serviço web que falhou e os serviços candidatos. Esta classe recebe como parâmetros a lista de serviços candidatos, vinda da atividade *Analyze*, a lista de algoritmos de similaridades que devem ser executados. Para o algoritmo de similaridade, representado pela classe *SimilarityAlgorithm*, devem ser passados como parâmetros os dados do problema atual e a lista de serviços candidatos. O método *solutions* deverá conter também o algoritmo que fará o cálculo de similaridade entre os parâmetros repassados.

4.1.3.1. Similaridade entre Serviços Web

Um modo natural de avaliar similaridade entre termos é calcular a distância entre os nós correspondentes dos itens que estão sendo comparados – quanto menor for o caminho entre os nós, mais similares serão os itens comparados. No caso de múltiplos caminhos, considera-se o de menor comprimento (LEE, KIM e LEE, 1993).

Deste modo, os parâmetros necessários para o cálculo de similaridade são os atributos do problema e a lista de serviços candidatos a substitutos. Os atributos do problema foram definidos na classe *Collector* através do método *sensor* e formatados pelo método *translator*. A lista de serviços candidatos foi elaborada pela classe *Analyze*, no método *SurrogateServices*. A partir destas informações é possível calcular o nível de similaridade entre o serviço falho e os candidatos.

Para realizar tal cálculo, foi introduzido ao *framework* o algoritmo elaborado por Resnik (Resnik, 1999), em que para sua execução, devem-se passar dois termos para que sua similaridade seja calculada. A similaridade é calculada retornando um valor entre 0 e 1, em que, quanto maior a semelhança dos termos, mais próximo de 1 é o valor retornado. Se os termos não forem similares, então o retorno é zero. Por exemplo, considere um serviço web que tem o tributo tipo do serviço o termo *Mercado* e outro que possui o termo *Supermercado*. A Figura 4.6 apresenta o grau de similaridade entre estes termos, em que nota-se a existência de um antecessor comum.

PUC-Rio - Certificação Digital Nº 1012657/CA

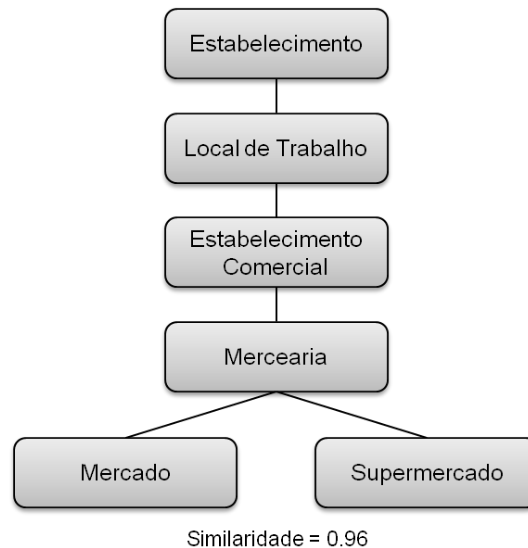

Figura 4.6 – Fragmento da árvore de similaridade.

Para se obter um melhor entendimento do funcionamento do mecanismo de similaridade, considere que para um determinado serviço, foram selecionados os seguintes atributos para serem monitorados: serviços (nome e tipo de serviço), ontologia e conteúdo (Data de Partida, Data de Chegada, Cabine). Considere a comparação entre o serviço falho, representado por Problema 1 (Figura 4.7) e o serviço candidato a substitutos, Caso 1 (Figura 4.8).

<u>Problema 1</u>

Serviços: O serviço contêm os parâmetros: nome do serviço (Agência de Viagens) e tipo de serviço (Pacote Cruzeiro Marítimo 10 dias);

Ontologia: Comprar Pacote;

Conteúdo: As informações solicitadas são: data de partida, data de chegada e cabine (refere-se à posição do quarto).

<u>Caso 1</u>

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

Ontologia: Comprar Pacote;

Conteúdo: As informações repassadas são: data de partida, data de chegada e cabine (refere-se à posição do quarto) e valor do pacote.

