UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RICARDO RODRIGUES LUCCA

Jason com serviços Web: agentes e ambientes heterogêneos

Trabalho Individual I
TI-123

Prof. Dr. Rafael Heitor Bordini
Orientador

Porto Alegre, janeiro de 2011

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS 3

LISTA DE FIGURAS 4

LISTA DE LISTAGENS 5

RESUMO 6

1 INTRODUÇÃO 7

2 REVISÃO BIBLIOGRÁFICA 9
2.1 Jason 9
2.1.1 Visão Geral 9
2.1.2 Infra-estrutura do Jason 11
2.2 Protocolos de serviços web 12
2.2.1 XML-RPC 13
2.2.2 SOAP 15
2.3 Trabalhos relacionados 17

3 TRABALHO DESENVOLVIDO 19
3.1 Servidor de Serviços Web 19
3.2 Cliente de Serviços Web 22

4 EXEMPLO DE USO 24
4.1 Exemplo *Room* 24
4.2 Exemplo *Game of Life* 27

5 CONCLUSÃO 30

REFERÊNCIAS 31

APÊNDICE A FONTE DO CLIENTE WS NA PLATAFORMA JASON . . 32

APÊNDICE B FONTE DO EXEMPLO *ROOM* 42

APÊNDICE C FONTE DO EXEMPLO *GAME OF LIFE* 48

LISTA DE ABREVIATURAS E SIGLAS

BDI	Beliefs−Desires−Intentions
FIPA	Foundation for Intelligent Physical Agents
MAS	Multi−Agent System
RPC	Remote Process Call
SOAP	Simple Object Access Protocol
WS	Web Service
WSS	Web Service Server
XML	eXtensible Markup Language
XML-RPC	Remote Process Call using XML

LISTA DE FIGURAS

Figura 2.1: Modelo da infra-estrutura do Jason. 12
Figura 2.2: Exemplo de requisição HTTP, o ator (o solicitante) na esquerda e a direita o servidor que responde (o atendente). 13

Figura 3.1: Diagrama relacionando o ciclo do agente Jason (esquerda) com os métodos à serem implementados em serviços (direita). 19
Figura 3.2: Diagrama da arquitetura do agente desenvolvido. 22
Figura 3.3: Diagrama do ambiente desenvolvido. 23

LISTA DE LISTAGENS

2.1 Arquivo de projeto do Jason para o exemplo *Room* 9
2.2 Agentes em ASL 10
2.3 Exemplo de chamada XML-RPC com os cabeçalhos HTTP sendo omitidos (CERAMI; LAURENT, 2002). 13
2.4 Exemplo de resposta XML-RPC com os cabeçalhos HTTP sendo omitidos (CERAMI; LAURENT, 2002). 14
2.5 Exemplo de chamada SOAP com os cabeçalhos HTTP sendo omitidos (CERAMI; LAURENT, 2002). 16
2.6 Exemplo de chamada SOAP com os cabeçalhos HTTP sendo omitidos (CERAMI; LAURENT, 2002). 17
4.1 Arquivo de projeto do Jason adaptado. 24
4.2 Método de percepção e ação do agente *porter*. 25
4.3 Métodos relacionados com a comunicação do agente *porter*. 25
4.4 Métodos do agente *paranoid*. 26
4.5 Arquivo de projeto do *Game-of-Life*. 27
4.6 Tipos usados pelo ambiente. 27
4.7 Métodos relativos às percepções. 28
4.8 Métodos para recuperar as percepções. 29
4.9 Método que realiza ações. 29

RESUMO

O presente trabalho visa aumentar a integração da plataforma Jason (BORDINI et al., 2004) com programas não desenvolvidos em linguagem Java. A finalidade é permitir que outras linguagens sejam utilizadas quando for vantajoso. Com essa finalidade, as principais abordagens em serviços Web (WS) foram estudadas (CERAMI; LAURENT, 2002; ALLMAN, 2003; KOHLHOFF; STEELE, 2003). As duas abordagens mais conhecidas são baseadas na tecnologia XML: XML-RPC e SOAP. Logo, um serviço Web será desenvolvido para tornar o ambiente e os agentes da plataforma Jason capazes de serem implementados em diferentes linguagens. Além disso, um levantamento teórico da área será realizado explicando os protocolos mencionados e trabalhos relacionados na área (PIUNTI; RICCI; SANTI, 2009; BELLIFEMINE; POGGI; RIMASSA, 1999).

Palavras-chave: Sistemas multi-agente, agentes cognitivos, serviços web, XML-RPC.

1 INTRODUÇÃO

Uma das arquiteturas mais conhecidas para o desenvolvimento de agentes cognitivos é a arquitetura BDI, onde (DASTANI; STEUNEBRINK, 2009; SILVA; PADGHAM, 2005) são dois trabalhos que utilizam-na. A plataforma Jason (BORDINI et al., 2004), baseada na arquitetura BDI, utiliza uma extensão da linguagem de programação abstrata (RAO, 1996), chamada AgentSpeak(L), para especificar as decisões dos agentes. Enquanto, os ambientes são unicamente códigos Java que, por muitas, vezes demostram graficamente os agentes atuando.

No processo de deliberação, as ações encontram-se agrupadas em um conjunto chamado de plano e cada plano possui um conjunto de ações para verificar o contexto de sua execução (DASTANI; STEUNEBRINK, 2009). O contexto serve para determinar os planos válidos para executar no presente momento. Em caso de haver mais de um plano válido, o primeiro será escolhido.

Uma decisão de um agente é a escolha feita para se realizar determinada atividade. Por exemplo: o agente X deseja empurrar o agente Y. Essa atividade ou ação de empurrar pode ser implementada de diferentes formas. Na primeira, o agente X encaminha uma mensagem para o agente Y dizendo que ele esta sendo empurrado. Na segunda, o agente X envia uma mensagem para o ambiente dizendo que esta empurrando o outro agente. Ambas implementações são possíveis em Jason e utilizam diferentes mecanismos. O que deve ficar claro aqui é a importância das decisões de projeto e, também, de se manter a coesão em projetos de MAS.

A motivação desse trabalho é permitir que a plataforma Jason se integre com outras linguagens que podem não ser do mesmo paradigma. Essa integração seria possível usando a interface de funções nativas do Java, porém essa solução deixaria nosso código com um nível de acoplamento elevado e restrito a um conjunto limitado de linguagens. Assim sendo, o uso de WS surge naturalmente para solucionar esses dois problemas. As principais abordagens usadas em WS utilizam XML-RPC e SOAP (CERAMI; LAURENT, 2002). Padrões mais novos existem, porém não com bibliotecas prontas para o conjunto de linguagens escolhidas ou não foi considerado um padrão tão conhecido.

Para o trabalho desenvolvido atingir o maior número de linguagens possíveis, um levantamento em linguagens imperativas (C e Java), funcionais (Haskell e Lisp) e de script (Lua, PHP e Python) foi realizado. Todas essas linguagens possuem ou suporte nativo ou suporte através de bibliotecas que podem ser instaladas para ambos os protocolos. Por exemplo: PHP tem suporte nativo aos dois protocolos; Java tem suporte nativo ao SOAP; Python tem suporte nativo ao XML-RPC. Dessa forma, ambos os protocolos foram considerados equivalentes ficando a decisão de qual utilizar baseando-se no detalhamento dos mesmos.

No capítulo 2, o estudo detalhado dos protocolos, a plataforma Jason e abordagens que

utilizam o conceito de serviços web para integrar sistemas são discutidas. No capítulo 3, os critérios utilizados na decisão e o projeto de implementação podem ser encontrados. No capítulo 4 é mostrado um passo-a-passo da montagem do ambiente e duas implementações que utilizam o desenvolvimento. Por fim, no capítulo 5 é debatido a experiência e os trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

O presente capítulo apresenta a revisão bibliográfica realizada, apresentando na seção 2.1 a plataforma Jason em sua totalidade. Na seção 2.2 são discutidos os protocolos de serviços web considerados mais relevantes. Na seção 2.3 são apresentados trabalhos relacionados com o que esta sendo proposto, discutindo suas diferenças com o presente.

2.1 Jason

A plataforma Jason (BORDINI et al., 2004), baseada na arquitetura BDI, utiliza uma extensão da linguagem de programação abstrata (RAO, 1996), chamada AgentSpeak(L), para especificar as decisões dos agentes. Na seção 2.1.1 é apresentado uma visão geral do Jason. Na seção 2.1.2 é debatido alguns detalhes da engenharia do Jason.

2.1.1 Visão Geral

Para explicar de forma mais didática será utilizado o exemplo *Room* que acompanha o Jason. Para isso será introduzido o arquivo de projeto do exemplo e, a partir desse, os demais arquivos. A Listagem 2.1 contém o arquivo de projeto. Na linha 2, *room* é o nome do projeto e, por isso, pode ser qualquer identificador. A partir da linha 3 os valores antes dos dois-pontos (:) são palavras reservadas que o Jason entende para diferentes propósitos e o valor a seguir (depois dos dois-pontos) é o valor associado. Assim, *infrastructure*, na linha 3, pode assumir três valores possíveis: (i) *Centralised*, normalmente utilizada; (ii) *jade*, utilizada quando se deseja integrar com agentes não Jason (jade); (iii) *saci*, utilizada quando deseja executar os agentes de maneira distribuída na rede.

Listagem 2.1: Arquivo de projeto do Jason para o exemplo *Room*

```
1 // Isso eh um comentario
  MAS room {
3   infrastructure: Centralised
    environment: RoomEnv
5   executionControl: jason.control.ExecutionControl
    agents: porter; claustrophobe; paranoid;
7 }
```

Continuando na Listagem 2.1, a entrada *environment* configura a classe de ambiente que será utilizada. A próxima entrada, normalmente não aparece, é a *executionControl* utilizada para mudar a forma com que os agentes são executados. O valor no exemplo é uma classe que obriga o próximo ciclo de deliberação somente acontecer quando todos os agentes terminaram o seu ciclo. O valor padrão é iniciar um novo ciclo de deliberação após 500ms do ciclo de deliberação anterior ter sido concluído, porém, algumas vezes

isso pode vir a apresentar problemas de sincronismo e, por isso, é interessante mostrar que há uma opção para controlar a forma de execução dos ciclos deliberativos. O usuário, inclusive, pode colocar sua própria classe como configuração.

Todas as entradas apresentadas até agora mapeiam para código Java. Os agentes são especificados na entrada *agents*. Como pode-se observar na Listagem 2.1, cada referência a um agente deve terminar com um ponto-e-vírgula (;). Há ainda como especializar cada um dos agentes mudando opções. No capítulo 4 será mostrado um exemplo dessas opções.

Antes de entrar em discussão sobre os agentes será explicado o exemplo sendo utilizado. No exemplo tem-se uma porta na sala e duas pessoas, uma claustrofóbica e outra paranoica. A pessoa claustrofóbica deseja que a porta da sala esteja aberta, enquanto que a paranoica deseja que a porta fique fechada. Assim, os agentes Jason são desenvolvidos em arquivos texto com extensão *ASL* visando montar esse modelo.

Logo, há três agentes na linha 6 da Listagem 2.1: (i) *porter* é o agente responsável pela porta e o único que conhece como abri-lá ou fecha-lá; (ii) *claustrophobe* é o agente que deseja deixar a porta sempre aberta; (iii) *paranoid* é o agente que deseja deixar a porta sempre fechada. A implementação desses agentes torna-se muito simples com o uso de eventos.

Esses eventos podem ser de adição (+) ou de remoção (-) de crenças, metas ou consultas. Todas as estruturas são semelhantes a chamada de função. No exemplo "telefone(808080822)", telefone pode ser uma crença ou uma ação de ambiente que pode ser executada dependendo de onde a entrada se localiza. Note que, se essa entrada for precedida pelo sinal de exclamação (!) ou de interrogação (?) então o significado é alterado respectivamente para uma meta ou uma consulta. Ainda há a possibilidade de preceder uma crença ou meta com sinal de adição ou de subtração significando ou adicionar/remover uma crença ou estar recebendo/removendo uma crença, meta ou consulta.

Listagem 2.2: Agentes em ASL

```
1  // claustrophobe.asl
   +locked(door) : true
3    <- .send(porter,achieve,~locked(door)).

5  // paranoid.asl
   +~locked(door)
7    <- .send(porter,achieve,locked(door)).

9  // porter.asl
   +!locked(door)[source(paranoid)]
11   : ~locked(door)
     <- lock.
13
   +!~locked(door)[source(claustrophobe)]
15   : locked(door)
     <- unlock.
```

Na Listagem 2.2 tem-se a continuação do exemplo e, por simplicidade, todos os fontes dos agentes encontram-se reunidos. O agente denominado *claustrophobe* será o primeiro a ser detalhado e corresponde as linhas 1 até 3 da Listagem 2.2. A programação em Jason é guiada por reatividade nas crenças e percepções do próprio agente, assim, é necessário uma forma de estruturar as ações à serem decididas. Essa forma é o plano. O plano pode ser ativado quando se deseja adicionar, consultar ou remover [1] uma crença ou meta. O plano da linha 2 até 3 da Listagem 2.2 será detalhado adiante.

Em um plano há sempre três divisões: gatilho do evento, contexto e corpo. A primeira e única à ser explicita é o gatilho do evento, que deve ocorrer para o plano ser disparado. No exemplo ele está limitado do carácter inicial até o dois-pontos (:). A divisão de contexto é onde se coloca as ações, crenças ou regras de inferência que tem que ser válidas para o corpo do plano ser considerado válido. Dessa forma, é importante tomar cuidado no que vai no contexto em razão dele sempre ser executado previamente para definir quais

[1] Uma remoção pode ser o momento de conter uma falhar, porém isso não será abordado.

planos devem ser descartados. A segunda divisão vai até a seta (<-) e não é obrigatória. A terceira divisão, também não é obrigatória, possui todas as ações a serem executadas pelo seu plano. No exemplo o plano tem em seu corpo uma ação interna do Jason denominada *send* que envia determinado dado (terceiro parâmetro) para o agente especificado (primeiro parâmetro) com determinado formato (segundo parâmetro).

Cabe chamar atenção para uma coisa, as ações internas podem ser definidas pelo usuário. Uma ação interna possui o seguinte formato "tcp.send", assim essa ação está definida no pacote *tcp* pela classe *send* que deve especializar a classe *DefaultInternalAction* do Jason. Entretanto, uma ação interna definida pelo Jason não possui indicativo de pacote que é o caso do ".send" presente no código dos agentes.

No exemplo da linha 3 na Listagem 2.2, o agente *claustrophobe* está enviando uma mensagem para o agente denominado *porter* ter a meta (*achieve* no segundo parâmetro) de não ($\sim$) ter a crença da porta estar fechada. Analogamente, o agente denominado *paranoid*, definido na linha 5 à 7, envia como crença ter a porta fechada. Logo, o agente *porter* pode ser entendido em sua quase totalidade na Listagem 2.2. Um entendimento completo do exemplo vem com o aprendizado das anotações que são dados que podem ser guardados juntos das crenças e essas anotações podem ter suas anotações também. No agente *porter* essas anotações são utilizadas somente para dizer que determinado plano só é válido quando tiver como fonte (do inglês *source*) um determinado agente. O presente exemplo possui as anotações, pois assume a hipótese do mundo aberto. Vale observar que, essas anotações podem ser removidas sem nenhum problema adicional, visto que, no mundo proposto o agente *porter* recebe a meta a ser alcançada de um dos dois agentes existentes.

A execução dos agentes acontece de forma arbitrária e deve-se ter em conta que o primeiro agente que irá enviar a mensagem para o agente *porter* dependerá de como o mundo inicia. Logo, se o mundo iniciar com a porta fechada o primeiro a enviar a solicitação será o agente *claustrophobe*. Já se o mundo for iniciado com a porta aberta o primeiro a enviar solicitação será o agente *paranoid*. Além disso, conforme a simulação vai correndo os dois agentes ficam alternando mensagens com o agente *porter* por causa do compartilhamento do mesmo recurso.

2.1.2 Infra-estrutura do Jason

A plataforma Jason tem uma base de software completamente extensível como é possível observar mediante a Figura 2.1. Nessa figura, a infra-estrutura encontra-se representada como um barramento na parte inferior. Ela pode ser configurada através da chave de configuração *infrastructure* no arquivo de projeto (ver Listagem 2.1).

A infraestrutura define todas as classes padrões que o usuário irá utilizar, por exemplo observe que o barramento liga-se a dois adaptadores: um do tipo agente e um de ambiente. Esses adaptadores permitem a compatibilidade entre implementações distintas de classes permitindo que as implementações não padrão variem sem ter que alterar a estrutura do barramento.