PUC-Rio - Certificação Digital Nº 1012657/CA

```
(Request
      : sender "sisAgent"
      : receiver "ageViagens"
      : ontology "Comprar Pacote"
      : content "Data de Partida, Data de Chegada, Cabine"
      : services (
                  : name "Agencia de Viagens"
                  : type "Pacote Cruzeiro Marítimo 10 dias")
)
```

Figura 4.7 – Ilustração de solicitação via padrão FIPA ACL.

```
(Inform
      : sender "ageTur"
      : receiver "sisAgent"
      : ontology "Comprar Pacote"
      : content "Data de Partida, Data de Chegada, Cabine, Valor"
      : services (
                  : name "Agencia de Viagens"
                  : type "Pacote Cruzeiro Marítimo")
)
```

Figura 4.8 – Ilustração de resposta via padrão FIPA ACL.

A Figura 4.9 apresenta o resultado do cálculo da similaridade entre o Problema 1 e o Caso 1. Nota-se que a coluna Problema 1 representa os atributos do serviço que falhou, a coluna Caso 1 representa os atributos que foram definidos para serem monitorados pela classe *Monitor*. A coluna similaridade representa o cálculo da similaridade, em que cada termo de todos os atributos são comparados e desta forma similaridade entre o tipo de serviço não recebeu a classificação zero. No caso do atributo valor, que está presente em um serviço candidato e não é solicitado pelo cliente, o resultado da semelhança é zero.

PUC-Rio - Certificação Digital Nº 1012657/CA

Atributos		Problema 1	Caso 1	Similaridade
Serviço	Nome	Agência de Viagens	Agência de Viagens	1
	Tipo	Pacote Cruzeiro Marítimo 10 dias	Pacote Cruzeiro Marítimo	0.6
Ontologia		Comprar Pacote	Comprar Pacote	1
Conteúdo		Data de Partida	Data de Partida	1
		Data de Chegada	Data de Chegada	1
		Cabine	Cabine	1
			Valor	0
Similaridade Total				0.8

Figura 4.9 – Similaridade entre o Problema 1 e o Caso 1.

Após a seleção do novo serviço web, é possível identificar pelos seus atributos qual o tipo de provedor daquele serviço e assim verificar a necessidade de um tradutor de mensagens para sanar a questão da interoperabilidade na comunicação entre o sistema e o serviço web. Este trabalho foi elaborado utilizando o conceito de sistema multiagentes e, portanto caso seja necessário realizar troca de informações com serviços que utilizam o padrão W3C então será necessário a tradução das informações. Se o novo serviço selecionado utilizar o padrão de agentes, então não será necessária a tradução de informações.

Ao final da execução do *Plan*, teremos um novo serviço selecionado para substituir o serviço falho. O próximo passo é a execução da atividade *Execute*, sendo esta responsável realizar as configurações necessárias ao sistema para que este possa se comunicar com o novo serviço web.

PUC-Rio - Certificação Digital Nº 1012657/CA

4.1.4. Execute

A atividade *Execute*, apresentada no diagrama de classes da Figura 4.10, é o último passo do loop de autoadaptação do sistema. Ela é responsável por realizar as configurações necessárias ao sistema a partir das informações recebidas do *Analyze*.

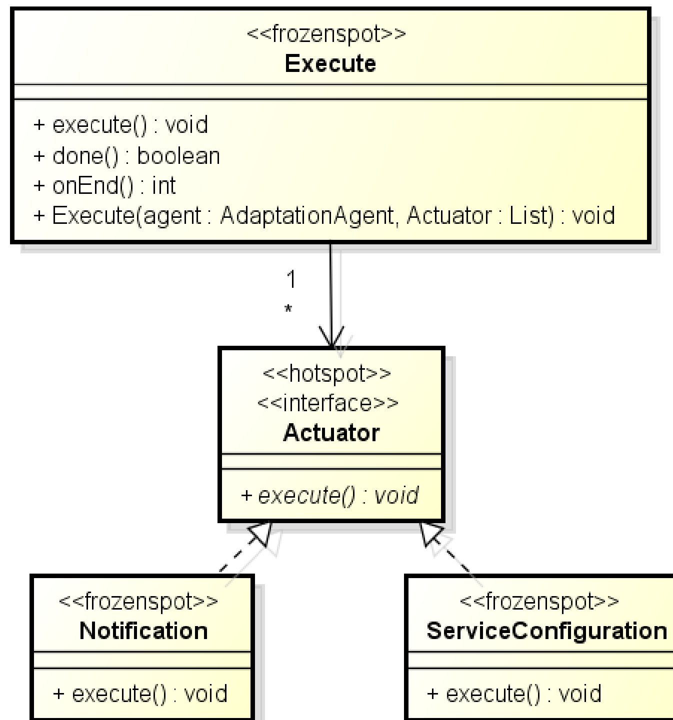

Figura 4.10 – Atividade *Execute*.

Na Figura 4.10 temos o diagrama de classes da atividade *Execute* onde se pode observar que esta é responsável por gerenciar um conjunto de atuadores, representado pela classe *Actuator*. Os atuadores utilizam a classe *Notification* para informar ao agente do sistema o novo serviço selecionado e se necessário, que sua mensagem precisará ser traduzida. A classe *ServiceConfiguration* é utilizada pelo atuadores para realizar as configurações necessária para que o agente tradutor esteja apto a realizar as traduções necessárias ao sistema.

PUC-Rio - Certificação Digital Nº 1012657/CA

4.1.4.1. Mecanismo para Tradução de Mensagens

Esta atividade é executada pelo agente *JadeGatewayAgent*, que foi incorporado a este trabalho através do *framework* WSIG elaborado pela Telecom Italia SpA (JADE, 2011). Este agente oferece suporte à questão da interoperabilidade, em que se percebe a necessidade da tradução das informações trocadas entre o sistema e o serviço web. O agente *JadeGatewayAgent* é capaz de traduzir mensagens ACL em SOAP e vice-versa. A Figura 4.11 apresenta uma ilustração da atividade executada pelo *JadeGatewayAgent*.

A classe *Notification* é responsável por informar ao agente do sistema a necessidade de tradução das mensagens, e a partir dessa notificação, todas as mensagens enviadas pelo agente do sistema serão direcionadas ao *JadeGatewayAgent*. A Classe *ServiceConfiguration* informará ao *JadeGatewayAgent* o serviço que proverá as informações. O agente *JadeGatewayAgent* traduzirá a mensagem do formato ACL para SOAP e enviará ao serviço web. Este mesmo agente receberá o retorno no formato SOAP, decodificará para ACL e encaminhará ao agente do sistema.

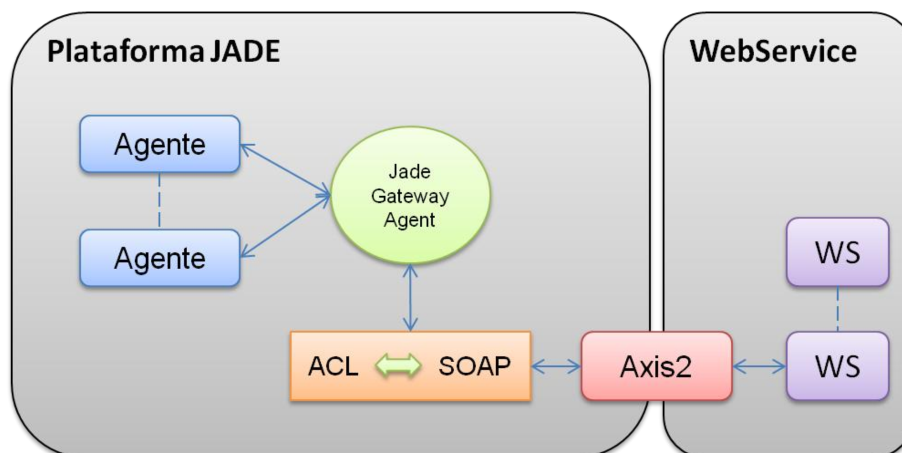

Figura 4.11 - Ilustração da atividade do agente *JadeGatewayAgent.*

Para obter um melhor entendimento do funcionamento do mecanismo de tradução de mensagens, considere que foram selecionados os seguintes atributos para serem monitorados: serviços (nome e tipo de serviço), ontologia e conteúdo (declividade e vegetação). Considere que o novo serviço selecionado foi implementado utilizando o padrão da W3C, com WSDL representado pela Figura 4.14. Neste caso, o *JadeGatewayAgent* entrará em atividade para realizar a tradução das mensagens enviadas pelo agente ao serviços web, do formato ACL para SOAP (Figura 4.12), e também realizará a tradução da resposta serviço web para o padrão do sistema, do formato SOAP para ACL (Figura 4.13).