Assim, há a possibilidade de informar para uma determinada simulação que um determinado agente utilizará uma arquitetura e/ou um raciocinador diferente dos demais agentes. Para se fazer isso no momento que se declara os agentes no projeto informa-se as chaves que encontram-se dentro das caixas com cantos arqueados na Figura 2.1. O exemplo "fb agArchClass KosMos.FireBrigadeArch agClass KosMos.Agent #1;" define o agente *fb* com a arquitetura usando a classe *FireBrigadeArch* do pacote *KosMos* (pode ser qualquer nome desejado) e utilizando como raciocinador a classe *Agent* do mesmo

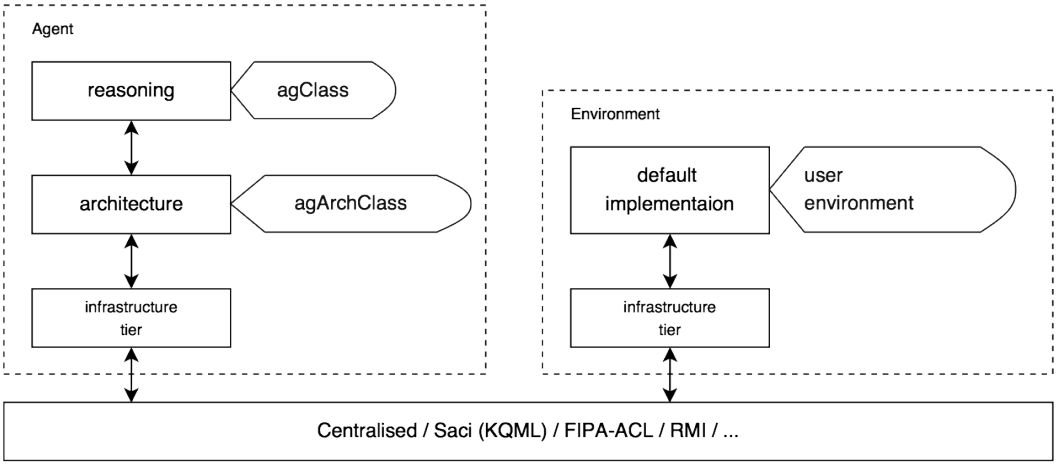

Figura 2.1: Modelo da infra-estrutura do Jason.

pacote. Note que, essas classes não existem no Jason e devem ser providas pelo usuário de alguma forma.

Como já explicado, a entrada *environment* no arquivo de projeto configura a classe de ambiente que será utilizada. Assim, somente um ambiente é permitido por simulação. Um dos motivos comuns de se implementar um ambiente é implementar as ações que o agente poderá realizar. Porém, também é possível alterar como os agentes "visualizam" o ambiente. Um agente pode ser implementado deixando as ações tanto em uma especialização da infraestrutura quanto direto no ambiente. Por ser o ambiente, o responsável pelas percepções então seria nele que a implementação de falhas em sensores ocorreria. Essas falhas podem ser implementadas através da remoção ou alteração das percepções. O agente, nesse ambiente, deveria ser capaz de aplicar regras para verificar essas falhas e assumir que há erros em seus dados.

2.2 Protocolos de serviços web

Os protocolos de serviços web são todos baseados no conceito pergunta-resposta. Esse conceito é a base da internet hoje através de inúmeros protocolos de transferência de dados. Dentre esses protocolos, o de maior destaque para o presente trabalho é o protocolo HTTP (*Hyper Text Transfer Protocol*) [2].

A comunicação do protocolo HTTP considera que sempre há duas entidades: o solicitante e o atendente. O solicitante envia algum dos comandos possíveis ao atendente que é responsável por interpretar o comando enviado e responde-lo adequadamente. Essa comunicação é ilustrada na Figura 2.2 que mostra uma solicitação de uma página da web através de um dos comandos possíveis no protocolo HTTP.

A troca de informações, no protocolo HTTP, é realizada sempre via texto puro. Assim, a presente seção explica os protocolos de serviços web que são os mais conhecidos baseados na linguagem XML [3] que podem ser utilizados sobre esse protocolo. Na subseção 2.2.1 introduz-se o protocolo XML-RPC e na subseção 2.2.2 debate-se o SOAP.

[2] Veja a RFC-1945 e RFC-2616 para maiores detalhes.

[3] Abordar a linguagem XML esta fora de escopo, visite http://www.w3.org/TR/xml/ para mais detalhes.

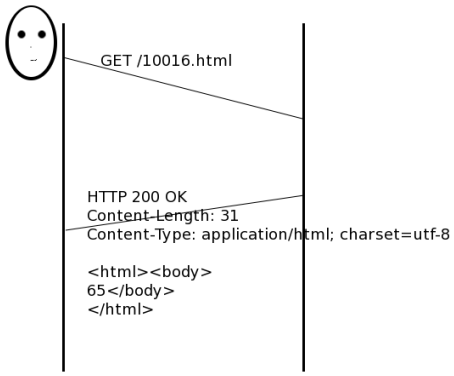

Figura 2.2: Exemplo de requisição HTTP, o ator (o solicitante) na esquerda e a direita o servidor que responde (o atendente).

2.2.1 XML-RPC

O XML-RPC surgiu em meados de 1998 visando permitir que funções ou procedimentos fossem feitos através de rede sobre o protocolo HTTP (CERAMI; LAURENT, 2002). Dessa forma, um programa cliente pode passar informações à um dado servidor com uma pequena descrição de sua solicitação e o servidor responde com uma falta ou com uma mensagem descrevendo o retorno propriamente dito. Uma falta é equivalente a um erro ou exceção podendo ter caráter temporário (base de dados sobrecarregada) ou permanente (requisição sendo feita é inválida).

O protocolo XML-RPC define 8 tipos de dados que podem ser utilizados tanto nos valores de parâmetros, quanto de retornos ou de faltas. Esses tipos encontram-se representados na Tabela 2.1. Dentre esses tipos de dados, os mais complexos são "<struct>" e "<array>". Nestes, ambos, admitem em sua valoração qualquer dos tipos de dados possíveis (inclusive seu próprio). Sobre o "<array>" não há uma obrigação de utilizar em sua valoração o mesmo tipo em todos os seus elementos. Toda a informação trocada é textual e deve estar claro que é responsabilidade do solicitante e/ou do servidor saber traduzir da sua forma de armazenamento para texto ou vice-versa.

Listagem 2.3: Exemplo de chamada XML-RPC com os cabeçalhos HTTP sendo omitidos (CERAMI; LAURENT, 2002).

```
<?xml version="1.0" encoding="UTF-8" ?>
<methodCall>
 <methodName>getWeather</methodName>
 <params>
  <param>
   <value><string>10016</string></value>
  </param>
 </params>
</methodCall>
```

As descrições que podem ser utilizadas tanto pelo solicitante quanto pelo servidor são fundamentais. O protocolo descreve duas estruturas primordiais relativas aos métodos. A primeira delas é a chamada de métodos que inicia com "<methodCall>" devendo conter um "<methodName>" que conterá o nome do procedimento sendo realizado. Quando o procedimento contiver parâmetros, eles devem estar contidos em "<params>" e para cada parâmetro um "<param>" contendo "<value>" deve ser informado. O valor do parâmetro pode ser qualquer um dos 8 tipos de dados possíveis (ver Tabela 2.1). Um exemplo dessa estruturação pode ser vista na Listagem 2.3.

Já, a segunda estrutura primordial é a de resposta do método e é iniciada com "<metho-

Tabela 2.1: Tipos de dados no XML-RPC.

Etiqueta	Descrição	Exemplo
<i4> ou <int>	Inteiro sinalizado de 4 bytes.	<i4>-21</i4>
<boolean>	0 (falso) ou 1 (verdadeiro).	<boolean>1</boolean>
<string>	Texto.	<string>Ricardo Lucca</string>
<double>	Valor de ponto flutuante com precisão dupla.	<double>-21.145</double>
<dateTime.iso8601>	Data e Tempo.	<dateTime.iso8601> 20100924T12:26:35 </dateTime.iso8601>
<base64>	Texto codificado em base 64.	<base64> bWUgYXByb3ZhcmVpPw== </base64>
<struct>	Estrutura com membros e cada membro com nome e um valor associado.	<struct> <member> <name>weather</name> <value><int>65</int></value> </member> ... </struct>
<array>	Vetor com valores.	<array><data> <value><boolean>0</boolean></value> <value><int>65</int></value> ... </data></array>

dResponse>". Os valores retornados pelo método são análogos aos parâmetros de chamada, isto é, devem estar em "<params>" e para cada valor retornado um "<param>" contendo "<value>" deve existir com um tipo de dado. Um exemplo de resposta pode ser visto na Listagem 2.4.

Um método, também pode retornar uma falta. Para isso o elemento raiz deve ser "<methodResponse>" por ser uma resposta de método e ao invés de "<params>" no interior será utilizado o elemento "<fault>". Assim, normalmente, quando a implementação deseja devolver detalhes do erro, uma estrutura ou vetor para conter maiores detalhes sobre o mesmo pode ser utilizada.

Listagem 2.4: Exemplo de resposta XML-RPC com os cabeçalhos HTTP sendo omitidos (CERAMI; LAURENT, 2002).

```
<?xml version="1.0" encoding="UTF-8" ?>
<methodResponse>
 <params>
  <param>
   <value><int>65</int></value>
  </param>
 </params>
</methodResponse>
```

Em (ALLMAN, 2003), uma analise do uso do XML-RPC tanto do ponto de tráfego de rede quanto de complexidade de código foi realizada. O estudo mostrou que para um pequeno volume de dados a ser transmitido e recebido, onde há pouca interação entre os participantes, o uso do XML-RPC seria equivalente à utilização de comunicação via *sockets*. Entretanto, em cenários onde o volume de dado é maior se torna bastante desparelho a utilização de XML-RPC contra *sockets*. Os dados comunicados via *sockets* podem ser em formato binário com a menor redundância possível, porém ao utilizar XML-RPC uma

representação textual do dado é necessária e, sempre, mesmo que pequeno, haverá redundância na informação de marcação.

2.2.2 SOAP

O SOAP é o nome de um protocolo análogo ao XML-RPC. Ele surgiu em março do ano 2000, porém não descreve em nenhum momento como objetos devem estar definidos para serem reconstruídos nas entidades envolvidas (CERAMI; LAURENT, 2002). O nome SOAP foi mantido pela W3C por se tratar de um protocolo que se destacou rapidamente por sua forte ligação com o XML Schema.

O protocolo SOAP descreve a comunicação entre cliente e servidor como sendo uma serie de mensagens SOAP. Essas mensagens são, basicamente, envelopes com o conteúdo a ser enviado. Cada envelope deve conter um corpo que é onde fica as informações do que deve ser feito e podem conter uma seção com informações adicionais para a aplicação. Além disso, a seção do corpo pode conter uma seção de falta, semelhante a forma do XML-RPC.

Antes de apresentar exemplos das mensagens SOAP, trocadas entre o cliente e o servidor, cada uma das seções da mensagem será abordada. A primeira seção chamada de envelope é o elemento raiz das mensagens SOAP. Essa seção é iniciada com "<SOAP-ENV:Envelope>" e nela contêm um controle de versão através do uso de *namespace* utilizando o atributo "xmlns:SOAP-ENV" [4]. Caso o *namespace* informado não seja válido a mensagem pode ser ignorada.

A seção seguinte é chamada *Header* e pode conter informações adicionais úteis para a aplicação. Essa seção não esta presente na maioria das mensagens e, dessa forma, ao prover uma determinada informação é possível indicar que o o não entendimento desta ocasione a falha do método. Assim, isso pode servir para diferentes propósitos. O elemento "<SOAP-ENV:Header>" define a raiz desta seção.

O corpo da mensagem é definido por "<SOAP-ENV:Body>" e será explicado juntamente com os exemplos. Lembra-se ainda que, dentro da seção corpo há uma em especial que trata de falha e é definido pelo "<SOAP-ENV:Fault>". Esse elemento pode conter 4 elementos que descrevem o erro: *faultcode*, o tipo de erro acontecido; *faultstring*, a descrição do erro acontecido; *faultactor*, onde aconteceu a falha [5]; *detail*, um campo livre utilizado para enviar dados de volta para a aplicação. Quando acontece uma falta esse é o único elemento que pode vir dentro do corpo da mensagem, os dados que descrevem o erro podem variar. Por exemplo, ser retornado *faultstring* e *detail* ou somente *faultstring* ou outra combinação dos elementos explicados.

A Listagem 2.5 demostra uma chamada SOAP sendo feita. No envelope da mensagem tem-se três *namespaces* informados, um deles utilizado para versionamento do SOAP (atributo "xmlns:SOAP-ENV") e outros dois utilizados para descrever as versões da codificação dos dados em XML Schema (atributos "xmlns:xsi" e "xmlns:xsd"). Além disso, dentro da seção de corpo da mensagem ha mais um *namespace* definido pelo atributo "xmlns:ns1" no método sendo chamado, no exemplo *getWeather* definido por "<ns1:getWeather>". Esse último *namespace* pode ser utilizado pela aplicação para fazer uma diferenciação entre métodos com versões diferentes.

[4] Note que no XML-RPC não existe essa preocupação

[5] Uma mensagem SOAP pode ser reenviada para outro servidor conhecido sem informar o usuário disso. Esse roteamento está fora do escopo e está sendo mencionado a título de curiosidade.

Listagem 2.5: Exemplo de chamada SOAP com os cabeçalhos HTTP sendo omitidos (CERAMI; LAURENT, 2002).

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSChema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSChema">
 <SOAP-ENV:Body>
  <ns1:getWeather xmlns:ns1="urn:examples:weatherservice"
   SOAP-ENV:encodingStyle="http://www.w3.org/2001/09/soap-encoding/">
    <zipcode xsi:type="xsd:string">10016</zipcode>
  </ns1:getWeather>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Todas as mensagens SOAP tem um envelope como sendo a forma primordial. Dentro desse, existe uma seção com dados adicionais e/ou uma seção com o corpo da mensagem. No corpo da mensagem cada um dos seus sub-elementos são métodos que estão sendo chamados e cada sub-elemento desses sub-elementos correspondem aos parâmetros do método. Assim, na Listagem 2.5 há um único método sendo chamado de nome *getWeather* e um parâmetro de nome *zipcode* do tipo "xsd:string" definido pelo atributo "xsi:type" valendo 10016. Se houvessem mais parâmetros desse métodos, eles viriam todos no mesmo nível do *zipcode*. No exemplo foi utilizado um dos tipos de codificação de parâmetros possíveis, porém isso é configurado em cada um dos métodos através do atributo "SOAP-ENV:encodingStyle".

Todos os tipos de dados do XML Schema podem ser utilizados. Assim, qualquer um dos 50 tipos possíveis podem ser utilizados no envio de dados via SOAP. Entre esses há tipos complexos como os vetores e estruturas que seriam equivalentes aos do XML-RPC, mas com formas de utilização diferentes.

A descrição da resposta do método é análoga a descrição de chamada do método, pois ambas são mensagens SOAP. Na resposta, tem-se um envelope encapsulando informações tanto adicionais quanto do corpo da mensagem. O corpo da mensagem tem como elemento raiz o elemento "<SOAP-ENV:Body>", porém cada um dos sub-elementos serão respostas de métodos solicitados e cada sub-elemento destes serão os valores de retorno do método.

Na Listagem 2.6, o elemento "ns1:getWeatherResponse" é a resposta do método. Ele possui apenas um valor retornado definido como sendo um inteiro através do atributo"xsi:type" com valor 65. O tipo inteiro é um tipo simples, porém, quando retornado um tipo complexo como sub-elemento de "<return>" pode haver necessidade do uso de etiquetas correspondentes ao tipo em questão. Por exemplo, no caso de vetores as etiquetas seriam "<item>" e, no caso de ser devolvido uma estrutura, cada etiqueta assumiria o nome do membro sendo sendo retornado.

Em (KOHLHOFF; STEELE, 2003), uma analise do SOAP foi realizada visando performance no tráfego da rede. Durante o trabalho, o autor comenta que é normal um aumento de até 10 vezes na representação textual dos dados, quando comparado com o tamanho do dado em sua forma binária. Declara, ainda, que, o principal ponto onde o SOAP gasta tempo é nas conversões entre representação binária e textual e vice-versa. Entretanto, ele conclui que com um tamanho de pacote de comunicação maior, a vantagem pode ser do SOAP. Dessa forma, pacotes muito pequenos seriam atrasados e o SOAP não sofreria atrasos.

Listagem 2.6: Exemplo de chamada SOAP com os cabeçalhos HTTP sendo omitidos (CERAMI; LAURENT, 2002).