PUC-Rio - Certificação Digital Nº 1012657/CA

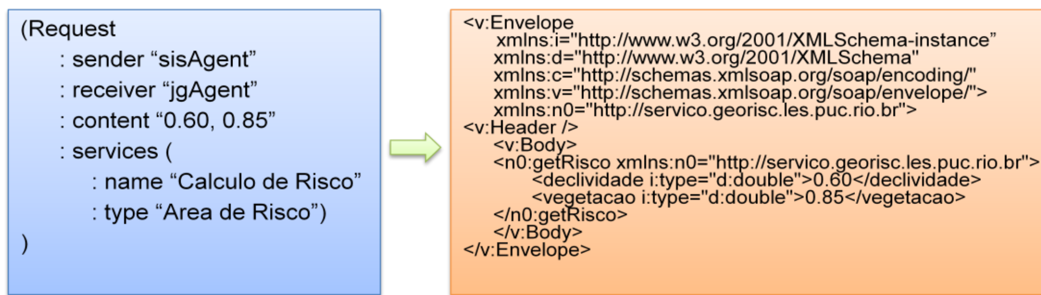

Figura 4.12 – Ilustração da tradução de uma mensagem ACL para SOAP.

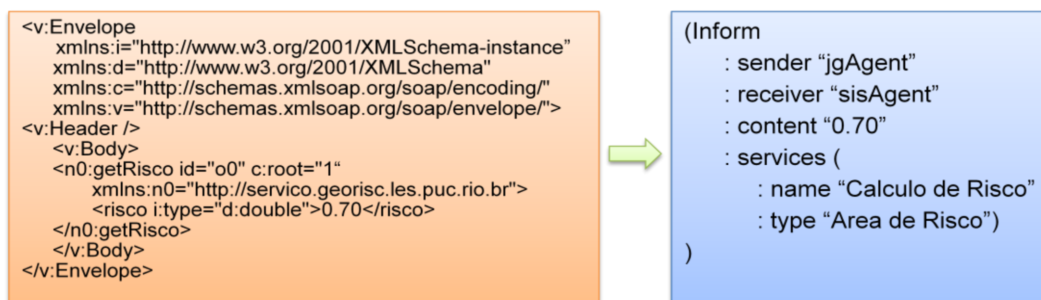

Figura 4.13 – Ilustração da tradução de uma mensagem SOAP para ACL.

Ao final da desta atividade, tem-se o sistema adaptado a se comunicar com o novo serviço. Caso seja necessária a tradução de informações, nota-se que dois agentes estarão operando, o agente que representa o sistema em si e o agente responsável por realizar as traduções das mensagens, neste caso o agente *JadeGatewayAgent*.

Deste modo, a união dos conceitos de sistemas multiagentes e computação autonômica, permitiu a elaboração de um *framework* que ofereça suporte ao desenvolvimento de aplicação para dispositivos móveis capazes de superar as questões de disponibilidade e interoperabilidade da comunicação entre estes dispositivos e os serviços web, em tempo de execução. Aplicações estendidas deste *framework* estão aptas a se reconfigurar e se readaptar caso o novo serviço selecionado utilize um protocolo de comunicação, descrição e dados diferente do protocolo utilizado pelo serviço anterior.