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSChema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSChema">
 <SOAP-ENV:Body>
  <ns1:getWeatherResponse xmlns:ns1="urn:examples:weatherservice"
   SOAP-ENV:encodingStyle="http://www.w3.org/2001/09/soap-encoding/">
   <return xsi:type="xsd:int">65</return>
  </ns1:getWeatherResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.3 Trabalhos relacionados

Trabalhos com agentes distribuídos existem os mais diversos (BELLIFEMINE; POGGI; RIMASSA, 1999; BUHLER; VIDAL, 2003; PIUNTI; RICCI; SANTI, 2009). Entre esses trabalhos, o Jade é um *framework* que utiliza o protocolo FIPA. Esse protocolo define formas de interação entre os diferentes agentes (BELLIFEMINE; POGGI; RIMASSA, 1999) e não como a interação ocorre. Observe, também, que o tempo entre a interação entre agentes para realocar tarefas é um fator crítico.

O Jade permite que agentes estejam dispersos utilizando a tecnologia RMI [6]. A execução de ações é através da adição e remoção de comportamentos. Esses podem ser de dois tipos: (i) simples; (ii) complexos. O comportamento simples equivale a uma ação executada sem interrupção que pode ser repetida até se obter o que se deseja. O comportamento complexo é um comportamento que pode ser interrompido e possui regras de pré e pós execução. Dessa forma, esse comportamento pode ser uma cadeia de ações a serem executadas em sequência ou de maneira não determinística. O Jason já possui uma forma de utilizar o Jade em seu ambiente, porém o uso do RMI não atende aos critérios do presente trabalho, visto que o RMI obrigaria a utilização de uma linguagem com orientação a objetos, porém, um dos critérios do presente trabalho é a independência de paradigma.

Em (BUHLER; VIDAL, 2003) o protocolo FIPA é utilizado na coordenação de agentes. Os servidores de serviço Web (WSS) possuem os recursos computacionais que são alocados para determinadas tarefas e os agentes decidem que tarefas serão alocadas para esses WSS. O conceito chave aqui é "atividades + coordenação" sendo que as atividades são realizadas pelos WSS e a coordenação (social) pelos agentes. A linguagem utilizada para descrever a coordenação foi uma linguagem de negócio. Esse trabalho é fortemente relacionado com o nosso, mas o objetivo dele é apenas coordenar os agentes em alto nível para se ter uma sociedade da melhor forma possível. Nossa meta é permitir que a plataforma Jason seja aberta e possa receber agentes e ambientes em diferentes linguagens seja ela orientada ou não à negócios e/ou à objetos.

O Cartago tem a finalidade de ser um meio de campo entre o agente e o ambiente (RICCI; VIROLI; OMICINI, 2006). Assim, para descrever o ambiente, duas abstrações de primeira classe são utilizadas. A primeira abstração é o agente que é o ator (ou personagem) existente no ambiente. Ele tem a capacidade de interagir ou criar os objetos com que vai trabalhar de maneira dinâmica. A segunda abstração é chamada de artefato e refere-se aos objetos que estão no mundo e podem ser utilizados pelo agente. Dessa forma, se um agente deseja interagir com um determinado artefato existe uma interface padrão que

[6] *Remote Method Interface* é um mecanismo análogo ao RPC, porém para orientação à objetos.

permite diversas formas de interação. Por exemplo, o agente pode ficar focado em um determinado artefato para receber notificações de mudanças no mesmo ou pode registrar interesse em propriedades especificas para quando houver modificação ser notificado e, ainda, pode executar ações especificas proporcionadas pelo artefato.

O Cartago-WS (PIUNTI; RICCI; SANTI, 2009) estende o Cartago com a finalidade de permitir que os artefatos sejam WS. Dessa forma, uma ação solicitada pelo agente pode ser exercida por um artefato que nem mesmo encontra-se na mesma máquina. Há uma serie de extensões possíveis de serem utilizadas que permitem características diferentes, por exemplo, coordenação e segurança. O Cartago-WS foi feito utilizando JAX-WS [7] baseado na tecnologia SOAP. Conforme mencionado no Capítulo 1, o presente trabalho se baseara diretamente na tecnologia SOAP ou na XML-RPC. Além disso, tanto os objetos quanto os agentes na implementação sendo proposta podem ser WS, enquanto, que no Cartago-WS somente os objetos.

[7] *Java Api for Xml Web Service*. Veja mais em http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/.

3 TRABALHO DESENVOLVIDO

O trabalho desenvolvido visa permitir que a plataforma Jason possa ter tanto ambiente quanto agentes implementados em outras linguagens. Assim, a plataforma Jason pode ser considerada uma arquitetura totalmente aberta, dado que, até o raciocínio dos agentes poderia ser feito fora da plataforma. A explicação de qual dos protocolos foi escolhido debatendo os critérios utilizados e a estrutura de métodos necessários no servidor será explicado na seção 3.1. O cliente desses métodos fica localizado dentro da plataforma Jason e os detalhes da implementação podem ser vistos na seção 3.2.

3.1 Servidor de Serviços Web

No capítulo 1 foi comentado que tanto SOAP quanto XML-RPC são mecanismos comuns para o conjunto de linguagens escolhidos, pois não importa qual deles se escolha haverá sempre pelo menos uma biblioteca para uso. Assim, o critério decisivo utilizado para decidir foi a simplicidade, visto que isso reflete nas mensagens e na curva de aprendizagem. Por exemplo, desconsiderando os dados entregues nas mensagens trocadas tem-se que o SOAP envia 472 bytes e o XML-RPC 173 bytes. Além disso, a curva de aprendizagem do XML-RPC é menor pois há quase seis vezes menos tipos de dados que o padrão SOAP.

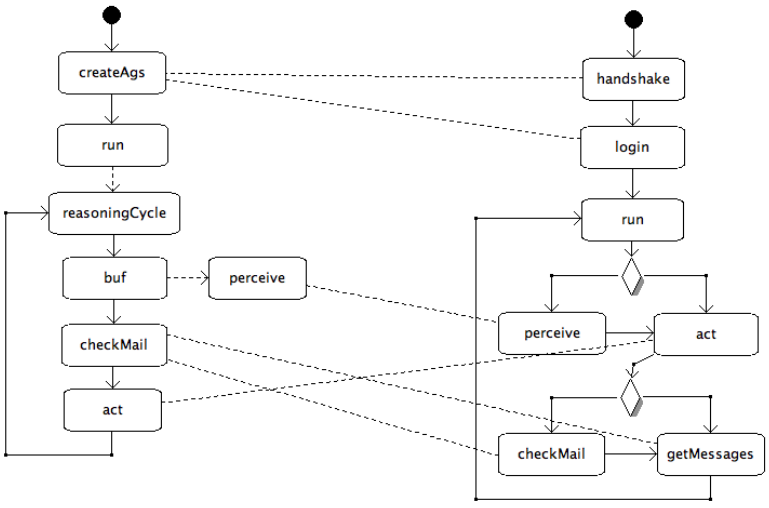

Figura 3.1: Diagrama relacionando o ciclo do agente Jason (esquerda) com os métodos à serem implementados em serviços (direita).

Tabela 3.1: Definição dos métodos obrigatórios no servidor de agentes.

Método	Parâmetros	Descrição
handshake	Desafio (String).	Resolve um desafio proposto.
login	Nome do agente (String).	Devolve um identificador (ID) numérico do agente.
logout	ID (Numero)	Libera o ID associado com um determinado agente.
perceive	ID (Numero) e Percepções (Array)	Associa um conjunto de percepções e retorna a ação do agente especificado.
act	ID (Numero) e Esvaziar percepções (Boolean)	Esvazia as percepções do agente e retorna sua ação.
checkMail	ID (Numero) e Mensagens (Array)	Entrega as mensagens novas e retorna as mensagens à serem enviadas aos outros agentes.
getMessages	ID (Numero)	Retorna todas as mensagens à serem enviadas aos outros.

Com o protocolo a ser utilizado definido, tudo que resta é definir como os agentes e o ambiente podem ser implementados utilizando WS. Um servidor sempre representa uma coleção de entidades do seu tipo determinado. No caso do ambiente deve-se lembrar que todos os métodos influenciarão um único ambiente compartilhado por diversos agentes.

Na Tabela 3.1 há as definições sobre os métodos obrigatórios a serem implementados no lado do servidor. Esses métodos são chamados pela plataforma Jason em momentos específicos para operar o agente. Os métodos *handshake* e *login* são chamados durante a criação do agente na inicialização da plataforma Jason. O primeiro método recebe um desafio (uma string qualquer) e deve responder o desafio da forma esperada. Desse modo, a plataforma Jason sabe que o WSS sendo utilizado tem o mínimo de confiabilidade exigido. O segundo método, *login*, serve para associar um identificador único com determinado agente. Esse identificador é utilizado nas chamadas seguintes da plataforma Jason. Ao término da plataforma Jason a função *logout* é chamada para liberar o identificador único criado.

Todo agente se comunica com outros agentes em algum nível, seja ela de forma direta (recebendo uma mensagem de outro) ou indireta (alterações de percepções do ambiente). Para a comunicação direta existir entre agentes fora da plataforma Jason, existem os métodos *checkMail* e *getMessages*. O método *checkMail* entrega as mensagens novas para o agente e, ao fim, chama *getMessages* para devolver todas as mensagens que o agente deseja enviar para os demais agentes. Note que, se o agente não tiver novas mensagens à serem recebidas, a plataforma pode chamar de maneira direta a função para consultar as mensagens à serem enviadas. Esse comportamento é observado na Figura 3.1.

Quanto a forma de comunicação indireta, os agentes percebem modificações no mundo através de "sensores" que refletem as percepções que o agente tem sobre as coisas no ambiente. De forma análoga às mensagens, as percepções são alteradas pelo método *perceive*. Esse método serve para cadastrar o estado atual conhecido do ambiente e tem como retorno uma ação que deve ser executada no ambiente através do método *act*. A

Tabela 3.2: Definição dos métodos obrigatórios no servidor do ambiente.

Método	Parâmetros	Descrição
handshake	Desafio (String).	Resolve um desafio proposto.
addPercept	Percepção	Insere uma percepção para todos os agentes.
addPerceptLocal	Nome do agente (String) e Percepção.	Insere uma percepção para um agente específico.
clearPercepts	Sem parâmetros.	Apaga as percepções comuns à todos os agentes.
clearPerceptsLocal	Nome do agente (String).	Apaga as percepções específicas de um agente.
havePercepts	Nome do agente (String).	Há ou não percepções novas.
getPercepts	Nome do agente (String) e envio atualizado (Boolean).	Retorna todas as percepções do agente.
removePercept	Percepção.	Retira uma percepção comum de todos os agentes.
removePerceptLocal	Nome do agente (String) e Percepção.	Retira uma percepção que é específica de um agente.
performAction	Nome do agente (String) e Ação.	O agente executa a ação especificada.

plataforma Jason decide chamar *act* ao invês de *perceive* quando não houver alterações nos sensores do agente ou quando não há percepção de nenhum dos sensores (conjunto vazio).

Na Tabela 3.2 tem-se a descrição dos métodos mandatórios para implementação de um WSS como ambiente. Esses métodos lidam em grande parte com percepções que serão explicados mais adiante. Quando, na inicialização da plataforma, o agente precisa de informações dos seus "sensores" é feita a chamada para o método *handshake*. Ele trabalha da mesma maneira que o WSS do agente. As funções que adicionam (*addPercept*), removem (*removePercept*) ou esvaziam (*clearPercepts*) o conjunto de percepções são aos pares para deixar claro quando opera no conjunto comum ou específico (são as mesmas funções, mas terminadas com a palavra Local).

A recuperação das percepções é feita unicamente via chamada ao método *getPercepts*. Esse método, quando chamado, recebe o agente operando para conseguir montar o conjunto de percepções do mesmo e se deve considerar isso como a percepção do ciclo do Jason para computar como atualização. O conjunto de percepções enviado é o conjunto comum ou global de percepções unido com o conjunto de percepções locais ou específicas do agente informado. O método *havePercepts* serve para informar a plataforma que há novas percepções ou não, assim a mesma sabe quando pegar as percepções.

Durante toda a simulação o agente realiza alguma ação, seja no ambiente ou não. Essas ações são realizadas utilizando o método *performAction* que recebe o agente (executor) e a ação (ato) para saber como as percepções serão afetadas. A ação recebida utiliza a estrutura de uma percepção. Assim, a percepção é uma tupla com três elementos: (i) operador, serve como nome da percepção e tenta dar ideia de qual a finalidade da mesma; (ii) termos, lista com a finalidade de permitir a parametrização da percepção; (iii) anota-

ções, lista de percepções referentes a mesma. Logo, a percepção de um agente que jey é um animal com 55% de chance de ser um cachorro e 40% de ser um gato pode ser expressa da seguinte forma: “(animal, [jey], [(cachorro, [0.55], []), (gato, [0.40], [])])”.

Cabe salientar ainda que ambas implementações são independentes, isto é, tanto o ambiente pode ser usado sem a implementação dos agentes quanto os agentes podem ser usados sem o ambiente. Essa independência permite que os agentes sejam utilizados com outros agentes da plataforma. Além disso, todo WSS deve ser desenvolvido com finalidade específica para uma dada simulação.

3.2 Cliente de Serviços Web

O cliente do WSS fica localizado na plataforma Jason. A implementação atual estende tanto a classe da arquitetura do agente, *AgArch*, quanto a classe de ambiente, *Environment*, para personalizar os comportamentos necessários no uso da plataforma. A Figura 3.2 demostra a implementação para o agente.

O agente é um cliente de WS e a interface construída para acessar o servidor implementado. A classe *Perceive* possui a responsabilidade de traduzir uma tupla de percepção [1] do e para o formato da plataforma. Já a classe *AgentClient* tem a responsabilidade de ser o único acesso ao WSS. O uso dessa classe é feito pela de arquitetura do agente que sabe o momento que ele precisa de uma coisa ou de outra.

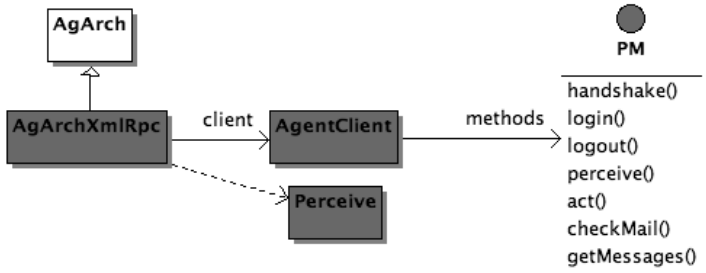

Figura 3.2: Diagrama da arquitetura do agente desenvolvido.

As mensagens de comunicação trocadas entre os agentes para uma comunicação direta são tuplas com o seguinte formato: (i) identificador único da mensagem; (ii) quem esta emitindo a mensagem, não é necessário preenche-lo; (iii) tipo da mensagem sendo enviada, de preferência aos formatos existentes do Jason; (iv) a quem se destina a mensagem sendo enviada; (v) mensagem sendo enviada. Desta forma, quando se deseja dizer para um agente (beltrano) que o disco esta gravado então a seguinte mensagem será enviada: “<id42,„tell,beltrano,disco(gravado)>”. As crenças dos agentes, entretanto, podem ser pensadas de duas formas: (i) o agente é responsável por manter internamente suas crenças; (ii) o agente guarda suas crenças junto das percepções utilizando as ações do ambiente.

Dadas essas duas opções, foi trabalhado com a segunda, que guarda as percepções e crenças dos agentes no ambiente e parece ser um modelo mais próximo do que existe hoje na plataforma Jason. Dessa forma, daqui por diante, quando for mencionado, o dado de percepção este fará alusão tanto a elementos percebidos do ambiente quanto

[1] Explicada na seção 3.1.

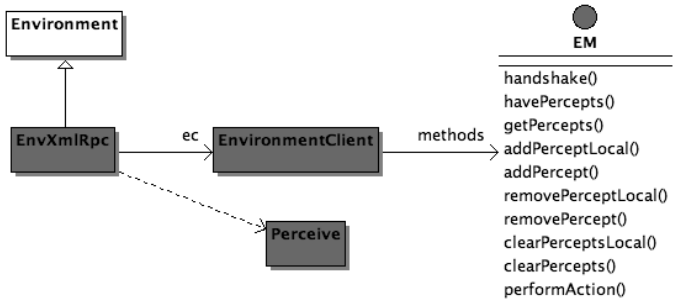

Figura 3.3: Diagrama do ambiente desenvolvido.

crenças, conclusões ou desejos feitos pelo agente. Assim, o ambiente é responsável por armazenar as percepções dos agentes e por conhecer como as ações dos agentes devem ser desempenhadas. Tanto o ambiente quanto o agente possuem uma estrutura de classes semelhantes. Essa semelhança tem como vantagem o ganho de velocidade quando, por ventura, for necessária alguma modificação no futuro.

Dessa forma, o desenvolvimento do lado do ambiente seguiu um modelo parecido com o do agente, conforme pode ser visto na Figura 3.3. Essa figura explica que o ambiente desenvolvido (classe *EnvXmlRpc*) especializa o ambiente da plataforma (classe *Environment*) para se responsabilizar por quando ir no servidor. A classe *Perceive* tem a responsabilidade de traduzir as tuplas recebidas de e para o formato da plataforma, conforme já explicado. A responsabilidade da classe *EnvironmentClient* é de ser o único meio de acessar o servidor externo.

Além disso, as interfaces servem para padronizar o acesso externo realizado. Essa escolha pelas interfaces foi feita por dois motivos. O primeiro foi deixar isolado os métodos que o servidor precisa responder. Enquanto, o segundo foi tirar vantagem da técnica de reflexibilidade que permite a construção de métodos que servem de *proxy*. A implementação do cliente feita na plataforma Jason pode ser visualizada no Anexo A.

4 EXEMPLO DE USO

O presente capítulo apresenta dois exemplos desenvolvidos. Na seção 4.1 se apresenta o exemplo *room* adaptado para executar dois agentes (*porter* e *paranoid*) na linguagem Python. Já o seguinte apresenta o exemplo *game of life*, adaptado para ter o ambiente desenvolvido em Haskell. Os agentes em Python estão na seção 4.2.

4.1 Exemplo *Room*

O exemplo *Room* foi adaptado para executar dois agentes de maneira externa à plataforma Jason. Assim, a Listagem 2.1 (pg. 9) deve ficar conforme a Listagem 4.1. Fora isso, os dois programas que implementam o servidor de agentes do *porter* e do *paranoid* devem ser disparados antes da plataforma Jason iniciar a simulação. A implementação desses pode ser vista no Anexo B.

Listagem 4.1: Arquivo de projeto do Jason adaptado.

```
// Isso eh um comentario
MAS room {
  infrastructure: Centralised
  environment: RoomEnv
  executionControl: jason.control.ExecutionControl
  agents:
    porter
    [url="http://localhost:8080"]
    agentArchClass maro.architecture.AgArchXmlRpc
    ;
    claustrophobe
    ;
    paranoid
    [url="http://localhost:8081"]
    agentArchClass maro.architecture.AgArchXmlRpc
    ;
}
```

A Listagem 4.1 define uma simulação na plataforma Jason com três agentes heterogêneos. O primeiro chamado *porter* encontra-se localizado em um WSS definido pela url "http://localhost:8080" que define o servidor sendo a máquina local na porta 8080. Nessa url poderia ser colocado lugares diferentes sem nenhum problema e a porta poderia ser omitida quando a mesma for a 80. O segundo agente nomeado *claustrophobe* é um agente Jason e o último agente é outro WSS localizado na porta 8081.

A implementação dos agentes em Jason podem ser consultadas na Listagem 2.2 (página 10). O agente *porter* como pode ser visto, deve trancar a porta quando receber do agente *paranoid* a meta *locked* e deve destrancar a porta quando receber do agente *claustrophobe* a meta $\sim$*locked*. De acordo com a Figura 3.2 (pg 22), há 7 métodos que o servi-

dor necessita implementar. Os métodos *handshake*, *login* e *logout* não serão comentados aqui.

Listagem 4.2: Método de percepção e ação do agente *porter*.

```
def perceive(id, percepts):
        data = agentL[id]
        agentL[id] = (data[0], percepts, data[2])
        return act(id, False)

def act(id, eraseAll):
        if eraseAll:
                data = agentL[id]
                agentL[id] = (data[0], [], data[2])
        data = agentL[id]
        if len(data[2]) >= 1:
                ret = exclusion_act(data[1], data[2][0])
                if len(ret) > 0:
                        agentL[id] = (data[0], data[1], data[2][1:])
                return ret
        return ""

def exclusion_act(perceptions, message_received):
        M = { "~locked(door)":"locked(door)", "locked(door)":"~locked(door)" }
        P = { "~locked":"lock", "locked":"unlock" }
        test = M[ message_received[4] ][:-1].split('(')
        for p in perceptions:
                functor = p[0]
                terms = p[1]
                annots = p[2]
                if functor == test[0] and terms[0] == test[1]:
                        return P[functor]
        return ""
```

Na Listagem 4.2 tem-se a implementação de dois dos métodos essenciais. O primeiro (*perceive*) recebe o identificador do agente criado na inicialização e a nova lista de percepções e usa esses dados para atualizar a tupla que contêm, respectivamente, o nome do agente, as percepções e as mensagens recebidas. Agora, o método *act* deve retornar a ação que o agente *porter* fará no momento e, para isso, recebe o identificador do agente e um valor lógico que indica se as percepções devem ser apagadas ou não. Assim, é possível chamar esse método sem passar por *perceive* quando o grupo de novas percepções for o grupo vazio. Portanto, a primeira atividade dessa função é verificar esse valor e limpar as percepções caso seja necessário. Após, o uso verifica se há mensagens, havendo chama uma função auxiliar para definir se há ou não ação a realizar e consome a mensagem.

Listagem 4.3: Métodos relacionados com a comunicação do agente *porter*.

```
def receiveMessages(id, messages):
        data = agentL[id]
        msgL = []
        for msg in messages:
                m = msg[1:-1].split(',')
                if m[1] == "claustrophobe" and m[4] == "~locked(door)":
                        msgL.append(m);
                elif m[1] == "paranoid" and m[4] == "locked(door)":
                        msgL.append(m);
        agentL[id] = (data[0], data[1], msgL)
        return sendMessages(id)

def sendMessages(id):
        return []
```

O agente que origina à mensagem é testado no momento do recebimento da mensagem, onde o agente pode descarta-la ou guarda-la. Esse comportamento pode ser obser-

vado na Listagem 4.3. Nela os métodos *checkMail* e *getMessages* mapeiam, respectivamente, para *receiveMessages* e *sendMessages*. O método de recebimento de mensagens, *receiveMessages*, sabe como parsear a tupla que veio (linha 5) e somente cadastra as mensagens relevantes para desempenhar as mesmas ações do código Jason. Dessa forma, ele só considera as mensagens relevantes que para o agente *paranoid* é um pedido de trancamento e para o agente *claustrophobe* é um pedido de destrancamento da porta. O agente *porter* não envia nenhum tipo de mensagens para outros agentes, assim a implementação do *getMessages* mapeado para *sendMessages* na implentação retorna um conjunto vazio representando que não deseja enviar mensagens.

Listagem 4.4: Métodos do agente *paranoid*.

```
def act2(id, eraseAll):
        if eraseAll:
                data = agentL[id]
                agentL[id] = (data[0], [], data[2])
        data = agentL[id]
        if len(data[1]) > 0:
                sendMessage = exclusion_act(data[1])
                if len(sendMessage) > 0:
                        data[2].append(sendMessage)
                        agentL[id] = (data[0], data[1], data[2])
        return ""

def exclusion_act(perceptions):
        p = perceptions[0]
        functor = p[0]
        terms = p[1]
        annots = p[2]
        if functor == "~locked" and terms[0] == "door":
                try:
                        msgId = agentL["msgId"]
                except KeyError:
                        msgId = 0
                agentL["msgId"] = msgId + 1
                return "<mprpc%02d,,achieve,porter,locked(door)>" % msgId
        return ""

def sendMessages(id):
        data = agentL[id]
        ret = []
        if len(data[2]) > 0:
                ret = data[2]
                agentL[id] = (data[0], data[1], [])
        return ret
```

No agente *paranoid* a situação se altera um pouco, a função de percepção (*perceive*), ação (*act*) e recebimento de mensagens (*checkMail*) não são mostrados na Listagem 4.4. O método de percepção é igual ao mostrado anteriormente, porém ele passa a chamar *act2* ao invés de *act*. Já no método *act*, o método *act2* só é chamado quando precisa limpar todas as crenças. O método *checkMail* simplesmente realiza a chamada a função *getMessages* (mapeada na implementação para *sendMessages*) para enviar para a plataforma Jason as mensagens desejadas.

A presente implementação considera uma mensagem como uma string. Essa string tem o seguinte formato: “<id, emissor, tipo, vitima, mensagem>”. Esses dados são usados para construir uma tupla com 5 elementos, conforme já explicado na seção 3.2. Essa string é construída na função interna *exclusion_act*, chamada somente quando há percepções, e guardada para ser enviada depois pela função *getMessages*. Essa função retorna as mensagens que o agente deseja enviar e limpa o campo de mensagens à serem enviadas.

4.2 Exemplo *Game of Life*

O presente exemplo tem como foco apresentar uma implementação do ambiente como WS. O serviço segue a interface apresentada na Figura 3.3 presente na página 23, onde existem 10 métodos. O presente foi organizado em um diretório e um arquivo *Python*. No arquivo *Python* tem-se a implementação do raciocínio dos agentes e no diretório tem-se a implementação do ambiente. O ambiente é tanto a implementação da API pública em WS quanto a interface que o usuário pode interagir (que na versão Jason existia).

Listagem 4.5: Arquivo de projeto do *Game-of-Life*.

```
MAS game_of_life {
  infrastructure: Centralised(pool,2)
  environment: maro.Environment.EnvXmlRpc("http://localhost:8079")
  executionControl: jason.control.ExecutionControl

  agents:
    cell
    [url="http://localhost:8080/RPC2"]
    agentArchClass maro.architecture.AgArchXmlRpc
    #3600 // matrix 60x60
    ;
}
```

A Listagem 4.5 mostra o arquivo de projeto adaptado. Na URL a parte com "RPC2" poderia ser omitida deixando somente o endereço com a porta. O ambiente recebe a URL da maneira como demostrada. A infraestrutura utilizada é a centralizada, porém é utilizado um conjunto de *threads* que no caso foi definido no valor 2 (duas *threads*).

Listagem 4.6: Tipos usados pelo ambiente.

```
type Action = Percept
data Percept = Percept {
        functor::String,
        params::[String],
        annots::[Percept]
    } deriving (Show,Eq,Ord)
data TPerception = Perceptions {
        global::[Percept],
        local::[(String, Percept)],
        updated::[String]
    } deriving (Show)
```

Antes de mostrar os trechos da implementação da parte do ambiente em *Haskell* é necessário falar sobre os dois tipos de dados que são usados para suportar o desenvolvimento. Na Listagem 4.6 tem-se os tipos, o primeiro denominado *Percept* é a tupla de percepções explicada na seção 3.1. Esse tipo de dado também pode ser referenciado como *Action*. Já o tipo *TPerception* contém, respectivamente, uma lista de percepções, uma lista com tuplas que mapeam nome do agente para uma percepção e uma lista de nomes de agentes. Esse tipo serve para guardar separadamente as percepções globais dos locais e guarda ainda os agentes que estão atualizados (o cliente já requisitou os dados atuais). Cabe salientar, ainda, que todas as listas são mantidas ordenadas para facilitar a busca nas mesmas.

Na Listagem 4.7 são mostrados os métodos implementados pelo WSS para adicionar, remover e limpar as percepções locais ou globais. Esses métodos tratam a lista de percepções que eles alteram como conjuntos. Dessa forma, não é necessário se preocupar com percepções repetidas porque elas não vão existir. Além disso, nas operações de remoção e limpeza é utilizada a diferença entre dois conjuntos. Essas funções da listagem encontram-se mapeadas e publicadas por um WS, porém, quando comparadas com a Tabela 3.2 (pg 21) pensa-se que há parâmetros em excesso. Na implementação do método *addPerceptLocal*, por exemplo, têm-se por parâmetros uma referência ao tipo interno *TPerceptions*, uma *string* (nome do agente), uma tupla de percepção e como retorno

um booleano encapsulado. Esse protótipo de função é possível pois quando se realiza o mapeamento das funções publicadas pelo WS passa-se alguns parâmetros da chamada a priori. Essa característica é utilizada em quase todas as funções e, principalmente, na *performAction*. Consulte no Anexo C a função *serving* (pg 57) para maiores detalhes.

Listagem 4.7: Métodos relativos às percepções.

```
addPercept :: IORef(TPerception) -> Percept -> IO Bool
addPercept p perception = do
    (Perceptions gp lp ua) <- readIORef p
    let gp' = insertSet perception gp
    modifyIORef p $ \(Perceptions _ lp _) -> (Perceptions gp' lp [])
    return True

addPerceptLocal :: IORef(TPerception) -> String -> Percept -> IO Bool
addPerceptLocal p name perception = do
    (Perceptions gp lp ua) <- readIORef p
    let lp' = insertSet (name,perception) lp
    modifyIORef p $ \(Perceptions gp _ _) -> (Perceptions gp lp' [])
    return True

removePercept :: IORef(TPerception) -> Percept -> IO Bool
removePercept p perception = do
    (Perceptions gp lp ua) <- readIORef p
    let gp' = gp `minus` [perception]
    modifyIORef p $ \(Perceptions _ lp _) -> (Perceptions gp' lp [])
    return True

removePerceptLocal :: IORef(TPerception) -> String -> Percept -> IO Bool
removePerceptLocal p name perception = do
    (Perceptions gp lp ua) <- readIORef p
    let lp' = lp `minus` [(name,perception)]
    modifyIORef p $ \(Perceptions gp _ _) -> (Perceptions gp lp' [])
    return True

clearPercepts :: IORef(TPerception) -> IO Bool
clearPercepts p = do
    (Perceptions gp lp ua) <- readIORef p
    if (null gp) then
        return True
     else do
        modifyIORef p $ \(Perceptions _ lp _) -> (Perceptions [] lp [])
        return True

clearPerceptsLocal :: IORef(TPerception) -> String -> IO Bool
clearPerceptsLocal p name = do
    (Perceptions gp lp ua) <- readIORef p
    if (null gp) then
        return True
     else do
        let lp' = lp `minus` [(name, perception) | (n, perception) <- lp, n==name]
        modifyIORef p $ \(Perceptions _ lp _) -> (Perceptions [] lp [])
        return True
```

O ambiente é responsável por armazenar todas as percepções que os agentes possuem, assim ele deve também disponibilizar formas desses dados serem recuperados. A Listagem 4.8 mostra os dois métodos que são usados no momento do agente conhecer as suas percepções. A função *havePercepts* utiliza a lista de agentes atualizados para devolver verdadeiro quando é necessário ocorrer uma chamada ao método *getPercepts*. O método *getPercepts* define as percepções à serem retornadas na linha 9 da Listagem 4.8 como sendo a união das percepções globais com o conjunto de percepções locais filtrados pelo nome do agente.

Listagem 4.8: Métodos para recuperar as percepções.

```
havePercepts :: IORef(TPerception) -> String -> IO Bool
havePercepts p name = do
    pp <- readIORef p
    return $ not $ (updated pp) `has` name

getPercepts :: IORef(TPerception) -> String -> Bool -> IO [Percept]
getPercepts p name isUpdate = do
    (Perceptions gp lp ua) <- readIORef p
    let r = union gp [lp' | (n, lp') <- lp, n == name]
    if isUpdate == False then
        return r
     else do
        let ua' = insertSet name ua
        modifyIORef p $ \(Perceptions gp lp _) -> (Perceptions gp lp ua')
        return r
```

A presente implementação do ambiente aceita quatro ações possíveis vindo do agente Jason. As ações nomeadas são *skip*, *live* e *die*, uma quarta ação existe e é a ação de ignorar para quando o ambiente não conhece-la não realizar nenhuma tarefa. Essa seleção é feita baseado no operador da ação recebida (linha 5 da Listagem 4.9).

Listagem 4.9: Método que realiza ações.

```
performAction :: IORef(TPerception) -> SyncModel -> GLWidget -> String
     -> Action -> IO Bool
performAction p controller viewer name action =
    let
        actionName = functor action
        actionPars = params action
        actionBy = name
    in
        case actionName of
            "skip"  -> cmdRealize p controller viewer actionBy Nothing
            "live"  -> cmdRealize p controller viewer actionBy (Just True)
            "die"   -> cmdRealize p controller viewer actionBy (Just False)
            _       -> return True -- ignore action
```

Na Listagem 4.9, a chamada a função interna *cmdRealize* permitiu que fosse abstraído código. Essa é a função que realiza o trabalho "sujo" de atualizar o modelo e solicitar a reexibição da interface com o usuário. Os únicos parâmetros úteis para a célula são: a referência as percepções, quem está realizando a ação e que tipo de ação é. Para representar o tipo de ação é utilizado o tipo *Maybe Bool*. Dessa forma, as funções *skip*, *live* e *die* são mapeadas, respectivamente, para não ser um tipo booleano ou ser um tipo booleano valendo verdadeiro ou ser um tipo booleano valendo falso. Maiores detalhes consulte o Anexo C.