PUC-Rio - Certificação Digital Nº 1012657/CA

```
- <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: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 4.14 – WSDL do novo serviço.

PUC-Rio - Certificação Digital Nº 1012657/CA

4.2. Pontos Fixos

Markiewicz e Lucena (MARKIEWICZ e LUCENA, 2001) descrevem pontos fixos como pontos imutáveis presentes no núcleo do *framework*, são porções de código já implementadas no *framework* que podem ser utilizadas para invocar os pontos flexíveis. Estes pontos fixos estarão presentes em todas as instâncias do *framework*. A seguir têm-se os pontos fixos do *framework* proposto e na Figura 4.15 os pontos fixos são representados pelo estereótipo "*frozenspot*".

- A lógica de execução do *control loop* utilizado como guia no ciclo de autoadaptação do sistema apresentado na Figura 4.1 é implementado através da classe *AdaptationControlLoop* e não pode ser modificado;

- Os comportamentos *Monitor, Analyze, Plan e Execute*;

- O mecanismo de elaboração da lista de serviços substitutos, representado pela classe *SurrogateService*;

- As atividades executas pelo comportamento *Execute*, *Notification* e *ServiceConfiguration* também não podem ser estendidas;

- O mecanismo de tradução de mensagens que é realizado pelo agente *JadeGatewasAgent*.

4.3. Pontos Flexíveis

Markiewicz e Lucena (MARKIEWICZ e LUCENA, 2001) descrevem pontos flexíveis como classes ou métodos abstratos que devem ser implementados, em que cada instância do *framework* pode conter implementação diferente para os pontos flexíveis. A seguir têm-se os pontos flexíveis do *framework* proposto e na Figura 4.15 os pontos flexíveis são representados pelo estereótipo "*hotspot*".

- Podem ser definidos novos mecanismos para monitoramento das mensagens e também de serviços web estendendo a classe abstrata *Collector*;

- A classe *RuleReasoning* permite sua extensão para definição de novas regras e bases de raciocínio;

PUC-Rio - Certificação Digital Nº 1012657/CA

- É possível a utilização de outros algoritmos para cálculo de similaridade dos atributos dos serviços web, que dever ser implementado através da classe *SimilarityAlgorithm*;

- Novos mecanismos para reconfiguração do sistema podem ser construídos através da interface *Actuator*.

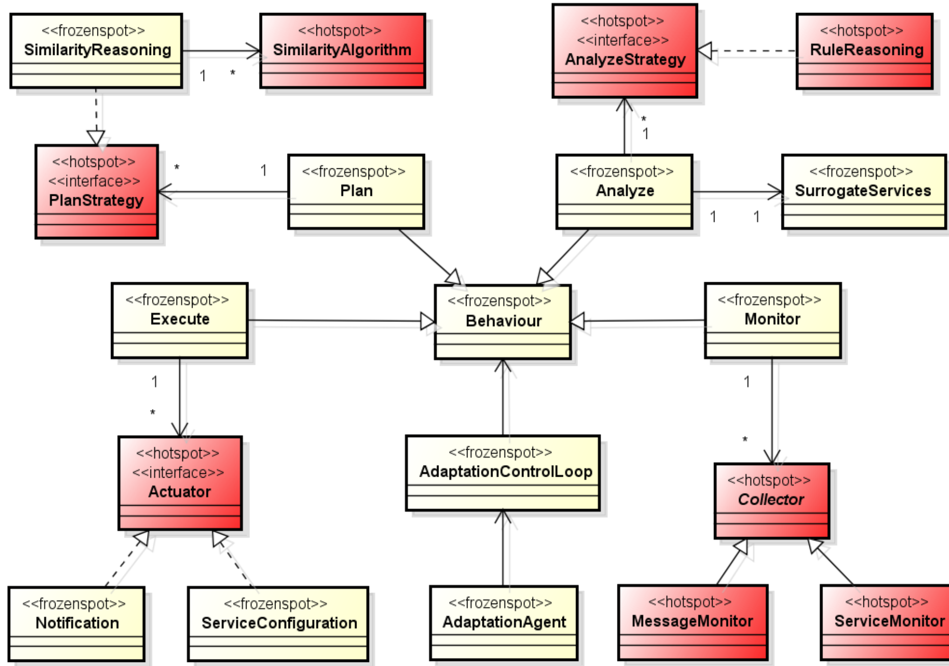

Figura 4.15 – Ilustração dos pontos fixos e flexíveis do *framework*.

PUC-Rio - Certificação Digital Nº 1012657/CA