5 CONCLUSÃO

No presente trabalho foi apresentado uma forma de aumentar a integração da plataforma Jason com outras linguagens além do Java. A forma apresentada utiliza o conceito de serviços para permitir a utilização de agentes e ambientes heterogêneos, isto é, diferentes componentes da plataforma podem rodar em ambientes computacionais diferentes.

Assim, ao se construir os métodos do servidor deve-se tomar cuidado para evitar problemas. Um método quando implementado incorretamente pode refletir somente em outro ponto. Dessa forma, é importante no futuro ter-se bibliotecas específicas para cada linguagem visando dar uma implementação padrão correta à todas as funções necessárias ao serviço web e seu desenvolvimento pode ser conforme demanda.

O exemplo do *Game Of Life* apresentado na seção 4.2 possui um ciclo médio de 28 segundos para receber a comunicação de todos os agentes. Enquanto que, na plataforma para todos os 3600 agentes (matriz de 60 por 60) o tempo é inferior a 1 segundo. Conforme já dito, o protocolo de comunicação é o *XML-RPC* baseado na linguagem *XML*. Em (KOHLHOFF; STEELE, 2003) é mostrado que a linguagem *XML* pode deixar as mensagens até 10 vezes maiores e que o principal tempo gasto na comunicação de um processo para outro é perdido no transformar o dado de binário para *XML* e vice-versa.

Dessa forma, a lentidão no sistema pode ser pensada como sendo culpa da quantidade de agentes e da representação textual dos dados trocados. O ideal seria trocar a representação textual e, também, rever o protocolo para estudar se há uma forma de evitar a troca de mensagens desnecessárias. Por exemplo, na implementação atual, se um agente não recebe e não envia mensagens para outros agentes a plataforma Jason chamará a função *checkMail* e obterá um retorno vazio todas as vezes. Outro exemplo é o mapeamento de todas as ações do ambiente em uma única função evitando que seja conhecido as ações válidas através da consulta da listagem de métodos do servidor.

O mecanismo de segurança criado pelo protocolo desenvolvido visa dar um mínimo de segurança aos servidores, porém é necessário pensar em mecanismos mais fortes para evitar o problema do homem no meio. Talvez permitir uma entrada no arquivo de configuração do projeto ativando uma cifragem à ser utilizada nas mensagens trocadas. Além disso, o uso de serviços permitiria a criação de uma ferramenta de teste que torna possível a substituição de um dos componentes (cliente ou um dos servidores) para se validar o outro componente. Esse teste seria realizado conhecendo as entradas e a saída esperada para a verificação do comportamento esperado. Essa forma de teste é conhecida por teste unitário e considera todo o serviço sendo testado como uma só unidade.

REFERÊNCIAS

ALLMAN, M. An evaluation of XML-RPC. *ACM SIGMETRICS Performance Evaluation Review*, ACM, v. 30, n. 4, p. 11, 2003.

BELLIFEMINE, F.; POGGI, A.; RIMASSA, G. JADE–A FIPA-compliant agent framework. In: CITESEER. *Proceedings of PAAM*. [S.l.], 1999. v. 99, p. 97–108.

BORDINI, R. et al. Jason: A java-based agentspeak interpreter used with saci for multi-agent distribution over the net, manual, first release edition. Jan 2004. Disponível em: <http://jason.sourceforge.net>.

BUHLER, P. A.; VIDAL, J. M. Adaptive workflow = web services + agents. In: *In Proceedings of the International Conference on Web Services*. [S.l.]: CSREA Press, 2003. p. 131–137.

CERAMI, E.; LAURENT, S. *Web services essentials*. [S.l.]: O'Reilly & Associates, Inc. Sebastopol, CA, USA, 2002.

DASTANI, M.; STEUNEBRINK, B. Modularity in bdi-based multi-agent programming languages. In: *WI-IAT '09: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*. Washington, DC, USA: IEEE Computer Society, 2009. p. 581–584. ISBN 978-0-7695-3801-3.

KOHLHOFF, C.; STEELE, R. Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems, 12 Int. In: *World Wide Web Conference (WWW2003)*. [S.l.: s.n.], 2003.

PIUNTI, M.; RICCI, A.; SANTI, A. SOA/WS Applications using Cognitive Agents working in CArtAgO Environments. In: *Decimo Workshop Nazionale "Dagli Oggetti agli Agenti"(WOA 2009)*. [S.l.: s.n.], 2009.

RAO, A. AgentSpeak (L): BDI agents speak out in a logical computable language. *Agents Breaking Away*, Springer, p. 42–55, 1996.

RICCI, A.; VIROLI, M.; OMICINI, A. CArtAgO: An infrastructure for engineering computational environments in MAS. *Workshop E4MAS*, Citeseer, p. 102–119, 2006.

SILVA, L. D.; PADGHAM, L. Planning on demand in BDI systems. *Proc. of ICAPS-05 (Poster)*, Citeseer, 2005.

APÊNDICE A FONTE DO CLIENTE WS NA PLATAFORMA JASON

Arquivo referente a interface dos métodos do ambiente: xmlrpc/EM.java

```
1  package maro.xmlrpc;
   import java.util.List;
3  interface EM {
     public Integer handshake(String challenge);
5    public boolean havePercepts(String agName);
     public List<Object> getPercepts(String agName, boolean onlyUpdate);
7    public boolean addPerceptLocal(String agName, Object o);
     public boolean addPercept(Object o);
9    public boolean removePerceptLocal(String agName, Object o);
     public boolean removePercept(Object o);
11   public boolean clearPerceptsLocal(String agName);
     public boolean clearPercepts();
13   public boolean performAction(String agName, Object s);
   }
```

Arquivo referente a interface dos métodos dos agentes: xmlrpc/PM.java

```
   package maro.xmlrpc;
2  import java.util.List;
   interface PM {
4    public Integer handshake(String challenge);
     public Integer login(String agName);
6    public boolean logout(Integer id);
     public String perceive(Integer id, Object[] percept);
8    public String act(Integer id, boolean eraseAll);
     public List<String> checkMail(Integer id, String[] news);
10   public List<String> getMessages(Integer id);
   }
```

Arquivo responsável pela serialização dos dados Jason: xmlrpc/Perceive.java

```
1  package maro.xmlrpc;
   import jason.asSyntax.ListTerm;
3  import jason.asSyntax.Literal;
   import jason.asSyntax.Term;
5  import java.util.LinkedList;
   import java.util.List;
7  public class Perceive {
     static public List<Object> dump(List<Literal> ll) {
9      List<Object> ret;
       if (ll == null) return null;
11     ret = new LinkedList<Object>();
       for (Literal l: ll) {
13       Object s = dumpLiteral(l);
         if (s != null) ret.add(s);
15     }
       return ret;
17   }
     static public List<Literal> undump(Object[] lo) {
19     List<Literal> ret;
       if (lo == null) return null;
```

```
21      ret = new LinkedList<Literal>();
        for (Object o: lo) {
23        String str = undumpLiteral(o);
          if (str != null) {
25          Literal li = Literal.parseLiteral(str);
            if (li != null) ret.add(li);
27        }
        }
29      return ret;
    }
31  static public Object dumpLiteral(Literal l) {
      List<Object> las;
33    List<String> lts;
      List<Term> lts_raw;
35    ListTerm las_raw;
      String functor_complete;
37    if (l == null) return null;
      lts_raw = l.getTerms();
39    las_raw = l.getAnnots();
      lts = new LinkedList<String>();
41    if (lts_raw != null) for (Term t: lts_raw) lts.add( t.toString() );
      las = new LinkedList<Object>();
43    if (las_raw != null) for (Term t: las_raw) las.add( dumpLiteral((Literal)t) );
      if (l.negated()) functor_complete = new String("~" + l.getFunctor());
45    else functor_complete = new String(l.getFunctor());
      return new Object[] {
47                functor_complete,
                  lts.toArray(),
49                las.toArray() };
    }
51  static public String undumpLiteral(Object o) {
      String ret;
53    if (o == null) return new String("");
      if (!(o instanceof Object[]) || ((Object[])o).length != 3) return null;
55    Object [] oa = (Object[]) o;
      if (!(oa[0] instanceof String)
57        || !(oa[1] instanceof Object[])
          || !(oa[2] instanceof Object[]))
59      return null;
      String functor = (String)   oa[0];
61    Object [] lts  = (Object[]) oa[1];
      Object [] lms  = (Object[]) oa[2];
63    ret = new String(functor);
      if (lts.length > 0) {
65      ret = ret + "(";
        boolean first = true;
67      for (Object lt: lts) {
          if (first == true) {
69          ret = ret + ((String) lt);
            first = false;
71        } else ret = ret + ",␣" + ((String)lt);
        }
73      ret = ret + ")";
      }
75    if (lms.length > 0) {
        ret = ret + "[";
77      boolean first = true;
        for (Object m: lms) {
79        if (first == true) {
            ret = ret + undumpLiteral(m);
81          first = false;
          } else ret = ret + ",␣" + undumpLiteral(m);
83      }
        ret = ret + "]";
85    }
      return ret;
87  }
}
```

Arquivo responsável pelo acesso ao agente: xmlrpc/AgentClient.java

```
  package maro.xmlrpc;
2 import redstone.xmlrpc.XmlRpcProxy;
```

```
import redstone.xmlrpc.XmlRpcFault;
import java.util.Random;
import java.net.URL;
import java.util.List;
public class AgentClient {
  protected Random random;
  protected PM methods;
  protected String myName;
  protected int id;
  private AgentClient() { /* denied */ }
  public AgentClient(String ag, String uri, String cipher) throws Exception {
    if (uri == null) throw new Exception("URI is nulled");
    methods
      = (PM)XmlRpcProxy.createProxy(new URL(uri), new Class[] {PM.class}, true);
    if (methods == null) throw new Exception("Cannot create proxy to methods...");
    random = new Random();
    myName = ag;
    connect();
  }
  protected void connect() throws Exception {
    // step 1 - handshake
    String challenge;
    int ret;
    int value;
    try {
      value = random.nextInt(32768); // 2 ** 15 ou 2 ^ 15
      challenge = new String("i" + Integer.toString(value));
      ret = methods.handshake( challenge );
    } catch (Exception e) { // se falhou no decodificar desafio falhou!
      throw new Exception("Web Server is not trusted - " + e);
    }
    try {
      // Quando a resposta do desafio veio errada falhou tb!
      // Entretanto, divide o try pra deixar melhor localizado
      if (ret != (value * value)) throw new Exception();
    } catch (Exception e) {
      throw new Exception("Web Server is not trusted - Response to Challenge is
          Wrong");
    }
    // step 2 - login
    login();
  }
  protected void login() throws Exception {
    Integer ret;
    try {
      ret = methods.login( myName );
    } catch (Exception e) {
      throw new Exception("Internal Error - " + e);
    }
    id = ret;
  }
  public void logout() throws Exception {
    try methods.logout( id );
    catch (Exception e) throw new Exception("Internal Error - " + e);
  }
  public String perceive(List<Object> ll) throws Exception {
    String act = null;
    try {
      if (ll == null) act = methods.act( id, false );
      else if (ll.isEmpty()) act = methods.act( id, true );
      else act = methods.perceive( id, ll.toArray() );
    } catch (Exception e)
      throw new Exception("Internal Error - " + e);
    return act;
  }
  public String[] checkMail(String []ms) throws Exception {
    List<String> ret = null;
    try {
      if (ms == null || ms.length == 0)
        ret = methods.getMessages(id);
      else ret = methods.checkMail(id, ms);
    } catch (Exception e)
      throw new Exception("Internal Error - " + e);
```

```
    if (ret == null) return null;
76    return ret.toArray(new String[0]);
    }
78 }
```

Arquivo responsável pelo acesso ao ambiente: xmlrpc/EnvironmentClient.java

```
  package maro.xmlrpc;
2 import redstone.xmlrpc.XmlRpcProxy;
  import redstone.xmlrpc.XmlRpcFault;
4 import redstone.xmlrpc.XmlRpcArray;
  import java.util.Random;
6 import java.net.URL;
  import java.util.ArrayList;
8 import java.util.List;
  public class EnvironmentClient {
10   protected Random random;
     protected EM methods;
12   private EnvironmentClient() { /* denied */ }
     public EnvironmentClient(String uri, String cipher) throws Exception {
14     if (uri == null) throw new Exception("URI␣is␣nulled");
       methods
16       = (EM)XmlRpcProxy.createProxy(new URL(uri), new Class[] {EM.class}, true);
       if (methods == null) throw new Exception("Cannot␣create␣proxy␣to␣methods...");
18     random = new Random();
       connect();
20   }
     protected void connect() throws Exception {
22     // step 1 - handshake
       String challenge;
24     int ret;
       int value;
26     try {
         value = random.nextInt(32768); // n, par / 2 ou (impar * 3) + 1
28       challenge = new String("i" + Integer.toString(value));
         ret = methods.handshake( challenge );
30     } catch (Exception e) { // se falhou no decodificar desafio falhou!
         throw new Exception("Web␣Server␣is␣not␣trusted␣-␣" + e);
32     }
       try {
34       int resp = 0;
         if ((value % 2) == 0) resp = value / 2;
36       else resp = 1 + 3 * value;

38       // Quando a resposta do desafio veio errada falhou tb!
         // Entretanto, divide o try pra deixar melhor localizado
40       if (ret < 0 || ret != resp) throw new Exception();
       } catch (Exception e) {
42       throw new Exception("Web␣Server␣is␣not␣trusted␣-␣Response␣to␣Challenge␣is␣
             Wrong");
       }
44     // step 2 - login
       login();
46   }
     protected void login() throws Exception {
48     return ; // the server not need this
     }
50   public void logout() throws Exception {
       return ; // the server not need this
52   }
     // [ [functor, [term], [annot]] ]
54   private Object[] removeXRA(XmlRpcArray xra) throws Exception {
       List<Object> lo = null;
56     if (xra == null) return null;
       lo = new ArrayList<Object>();
58     for (Object o: xra) {
         if (o instanceof XmlRpcArray) {
60         List<Object> elem = new ArrayList<Object>();
           for (Object e: ((XmlRpcArray) o)) {
62           if (elem.size() >= 3)
               throw new Exception("Tuple␣perception␣is␣not␣correct!");
64           if (e instanceof XmlRpcArray) {
               if (elem.size() == 2)
```

```
66              elem.add( removeXRA((XmlRpcArray)e) );
              else if (elem.size() == 1)
68              elem.add( ((XmlRpcArray)e).toArray() );
              else throw new Exception("Tuple␣percerption␣is␣invalid!");
70          } else elem.add( e );
        }
72        lo.add( elem.toArray() );
      } else
74        throw new Exception("Tuple␣perception␣should␣be␣array␣or␣list");
    }
76    return lo.toArray();
  }
78  public Object[] doPercepts(String agName) throws Exception {
    try {
80      Object[] ret = null;
      if ( methods.havePercepts(agName) ) {
82        XmlRpcArray xra = (XmlRpcArray) methods.getPercepts(agName, true);
        if (xra != null) ret = removeXRA(xra);
84      }
      return ret;
86    } catch (Exception e) {
      e.printStackTrace();
88    throw new Exception ("Internal␣Error␣-␣" + e);
    }
90  }
  public Object[] getPercepts(String agName) throws Exception {
92  try {
      Object[] ret = null;
94    XmlRpcArray xra = (XmlRpcArray) methods.getPercepts(agName, false);
      if (xra != null) ret = removeXRA(xra);
96    return ret;
    } catch (Exception e) {
98    throw new Exception ("Internal␣Error␣-␣" + e);
    }
100 }
  public boolean addPercept(Object ld, String agName) throws Exception {
102 try {
      boolean ret;
104   if (agName == null || agName.isEmpty())
        ret = methods.addPercept(ld);
106   else
        ret = methods.addPerceptLocal(agName, ld);
108   return ret;
    } catch (Exception e)
110   throw new Exception ("Internal␣Error␣-␣" + e);
  }
112 public boolean removePercept(Object ld, String agName) throws Exception {
    try {
114   boolean ret;
      if (agName == null || agName.isEmpty())
116     ret = methods.removePercept(ld);
      else
118     ret = methods.removePerceptLocal(agName, ld);
      return ret;
120 } catch (Exception e)
      throw new Exception ("Internal␣Error␣-␣" + e);
122 }
  public boolean clearPercepts(String agName) throws Exception {
124 try {
      boolean ret;
126   if (agName == null || agName.isEmpty())
        ret = methods.clearPercepts();
128   else
        ret = methods.clearPerceptsLocal(agName);
130   return ret;
    } catch (Exception e)
132   throw new Exception ("Internal␣Error␣-␣" + e);
  }
134 public boolean executeAction(Object ld, String agName) throws Exception {
    try {
136   boolean ret = methods.performAction(agName, ld);
      return ret;
138 } catch (Exception e)
```

```
        throw new Exception ("Internal Error - " + e);
140   }
  }
```

Arquivo referente ao ambiente Jason: Environment/EnvXmlRpc.java

```
 1 package maro.Environment;
   import maro.xmlrpc.EnvironmentClient;
 3 import maro.xmlrpc.Perceive;
   import jason.environment.Environment;
 5 import jason.asSyntax.Literal;
   import jason.asSyntax.Structure;
 7 import java.util.ArrayList;
   import java.util.List;
 9 public class EnvXmlRpc extends Environment {
     protected EnvironmentClient ec;
11   protected String secret;
     protected String url;
13   private EnvironmentClient getClient() throws Exception {
       if (ec == null) ec = new EnvironmentClient(url, secret);
15     return ec;
     }
17   public void init(String[] args) {
       url = null;
19     secret = null;
       if (args.length == 1) {
21       url = args[0];
       } else if (args.length == 2) {
23       url = args[0];
         secret = args[1];
25     } else {
         //throw new Exception("Please, inform a URL and, optionaly, a secret");
27     }
       try {
29       EnvironmentClient e = getClient();
         if (e == null) throw new Exception("Unknow error");
31     } catch (Exception ex) {
         System.err.println("Error connecting to environment: " + ex);
33       ex.printStackTrace();
       }
35   }
     public List<Literal> getPercepts(String agName) {
37     List<Literal> ll = new ArrayList<Literal>();
       try {
39       Object[] lo = getClient().doPercepts(agName);
         if (lo == null) return null;
41       ll.addAll( Perceive.undump(lo) );
       } catch (Exception ex) {
43       System.err.println("Error in getPercepts: " + ex);
         ex.printStackTrace();
45       return null;
       }
47     return ll;
     }
49   public List<Literal> consultPercepts(String agName) {
       List<Literal> ll = new ArrayList<Literal>();
51     try {
         Object[] lo = getClient().getPercepts(agName);
53       if (lo == null) return null;
         ll.addAll( Perceive.undump(lo) );
55     } catch (Exception ex) {
         System.err.println("Error in consultPercepts: " + ex);
57       ex.printStackTrace();
         return null;
59     }
       return ll;
61   }
     public void addPercept(Literal per) {
63     try {
         boolean isOK = getClient().addPercept( Perceive.dumpLiteral(per), null );
65       if (isOK == false) throw new Exception("Error in addPercept global");
       } catch (Exception ex) {
67       System.err.println("Error in addPercept: " + ex);
```

```
      ex.printStackTrace();
69    }
    }
71  public void addPercept(String agName, Literal per) {
      try {
73      boolean isOK = getClient().addPercept( Perceive.dumpLiteral(per), agName );
        if (isOK == false) throw new Exception("Error␣in␣addPercept␣local");
75    } catch (Exception ex) {
        System.err.println("Error␣in␣addPercept:␣" + ex);
77      ex.printStackTrace();
      }
79  }
    public boolean removePercept(Literal per) {
81    try {
        boolean isOK
83        = getClient().removePercept( Perceive.dumpLiteral(per), null );
        if (isOK == false) throw new Exception("Error␣in␣removePercept␣global");
85      return isOK;
      } catch (Exception ex) {
87      System.err.println("Error␣in␣removePercept:␣" + ex);
        ex.printStackTrace();
89    }
      return false;
91  }
    public boolean removePercept(String agName, Literal per) {
93    try {
        boolean isOK
95        = getClient().removePercept( Perceive.dumpLiteral(per), agName );
        if (isOK == false) throw new Exception("Error␣in␣removePercept␣local");
97      return isOK;
      } catch (Exception ex) {
99      System.err.println("Error␣in␣removePercept:␣" + ex);
        ex.printStackTrace();
101   }
      return false;
103 }
    public void clearPercepts() {
105   try {
        boolean isOK = getClient().clearPercepts( null );
107     if (isOK == false) throw new Exception("Error␣in␣clearPercepts␣global");
      } catch (Exception ex) {
109     System.err.println("Error␣in␣clearPercepts:␣" + ex);
        ex.printStackTrace();
111   }
    }
113 public void clearPercepts(String agName) {
      try {
115     boolean isOK = getClient().clearPercepts( agName );
        if (isOK == false) throw new Exception("Error␣in␣clearPercepts␣local");
117   } catch (Exception ex) {
        System.err.println("Error␣in␣clearPercepts:␣" + ex);
119     ex.printStackTrace();
      }
121 }
    public boolean executeAction(String agName, Structure act) {
123   try {
        boolean isOK
125       = getClient().executeAction(Perceive.dumpLiteral((Literal)act), agName);
        if (isOK == false) throw new Exception("Error␣in␣action␣from␣" + agName);
127     return isOK;
      } catch (Exception ex) {
129     System.err.println("Error␣in␣action␣from␣" + agName + ":␣" + ex);
        ex.printStackTrace();
131   }
      return false;
133 }
    /* As funcoes abaixo foram consideradas irelevantes para o TI
135    e podem ser desenvolvidas no futuro sob-demanda. */
    public int removePerceptsByUnif(Literal per) {
137   System.err.println("removePerceptsByUnif/1␣do␣not␣use!!");
      return 0;
139 }
    public int removePerceptsByUnif(String agName, Literal per) {
```

```
141     System.err.println("removePerceptsByUnif/2 do not use!!");
        return 0;
143   }
      public boolean containsPercept(Literal per) {
145     System.err.println("containsPercept/2 do not use!!");
        return false;
147   }
      public boolean containsPercept(String agName, Literal per) {
149     System.err.println("containsPercept/2 do not use!!");
        return false;
151   }
    }
```

Arquivo referente à arquitetura do Agente Jason: architecture/AgArchXmlRpc.java

```
    package maro.architecture;
2   import maro.asSemantics.AgXmlRpc;
    import maro.xmlrpc.AgentClient;
4   import maro.xmlrpc.Perceive;
    import jason.architecture.AgArch;
6   import jason.mas2j.ClassParameters;
    import jason.asSemantics.TransitionSystem;
8   import jason.asSemantics.IntendedMeans;
    import jason.asSemantics.Circumstance;
10  import jason.asSemantics.ActionExec;
    import jason.asSemantics.Intention;
12  import jason.asSemantics.Message;
    import jason.asSemantics.Unifier;
14  import jason.asSemantics.Option;
    import jason.asSemantics.Agent;
16  import jason.runtime.Settings;
    import jason.asSyntax.PlanBodyImpl;
18  import jason.asSyntax.PlanBody;
    import jason.asSyntax.Literal;
20  import jason.asSyntax.Trigger;
    import jason.asSyntax.Plan;
22  import jason.JasonException;
    import javax.management.ObjectName;
24  import java.util.List;
    public class AgArchXmlRpc extends AgArch {
26    protected String secretAgent;
      private AgentClient client; // ws handler
28    private boolean sleeping;
      public AgArchXmlRpc () {
30      super();
        client = null;
32      sleeping = true;
      }
34    private String unquoteP(Settings stts, String str) {
        String val;
36      if (str == null)  return null;
        val = stts.getUserParameter(str);
38      if (val == null)  return null;
        return ObjectName.unquote(val);
40    }
      private AgentClient getClient() throws Exception {
42      if (client == null) {
          Settings stts = getTS().getSettings();
44        String url = unquoteP(stts, "url");
          if (url == null) {
46          throw new Exception("URL is requeried to identify WS' Server!");
          }
48
          client = new AgentClient(getAgName(), url, unquoteP(stts, "secretAgent"));
50      }
        return client;
52    }
      @Override
54    public void initAg(String agClass, ClassParameters bbPars, String asSrc,
          Settings stts)
          throws JasonException {
56      try {
          // agClass, bbPars, asSrc is not used :p
```

```
58        Agent ag = new AgXmlRpc();
          TransitionSystem ts = new TransitionSystem(ag, new Circumstance(), stts,
              this);
60        ag.setBB(null);
          ag.initAg();
62        ag.setTS(ts);
        } catch (Exception e)
64        throw new JasonException("as2j: error creating the customised Agent class! -
              ", e);
    }
66  public void stopAg() {
      super.stopAg();
68    try {
        getClient().logout();
70    } catch (Exception e) {
        System.err.println("as2j: error stop agent! - " + e);
72      e.printStackTrace();
      }
74  }
    public List<Literal> perceive() {
76    try {
        List<Literal> p = super.perceive();
78      String act = getClient().perceive(Perceive.dump(p));
        if (!act.isEmpty()) addAction(act);
80    } catch (Exception e) {
        System.err.println("as2j: error in agent! - " + e);
82      e.printStackTrace();
      }
84    return null;
    }
86  private void addAction(String act) {
      Intention intention = new Intention();
88    IntendedMeans im;
      PlanBody bd;
90    Trigger te;
      Literal li;
92    Unifier un;
      Option opt;
94    Plan plan;
      // configuration... literal
96    li = Literal.parseLiteral(act);
      // configuration... trigger
98    te = new Trigger(
          Trigger.TEOperator.add,
100       Trigger.TEType.achieve,
          Literal.parseLiteral("rpc[source(self)]")
102     );
      // configuration... plan
104   bd = new PlanBodyImpl(
          PlanBody.BodyType.action,
106       li);
      plan = new Plan(null, te, null, bd);
108   // configuration... option
      un = new Unifier();
110   opt = new Option(plan, un);
      // configuration... IntendedMeans
112   im = new IntendedMeans(opt, te);
      // configuration... done!
114   intention.push(im);
      getTS().getC().setAction( new ActionExec(li, intention) );
116   sleeping = false;
    }
118 public void act(ActionExec action, List<ActionExec> feedback) {
      super.act(action, feedback);
120   sleeping = true;
    }
122 public boolean canSleep() {
      return sleeping && super.canSleep();
124 }
    public void checkMail() {
126   super.checkMail();
      try {
128     Object []os = getTS().getC().getMailBox().toArray();
```

```
        String []ms = null;
130     String []rs = null;
        if (os != null && os.length > 0) {
132       int i = 0;
          ms = new String[ os.length];
134       for (Object o: os) {
            Message m = (Message) o;
136         ms[i] = m.toString();
            i++;
138       }
          getTS().getC().getMailBox().clear();
140     }
        rs = getClient().checkMail(ms);
142     if (rs != null && rs.length > 0) {
          for (String s : rs) { // r = a message
144         if (s.isEmpty())
              continue;
146         Message m = Message.parseMsg( s );
            // Quando enviado vazio, ele nao preenche automatico
148         // por isso reconfiguramos para nulo
            if (m.getSender() != null && m.getSender().isEmpty())
150           m.setSender(null);
            if (m.getReceiver() == null || m.getReceiver().isEmpty()) {
152           broadcast(m);
            } else sendMsg(m);
154       }
        }
156   } catch (Exception e) {
        System.err.println("as2j:␣error␣checking␣Mailbox␣of␣agent!␣-␣" + e);
158     e.printStackTrace();
    }
160 }
}
```

Arquivo referente ao Raciocinador do Agente Jason: asSemantics/AgXmlRpc.java

```
1 package maro.asSemantics;
  import jason.asSemantics.Intention;
3 import jason.asSemantics.Agent;
  import jason.asSyntax.Literal;
5 import java.util.List;
  public class AgXmlRpc extends Agent {
7   /* O agente do lado Jason nao deve realizar
       nada, por isso essas duas funcoes principais
9      encontram-se vazias. */
    public void buf(List<Literal> percepts) { }
11  public List<Literal>[] brf(Literal beliefToAdd, Literal beliefToDel,  Intention
        i) {
      return null;
13  }
}
```

APÊNDICE B FONTE DO EXEMPLO *ROOM*

Arquivo de construção: bin/build.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--
                    This file was generated by Jason 1.3.2
                    http://jason.sf.net
                    Agosto 31, 2010 - 12:27:56
-->
<project name   ="room"
         basedir=".."
         default="run">
    <property name="mas2j.project.file" value="Room.mas2j"/>
    <property name="debug" value=""/> <!-- use "-debug" to run in debug mode -->
    <property name="build.dir" value="${basedir}/bin/classes" />
    <property name="jasonJar" value="${basedir}/../../maro.jar" />
    <path id="project.classpath">
        <pathelement location="${basedir}"/>
        <pathelement location="${build.dir}"/>
        <pathelement location="${jasonJar}"/>
        <fileset dir="${basedir}/../../lib" >  <include name="*.jar" /> </fileset>
    </path>
    <!-- tasks the user can override in his/her c-build.xml script -->
    <target name="user-init">
    </target>
    <target name="user-end">
    </target>
    <target name="init">
        <mkdir dir="${build.dir}" />
        <antcall target="user-init" />
    </target>
    <target name="compile" depends="init">
        <condition property="srcdir" value="${basedir}/src/java" else="${basedir}"
            >
            <available file="${basedir}/src/java" />
        </condition>
        <javac srcdir="${srcdir}" destdir="${build.dir}" debug="true" optimize="
            true" >
            <classpath refid="project.classpath"/>
        </javac>
    </target>
    <target name="jar" depends="compile">
        <delete file="${ant.project.name}.jar" />
        <copy file="${jasonJar}" tofile="${ant.project.name}.jar" />
        <copy file="${mas2j.project.file}" tofile="default.mas2j" />
        <jar update="yes" jarfile="${ant.project.name}.jar" >
            <fileset dir="${basedir}">
                <include name="**/*.asl" />
                <include name="**/*.mas2j" />
            </fileset>
            <fileset dir="${build.dir}">
                <include name="**/*.class" />
            </fileset>
        </jar>
        <delete file="default.mas2j" />
    </target>
    <target name="jnlp" depends="jar" >
```

```
        <mkdir dir="${basedir}/${ant.project.name}-jws"/>
54      <java classname="jason.infra.centralised.CreateJNLP"
              failonerror="true" fork="yes" dir="${basedir}/${ant.project.name}-
                  jws" >
56          <classpath refid="project.classpath"/>
            <arg line="${ant.project.name}␣${mas2j.project.file}"/>
58      </java>
        <copy todir="${basedir}/${ant.project.name}-jws" failonerror="no">
60          <fileset dir="${basedir}/lib" includes="**/*.jar" />
            <fileset dir="${basedir}" includes="${ant.project.name}.jar" />
62          <fileset dir="/Volumes/Jason/Jason-1.3.2/Jason.app/Contents/Resources/
                Java/src/images" includes="Jason-GMoreau-Icon.jpg" />
        </copy>
64      <signjar jar="${basedir}/${ant.project.name}-jws/${ant.project.name}.jar"
            alias="jason"
            storepass="rbjhja" keypass="rbjhja" keystore="/Volumes/Jason/Jason
                -1.3.2/Jason.app/Contents/Resources/Java/src/jasonKeystore" />
66      <echo message="**" />
        <echo message="**␣Java␣Web␣Start␣application␣created␣in␣directory␣${ant.
            project.name}-jws" />
68      <echo message="**␣Update␣the␣codebase␣(in␣the␣second␣line␣of␣the␣.jnlp␣
            file)" />
        <echo message="**␣with␣the␣URL␣where␣you␣will␣upload␣the␣application." />
70      <echo message="**" />
    </target>
72  <target name="run" depends="compile" >
        <echo message="Running␣project␣${ant.project.name}" />
74      <java classname="jason.infra.centralised.RunCentralisedMAS"
              failonerror="true" fork="yes" dir="${basedir}" >
76          <classpath refid="project.classpath"/>
            <arg line="${mas2j.project.file}␣${debug}␣"/>
78          <jvmarg line="-Xmx500M␣-Xss8M"/>
        </java>
80      <antcall target="user-end" />
    </target>
82  <target name="clean" >
        <delete failonerror="no" includeEmptyDirs="true" verbose="true">
84          <fileset dir="${basedir}" includes="**/*.class"/>
        </delete>
86  </target>
</project>
```

Arquivo de projeto: Room.mas2j

```
1 // Adaptado para usar os agentes em servicos web
  MAS room {
3       infrastructure: Centralised
        environment: RoomEnv
5       executionControl: jason.control.ExecutionControl

7       agents:
                porter
9                       [url="http://localhost:8080"]
                        agentArchClass maro.architecture.AgArchXmlRpc
11                      ;
                claustrophobe;
13              paranoid
                        [url="http://localhost:8081"]
15                      agentArchClass maro.architecture.AgArchXmlRpc
                        ;
17 }
```

Arquivo do ambiente: RoomEnv.java

```
  import jason.asSyntax.Literal;
2 import jason.asSyntax.Structure;
  import jason.environment.Environment;
4
  public class RoomEnv extends Environment {
6     Literal ld  = Literal.parseLiteral("locked(door)");
      Literal nld = Literal.parseLiteral("~locked(door)");
8     boolean doorLocked = true;
      @Override
```

```
10      public void init(String[] args) {
            addPercept(ld);
12      }
        @Override
14      public boolean executeAction(String ag, Structure act) {
                System.out.println("Agent␣"+ag+"␣is␣doing␣"+act);
16              clearPercepts();
                if (act.getFunctor().equals("lock"))
18                      doorLocked = true;
                if (act.getFunctor().equals("unlock"))
20                      doorLocked = false;
                if (doorLocked)  addPercept(ld);
22              else  addPercept(nld);
                return true;
24      }
    }
```

Arquivo do agente em ASL: claustrophobe.asl

```
    +locked(door) : true
2     <- .send(porter,achieve,~locked(door)).
```

Arquivo do agente em Python: server-paranoid.py

```
1   from SimpleXMLRPCServer import SimpleXMLRPCServer
    from random import randint
3   server = SimpleXMLRPCServer(("localhost", 8081), logRequests=False)
    server.register_introspection_functions()
5   agentL = {}
    def handshake(challenge):
7           if challenge[0] != 'i':
                    return int(-1)
9           ret = int(challenge[1:])
            ret *= ret;
11          return ret
    def login(agent_name):
13          max = 2 ** 20
            pseudo = randint(1, max)
15          try:
                    temp = agentL[pseudo]
17                  while True:
                            pseudo = (pseudo + 1) % max
19                          temp = agentL[pseudo]
            except KeyError:
21                  agentL[pseudo] = (agent_name, [], [])
                    print "Create␣", agent_name, "referenced␣by", pseudo
23          return pseudo
    def logout(id):
25          try:
                    ag = agentL.pop(id)
27                  print "Logout␣of␣", id, "OK!"
                    return True
29          except:
                    print "Logout␣of␣", id, "failed!"
31                  return False
    def perceive(id, percepts):
33          data = agentL[id]
            agentL[id] = (data[0], percepts, data[2])
35          return act2(id, False)
    def act2(id, eraseAll):
37          if eraseAll:
                    data = agentL[id]
39                  agentL[id] = (data[0], [], data[2])
            data = agentL[id]
41          if len(data[1]) > 0:
                    sendMessage = exclusion_act(data[1])
43                  if len(sendMessage) > 0:
                            data[2].append(sendMessage)
45                          agentL[id] = (data[0], data[1], data[2])
            return ""
47  def act(id, eraseAll):
            if eraseAll:
49                  return act2(id, eraseAll)
```

```
        return ""
def exclusion_act(perceptions):
        p = perceptions[0]
        functor = p[0]
        terms = p[1]
        annots = p[2]
        if functor == "~locked" and terms[0] == "door":
                try:
                        msgId = agentL["msgId"]
                except KeyError:
                        msgId = 0
                agentL["msgId"] = msgId + 1
                return "<mprpc%02d,,achieve,porter,locked(door)>" % msgId
        return ""
def receiveMessages(id, messages):
        return sendMessages(id)
def sendMessages(id):
        data = agentL[id]
        ret = []
        if len(data[2]) > 0:
                ret = data[2]
                agentL[id] = (data[0], data[1], [])
        return ret
server.register_function(handshake, "PM.handshake")
server.register_function(login, "PM.login")
server.register_function(logout, "PM.logout")
server.register_function(perceive, "PM.perceive")
server.register_function(act, "PM.act")
server.register_function(receiveMessages, "PM.checkMail")
server.register_function(sendMessages, "PM.getMessages")
server.serve_forever();
```

Arquivo do agente em Python: server-porter.py

```
from SimpleXMLRPCServer import SimpleXMLRPCServer
from random import randint
server = SimpleXMLRPCServer(("localhost", 8080), logRequests=False)
server.register_introspection_functions()
agentL = {}
def handshake(challenge):
        if challenge[0] != 'i':
                return int(-1)
        ret = int(challenge[1:])
        ret *= ret;
        return ret
def login(agent_name):
        max = 2 ** 20
        pseudo = randint(1, max)
        try:
                temp = agentL[pseudo]
                while True:
                        pseudo = (pseudo + 1) % max
                        temp = agentL[pseudo]
        except KeyError:
                agentL[pseudo] = (agent_name, [], [])
                print "Create ", agent_name, "referenced by", pseudo
        return pseudo
def logout(id):
        try:
                ag = agentL.pop(id)
                print "Logout of ", id, "OK!"
                return True
        except:
                print "Logout of ", id, "failed!"
                return False
def perceive(id, percepts):
        data = agentL[id]
        agentL[id] = (data[0], percepts, data[2])
        return act(id, False)
def act(id, eraseAll):
        if eraseAll:
                data = agentL[id]
                agentL[id] = (data[0], [], data[2])
        data = agentL[id]
        if len(data[2]) >= 1:
                ret = exclusion_act(data[1], data[2][0])
                if len(ret) > 0:
                        agentL[id] = (data[0], data[1], data[2][1:])
                return ret
        return ""
def exclusion_act(perceptions, message_received):
        M = { "~locked(door)":"locked(door)", "locked(door)":"~locked(door)" }
        P = { "~locked":"lock", "locked":"unlock" }
        test = M[ message_received[4] ][:-1].split('(')
        for p in perceptions:
                functor = p[0]
                terms = p[1]
                annots = p[2]
                if functor == test[0] and terms[0] == test[1]:
                        return P[functor]
        return ""
def receiveMessages(id, messages):
        data = agentL[id]
        msgL = []
        for msg in messages:
                m = msg[1:-1].split(',')
                if m[1] == "claustrophobe" and m[4] == "~locked(door)":
                        msgL.append(m);
                elif m[1] == "paranoid" and m[4] == "locked(door)":
                        msgL.append(m);
        agentL[id] = (data[0], data[1], msgL)
        return sendMessages(id)
def sendMessages(id):
        return []
server.register_function(handshake, "PM.handshake")
```

```
server.register_function(login, "PM.login")
server.register_function(logout, "PM.logout")
server.register_function(perceive, "PM.perceive")
server.register_function(act, "PM.act")
server.register_function(receiveMessages, "PM.checkMail")
server.register_function(sendMessages, "PM.getMessages")
server.serve_forever();
```

APÊNDICE C FONTE DO EXEMPLO *GAME OF LIFE*

Arquivo de construção: bin/build.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--
                    This file was generated by Jason 1.3.2
                    http://jason.sf.net

                    Agosto 14, 2010 - 23:33:04
-->
<project name   ="game_of_life"
         basedir=".."
         default="run">
    <property name="mas2j.project.file" value="game-of-life.mas2j"/>
    <property name="debug" value=""/> <!-- use "-debug" to run in debug mode -->
    <property name="build.dir" value="${basedir}/bin/classes" />
    <property name="jasonJar" value="${basedir}/../../maro.jar" />
    <path id="project.classpath">
        <pathelement location="${basedir}"/>
        <pathelement location="${build.dir}"/>
        <pathelement location="${jasonJar}"/>
        <fileset dir="${basedir}/../../lib" >  <include name="*.jar" /> </fileset>
    </path>
    <!-- tasks the user can override in his/her c-build.xml script -->
    <target name="user-init">
    </target>
    <target name="user-end">
    </target>
    <target name="init">
        <mkdir dir="${build.dir}" />
        <antcall target="user-init" />
    </target>
    <target name="compile" depends="init">
        <condition property="srcdir" value="${basedir}/src/java" else="${basedir}"
             >
            <available file="${basedir}/src/java" />
        </condition>
        <javac srcdir="${srcdir}" destdir="${build.dir}" debug="true" optimize="
            true" >
            <classpath refid="project.classpath"/>
        </javac>
    </target>
    <target name="jar" depends="compile">
        <delete file="${ant.project.name}.jar" />
        <copy file="${jasonJar}" tofile="${ant.project.name}.jar" />
        <copy file="${mas2j.project.file}" tofile="default.mas2j" />
        <jar update="yes" jarfile="${ant.project.name}.jar" >
            <fileset dir="${basedir}">
                <include name="**/*.asl" />
                <include name="**/*.mas2j" />
            </fileset>
            <fileset dir="${build.dir}">
                <include name="**/*.class" />
            </fileset>
        </jar>
        <delete file="default.mas2j" />
    </target>
```

```
    <target name="jnlp" depends="jar" >
54      <mkdir dir="${basedir}/${ant.project.name}-jws"/>
        <java classname="jason.infra.centralised.CreateJNLP"
56            failonerror="true" fork="yes" dir="${basedir}/${ant.project.name}-
                  jws" >
            <classpath refid="project.classpath"/>
58          <arg line="${ant.project.name} ${mas2j.project.file}"/>
        </java>
60      <copy todir="${basedir}/${ant.project.name}-jws" failonerror="no">
            <fileset dir="${basedir}/lib" includes="**/*.jar" />
62          <fileset dir="${basedir}" includes="${ant.project.name}.jar" />
            <fileset dir="/Volumes/Jason/Jason-1.3.2/Jason.app/Contents/Resources/
                Java/src/images" includes="Jason-GMoreau-Icon.jpg" />
64      </copy>
        <signjar jar="${basedir}/${ant.project.name}-jws/${ant.project.name}.jar"
            alias="jason"
66          storepass="rbjhja" keypass="rbjhja" keystore="/Volumes/Jason/Jason
                -1.3.2/Jason.app/Contents/Resources/Java/src/jasonKeystore" />
        <echo message="**" />
68      <echo message="** Java Web Start application created in directory ${ant.
            project.name}-jws" />
        <echo message="** Update the codebase (in the second line of the .jnlp
            file)" />
70      <echo message="** with the URL where you will upload the application." />
        <echo message="**" />
72  </target>
    <target name="run" depends="compile" >
74      <echo message="Running project ${ant.project.name}" />
        <java classname="jason.infra.centralised.RunCentralisedMAS"
76            failonerror="true" fork="yes" dir="${basedir}" >
            <classpath refid="project.classpath"/>
78          <arg line="${mas2j.project.file} ${debug} "/>
            <jvmarg line="-Xmx500M -Xss8M"/>
80      </java>
        <antcall target="user-end" />
82  </target>
    <target name="clean" >
84      <delete failonerror="no" includeEmptyDirs="true" verbose="true">
            <fileset dir="${basedir}" includes="**/*.class"/>
86      </delete>
    </target>
88 </project>
```

Arquivo de projeto: Game-of-life.mas2j

```
MAS game_of_life {
2   infrastructure: Centralised(pool,2)
    environment: maro.Environment.EnvXmlRpc("http://localhost:8079")
4   executionControl: jason.control.ExecutionControl

6   agents:
      cell
8     [url="http://localhost:8080/RPC2"]
      agentArchClass maro.architecture.AgArchXmlRpc
10    #3600 // matrix 60x60
      ;
12 }
```

Arquivo de agente: server-cell.py

```
from SimpleXMLRPCServer import SimpleXMLRPCServer
2 from random import randint
server = SimpleXMLRPCServer(("localhost", 8080))
4 server.register_introspection_functions()
agentL = {}
6 def handshake(challenge):
        if challenge[0] != 'i':
8               return int(-1)
        ret = int(challenge[1:])
10      ret *= ret;
        return ret
12 def login(agent_name):
        max = 2 ** 20
```

```
        pseudo = randint(1, max)
        try:
                temp = agentL[pseudo]
                while True:
                        pseudo = (pseudo + 1) % max
                        temp = agentL[pseudo]
        except KeyError:
                agentL[pseudo] = (agent_name, [])
        return pseudo
def logout(id):
        try:
                ag = agentL.pop(id)
                return True
        except:
                return False
def perceive(id, percepts):
        data = agentL[id]
        agentL[id] = (data[0], percepts)
        return act(id, False)
def act(id, eraseAll):
        if eraseAll:
                data = agentL[id]
                agentL[id] = (data[0], [])
        data = agentL[id]
        actions = { 0: "die", 2: "skip", 3:"live" }
        return actions[aliveNeighbors(data[1])]
def aliveNeighbors(perceptArray):
        valid = [ 2, 3 ]
        for elem in perceptArray:
                if "alive_neighbors" == elem[0]:
                        for v in valid:
                                if v == int(elem[1][0]):
                                        return v
        return 0
def receiveMessages(id, messages):
        print "self", messages
        return sendMessages(id)
def sendMessages(id):
        return []
server.register_function(handshake, "PM.handshake")
server.register_function(login, "PM.login")
server.register_function(logout, "PM.logout")
server.register_function(perceive, "PM.perceive")
server.register_function(act, "PM.act")
server.register_function(receiveMessages, "PM.checkMail")
server.register_function(sendMessages, "PM.getMessages")
server.serve_forever();
```

Arquivo de ambiente: hs/Environment.hs

```
module Main
where
import Control.Concurrent ( forkIO )
import Control.Concurrent.MVar
import LuccaDisplay
import LuccaModel
import LuccaWS
-- no ghc passar
--    -threaded
--    -hide-package haxr-th-3000.5
--    -hide-package monads-fd
main = do
    conf <- setup 8079
    model <- newModel 60
    control <- newEmptyMVar
    display <- createDisplay control
    forkIO $ serving conf control display
    putMVar control model -- send the notify to serving start
    model' <- takeMVar control -- when this finish the serving start end
    putMVar control model' -- and we need put the new model again
    renderize display
```

Arquivo de modelo do ambiente: hs/LuccaModel.hs

```
module LuccaModel (
    Model, SyncModel,
    newModel, cleanModel, realize, viewerRealizeByPos, getTable, lock, unlock,
        getStep,
    getNeighbors
) where
import Control.Concurrent.MVar
import Data.List.Ordered
import Control.Monad (when, fail)
import System.Random
import Data.IORef
import Data.Array
import LuccaData

newModel :: Int -> IO Model
newModel n = do
    let
        start = 1
        end = n*n
        valueDefault = False
    l <- randomizeData [start..end] 15
    t <- newIORef $ array(start,end) l
    u <- newIORef []
    r <- newIORef 0
    return $ Model end t u r
rollDice :: Int -> Int -> IO Int
rollDice s e = getStdRandom (randomR (s,e))
randomizeData :: [Int] -> Int -> IO [(Int, Bool)]
randomizeData [] percentageTrue = return []
randomizeData (key:keys) percentageTrue = do
    limited <- rollDice 1 100
    let value = if (limited < percentageTrue) then True else False
    ret <- randomizeData keys percentageTrue
    return $ (key,value):ret
lock :: SyncModel -> IO Model
lock = takeMVar
unlock :: SyncModel -> Model -> IO ()
unlock = putMVar
-- Model deve ser o valor retirado do SyncModel
getTable :: Model -> IO [Bool]
getTable model = do
    table' <- readIORef $ table model
    return $ elems table'
getTableSize :: Model -> (Int, Int)
getTableSize model = (1, tableSize model)
-----------------------------------------
-- This is only executed in server thread
-----------------------------------------
realize :: SyncModel -> String -> Maybe Bool -> IO Bool
realize controller name flag = do
    model <- lock controller
    let n = (read (drop 4 name)) :: Int
    updateCellList model n
    realize2 model n flag
    bool <- compareLengthCellList model
    unlock controller model
    return bool
realize2 :: Model -> Int -> Maybe Bool -> IO ()
realize2 model cell Nothing = return () -- ignore. Do nothing
realize2 model cell (Just flag) = do -- the cell can be activate or deactive
    t <- readIORef $ table model
    let t' = t // [ (cell, flag) ]
    modifyIORef (table model) $ \_ -> t'
    return ()
updateCellList :: Model -> Int -> IO ()
updateCellList model cell = do
    u <- readIORef $ acted model
    let u' = insertSet cell u
    modifyIORef (acted model) $ \_ -> u'
compareLengthCellList :: Model -> IO Bool
compareLengthCellList model = do
```

```
    t <- readIORef $ acted model
    return $ (tableSize model) == (length t)
cleanModel :: Model -> IO ()
cleanModel model = do
    modifyIORef (acted model) $ \_ -> [] -- cleaning notified agents!
    s <- readIORef $ step model
    modifyIORef (step model) $ \s -> (s + 1)
    return ()
getStep :: Model -> IO Int
getStep model = readIORef $ step model
-----------------------------------------------
-- This point is executed in Display Thread
-----------------------------------------------
viewerRealizeByPos :: Model -> Int -> IO ()
viewerRealizeByPos model pos = do
    flag <- realize2 model pos (Just True)
    return ()
getNeighbors :: Model -> IO [(String,Int)]
getNeighbors model = do
    elems <- getTable model
    let
        limits@(start,end) = getTableSize model
        line = truncate $ sqrt $ fromIntegral end
    return $ getNeighbors2 limits line elems start
getNeighbors2 :: (Int,Int) -> Int -> [Bool] -> Int -> [(String,Int)]
getNeighbors2 limits@(_,end) nline elems cell =
    let
        cnl = check_neighbors_live limits nline elems cell
        next = getNeighbors2 limits nline elems (cell + 1)
        name = "cell" ++ show cell
    in if cell == end then
        [(name, cnl)]
    else
        [(name, cnl)] ++ next
check_neighbors_live :: (Int,Int) -> Int -> [Bool] -> Int -> Int
check_neighbors_live limits@(_,end) nline elems cell =
    alive_neighbors
    where
        alive_neighbors = check idx [] elems
        idx = limitedIdx limits mapIdx
        mapIdx = [linePS..(linePS+2)] ++ [lineAS..(lineAS+2)] ++ [(cell-1),(cell
            +1)]
        linePS = cell - 1 - nline
        lineAS = cell - 1 - nline
check :: [Int] -> [Bool] -> [Bool] -> Int
check [] bs _ = countTrue bs
check (idx:idxs) bs elems =
    let
        el = take 1 $ drop (idx-1) elems
        bs' = el ++ bs
    in check idxs bs' elems
limitedIdx :: (Int, Int) -> [Int] -> [Int]
limitedIdx _ [] = []
limitedIdx l@(s,e) (i:is) =
    if i >= 1 && i <= e then
        [i] ++ limitedIdx l is
    else
        limitedIdx l is
countTrue :: [Bool] -> Int
countTrue [] = 0
countTrue (True:bs) = 1 + countTrue bs
countTrue (False:bs) = 0 + countTrue bs
```

Arquivo de dados do ambiente: hs/LuccaData.hs

```
-- {-# LANGUAGE TemplateHaskell #-}
module LuccaData where
import Control.Concurrent.MVar
import Data.Array
import Data.IORef
import Network.XmlRpc.Internals
import Qtc.ClassTypes.Opengl
import Qtc.ClassTypes.Gui
```

```
import Qtc.Core.Base
import Qtc.Gui.QWidget
import Qtc.Opengl.QGLWidget
-- Utilizado no Model
data Model = Model {
        tableSize :: Int, -- n x n
        table :: IORef (Array Int Bool),
        acted :: IORef [Int],
        step :: IORef Int
    }
type SyncModel = MVar Model
-- Utilizado no Display
type MyQWidget = QWidgetSc (CMyQWidget)
data CMyQWidget = CMyQWidget
type GLWidget = QGLWidgetSc (CGLWidget)
data CGLWidget = CGLWidget

-- Utilizado em WS
type Action = Percept
data Percept = Percept {
        functor::String,
        params::[String],
        annots::[Percept]
    }
    deriving (Show,Eq,Ord)
data TPerception = Perceptions {
        global::[Percept],
        local::[(String, Percept)],
        updated::[String]
    }
    deriving (Show)
instance XmlRpcType Percept where
    toValue p = toValue $ [toValue (functor p), toValue (params p), toValue (
        annots p)]
    fromValue v = do
        (ValueArray (fu:pa:an:[])) <- fromValue v
        fu' <- fromValue fu
        pa' <- fromValue pa
        an' <- fromValue an
        return $ Percept fu' pa' an'
    getType _ = TArray
-- Nao deveria haver codigo aqui...
myQWidget :: IO (MyQWidget)
myQWidget = qSubClass (qWidget ())
gLWidget :: IO (GLWidget)
gLWidget = qSubClass (qGLWidget ())
updateDisplay :: GLWidget -> IO ()
updateDisplay w = do
    updateGL w ()
```

Arquivo do mostrador do ambiente: hs/LuccaDisplay.hs

```
{-# OPTIONS -fglasgow-exts #-}
module LuccaDisplay (
    GLWidget,
    renderize, createDisplay, updateDisplay
) where
import Qtc.Classes.Qccs
import Qtc.Classes.Qccs_h
import Qtc.Classes.Gui
import Qtc.ClassTypes.Opengl
import Qtc.ClassTypes.Gui
import Qtc.ClassTypes.Core
import Qtc.Core.Base
import Qtc.Gui.Base
import Qtc.Core.QSize
import Qtc.Gui.QApplication
import Qtc.Gui.QHBoxLayout
import Qtc.Gui.QWidget
import Qtc.Gui.QColor
import Qtc.Gui.QMouseEvent
import Qtc.Opengl.QGLWidget
import Qtc.Opengl.QGLWidget_h
```

```
22 import Data.IORef
   import Graphics.Rendering.OpenGL as GL
24 import LuccaModel
   import LuccaData
26
   createDisplay :: SyncModel -> IO GLWidget
28 createDisplay model = do
       app <- qApplication ()
30     w <- gLWidget
       pos <- newIORef (0,0,False)
32     sz <- newIORef((0::GLdouble),(0::GLdouble))
       setHandler w "initializeGL()" $ initgl pos sz model
34     setHandler w "(QSize)minimumSizeHint()" $ minSizeHint
       setHandler w "(QSize)sizeHint()" $ szHint
36     setHandler w "mousePressEvent(QMouseEvent*)" $ msPressEvent pos
       return w
38 renderize :: GLWidget -> IO ()
   renderize w = do
40     root <- myQWidget
       layout <- qHBoxLayout ()
42     addWidget layout w
       setLayout root layout
44     qshow root ()
       ok <- qApplicationExec ()
46     returnGC
   msPressEvent :: IORef(Int, Int, Bool) -> GLWidget -> QMouseEvent () -> IO ()
48 msPressEvent pos this mev = do
       mx <- qx mev ()
50     my <- qy mev ()
       modifyIORef pos $ \(px, py, _) -> (mx, my, True)
52     updateGL this ()
   minSizeHint :: GLWidget -> IO (QSize ())
54 minSizeHint _ = qSize (100::Int, 100::Int)
   szHint :: GLWidget -> IO (QSize ())
56 szHint _ = qSize (600::Int, 600::Int)
   initgl :: IORef(Int, Int, Bool) -> IORef(GLdouble, GLdouble) -> SyncModel ->
       GLWidget -> IO ()
58 initgl pos sz model this =
       do
60     tp <- qColorFromRgbF (0.0::Double, 0.0::Double, 0.0::Double, 0.0::Double)
       qglClearColor this tp
62     shadeModel $= Flat
       depthFunc $= Just Less
64     cullFace $= Just Back
       setHandler this "resizeGL(int,int)" $ rsz sz
66     setHandler this "paintGL()" $ dsply pos sz model
       return ()
68 rsz :: IORef(GLdouble, GLdouble) -> GLWidget -> Int -> Int -> IO ()
   rsz sz this x y
70   = do
       let
72         side = min x y
           mx = (fromIntegral x / 60) :: GLdouble
74         my = (fromIntegral y / 60) :: GLdouble
       GL.viewport $= (Position 0 0, GL.Size (fromIntegral x) (fromIntegral y))
76     matrixMode $= Projection
       loadIdentity
78     ortho 0 (fromIntegral x) (fromIntegral y) 0 1.0 10.0
       matrixMode $= Modelview 0
80     modifyIORef sz $ \(sx, sy) -> (mx, my)
       return ()
82 updateModel :: Model -> IORef (Int, Int, Bool) -> IORef(GLdouble, GLdouble) -> IO
       ()
   updateModel model pos size = do
84     (rx, ry, flag) <- readIORef pos
       (sx, sy) <- readIORef size
86     modifyIORef pos $ \(rx,ry,_) -> (rx,ry,False)
       if flag == True then
88         let
               x = (fromIntegral rx) / sx
90             y = (fromIntegral ry) / sy
               v = (truncate x) + (truncate y) * 60
92         in viewerRealizeByPos model (v + 1)
```

```
     else
94         return ()
drawElem :: Int -> [Bool] -> GLdouble -> GLdouble -> IO ()
96 drawElem _ [] _ _ = return ()
drawElem current (x:xs) sx sy  = do
98     let
           line = div current 60
100        column = mod current 60
           x1 = sx * (fromIntegral column) :: GLdouble
102        x2 = x1 + sx
           y1 = sy * (fromIntegral line) :: GLdouble
104        y2 = y1 + sy
       drawElem (current + 1) xs sx sy
106    if x == True then
           renderPrimitive TriangleStrip $ do
108            GL.color $ Color3 (1.0::GLfloat) (1.0::GLfloat) (1.0::GLfloat)
               vertex $ Vertex3 x1 y1 (-5.0)
110            vertex $ Vertex3 x2 y1 (-5.0)
               vertex $ Vertex3 x2 y2 (-5.0)
112            vertex $ Vertex3 x1 y2 (-5.0)
               vertex $ Vertex3 x1 y1 (-5.0)
114            vertex $ Vertex3 x2 y1 (-5.0)
               return ()
116     else
           return ()
118 drawModel :: Model -> IORef(GLdouble, GLdouble) -> IO ()
drawModel model size = do
120    (sx,sy) <- readIORef size
       elements <- getTable model
122    drawElem 0 elements sx sy
       returnGC
124 dsply :: IORef (Int, Int, Bool) -> IORef(GLdouble, GLdouble) -> SyncModel ->
       GLWidget -> IO ()
dsply pos sz model this = do
126    model' <- lock model
       GL.clear [ ColorBuffer, DepthBuffer ]
128    loadIdentity
       updateModel model' pos sz -- liberar o nao o usuario ficar clicando
130    drawModel model' sz
       unlock model model'
132    returnGC
```

Arquivo do receptor do ambiente: hs/LuccaWS.hs

```
module LuccaWS (
2      serving,
       setup
4  ) where
import Network.Socket
6  import Network.XmlRpc.Server
import Network.XmlRpc.Internals
8  import Data.IORef
import Data.List.Ordered
10 import Happstack.Server.SimpleHTTP
import Control.Monad.Trans
12 import Data.ByteString.Lazy.Char8 (unpack)
import LuccaModel
14 import LuccaData
handshake :: String -> IO Int
16 handshake s =
       if key == 'i' then
18         return result
       else
20         return $ -1
       where
22         key = head s
           value = read (tail s) :: Int
24         m = mod value 2
           result =
26             if m == 0 then
                   truncate $ fromIntegral(value) / 2
28             else
                   1 + value * 3
```

```
addPercept :: IORef(TPerception) -> Percept -> IO Bool
addPercept p perception = do
    (Perceptions gp lp ua) <- readIORef p
    let gp' = insertSet perception gp
    modifyIORef p $ \(Perceptions _ lp _) -> (Perceptions gp' lp [])
    return True
addPerceptLocal :: IORef(TPerception) -> String -> Percept -> IO Bool
addPerceptLocal p name perception = do
    (Perceptions gp lp ua) <- readIORef p
    let lp' = insertSet (name,perception) lp
    modifyIORef p $ \(Perceptions gp _ _) -> (Perceptions gp lp' [])
    return True
havePercepts :: IORef(TPerception) -> String -> IO Bool
havePercepts p name = do
    pp <- readIORef p
    return $ not $ (updated pp) `has` name
getPercepts :: IORef(TPerception) -> String -> Bool -> IO [Percept]
getPercepts p name isUpdate = do
    (Perceptions gp lp ua) <- readIORef p
    -- r eh a uniao das percepcoes de gp com as percepcoes (segundo
    -- elemento) dos dados locais filtrados pelo nome
    let r = union gp [lp' | (n, lp') <- lp, n == name]
    if isUpdate == False then
        return r
     else do
        let ua' = insertSet name ua
        modifyIORef p $ \(Perceptions gp lp _) -> (Perceptions gp lp ua')
        return r
removePercept :: IORef(TPerception) -> Percept -> IO Bool
removePercept p perception = do
    (Perceptions gp lp ua) <- readIORef p
    let gp' = gp `minus` [perception]
    modifyIORef p $ \(Perceptions _ lp _) -> (Perceptions gp' lp [])
    return True
removePerceptLocal :: IORef(TPerception) -> String -> Percept -> IO Bool
removePerceptLocal p name perception = do
    (Perceptions gp lp ua) <- readIORef p
    let lp' = lp `minus` [(name,perception)]
    modifyIORef p $ \(Perceptions gp _ _) -> (Perceptions gp lp' [])
    return True
clearPercepts :: IORef(TPerception) -> IO Bool
clearPercepts p = do
    (Perceptions gp lp ua) <- readIORef p
    if (null gp) then
        return True
     else do
        modifyIORef p $ \(Perceptions _ lp _) -> (Perceptions [] lp [])
        return True
clearPerceptsLocal :: IORef(TPerception) -> String -> IO Bool
clearPerceptsLocal p name = do
    (Perceptions gp lp ua) <- readIORef p
    if (null gp) then
        return True
     else do
        -- lp' eh o lp menos as tuplas que tem como primeiro elemento name
        let lp' = lp `minus` [(name, perception) | (n, perception) <- lp, n==name]
        modifyIORef p $ \(Perceptions _ lp _) -> (Perceptions [] lp [])
        return True
createAllPerceptions :: IORef(TPerception) -> [(String,Int)] -> Int -> IO ()
createAllPerceptions _ [] _ = return ()
createAllPerceptions p ((c@(n,a)):cs) step = do
    addPerceptLocal p n (Percept "alive_neighbors" [show a] [])
    addPerceptLocal p n (Percept "step" [show step] [])
    createAllPerceptions p cs step
turnPercepts :: IORef(TPerception) -> Model -> IO ()
turnPercepts p m = do
    modifyIORef p $ \(Perceptions _ _ _) -> (Perceptions [] [] [])
    step <- getStep m
    neighborsList <- getNeighbors m
    createAllPerceptions p neighborsList step
cmdRealize :: IORef(TPerception) -> SyncModel -> GLWidget -> String -> Maybe Bool
    -> IO Bool
cmdRealize p c v n f = do
```

```
102     nextStep <- realize c n f
        if nextStep then
104         do
                model <- lock c
106             cleanModel model
                turnPercepts p model
108             unlock c model
                updateDisplay v
110             return True
         else
112         return True
    performAction :: IORef(TPerception) -> SyncModel -> GLWidget -> String -> Action
        -> IO Bool
114 performAction p controller viewer name action =
        let
116         actionName = functor action
            actionPars = params action
118         actionBy = name
        in
120         case actionName of
                "skip"  -> cmdRealize p controller viewer actionBy Nothing
122             "live"  -> cmdRealize p controller viewer actionBy (Just True)
                "die"   -> cmdRealize p controller viewer actionBy (Just False)
124             _       -> return True -- ignore action
    createAllPerceptionsBlock :: IORef(TPerception) -> SyncModel -> IO ()
126 createAllPerceptionsBlock p c = do
        model <- lock c
128     step <- getStep model
        n <- getNeighbors model
130     createAllPerceptions p n step
        unlock c model
132 serving :: (Socket, Conf) -> SyncModel -> GLWidget -> IO ()
    serving d@(socket, conf) controller viewer = do
134     internalData <- newIORef(Perceptions [] [] [])
        createAllPerceptionsBlock internalData controller
136     let
            conf = Conf 8079 Nothing
138         ms = [
                    ("EM.handshake", fun          $ handshake),
140                 ("EM.addPercept", fun         $ addPercept internalData),
                    ("EM.addPerceptLocal", fun    $ addPerceptLocal internalData),
142                 ("EM.havePercepts", fun       $ havePercepts internalData),
                    ("EM.getPercepts", fun        $ getPercepts internalData),
144                 ("EM.removePercept", fun      $ removePercept internalData),
                    ("EM.removePerceptLocal", fun $ removePerceptLocal internalData),
146                 ("EM.clearPercepts", fun      $ clearPercepts internalData),
                    ("EM.clearPerceptsLocal", fun $ clearPerceptsLocal internalData),
148                 ("EM.performAction", fun      $ performAction internalData
                        controller viewer)
                ]
150         handler = do
                Body b <- fmap rqBody askRq
152             liftIO $ handleCall (methods ms) (unpack b)
        simpleHTTPWithSocket socket conf handler
154 setup :: Int -> IO (Socket, Conf)
    setup port = do
156     let c = Conf port Nothing
        s <- bindPort c
158     return (s, c)
```