2
Fundamentação Teórica

Neste capítulo apresentaremos os principais conceitos e tecnologias utilizadas no desenvolvimento deste trabalho. A Seção 2.1 apresenta uma visão geral de sistemas multiagentes. A arquitetura BDI é descrita na Seção 2.2. A linguagem *AgentSpeak* é descrita na Seção 2.3. Na Seção 2.4 apresentamos o interpretador Jason. Conceitos sobre normas e agentes normativos são apresentados na Seção 2.5. Finalmente, na Seção 2.6 é fornecida uma visão geral sobre a linguagem de especificação Z utilizada na formalização da arquitetura proposta neste trabalho.

2.1
Sistemas Multiagentes

Sistemas multiagentes são sistemas compostos por múltiplas entidades (conhecidas como *agentes*) que interagem a fim de atingir objetivos comuns ou diferentes (Woolridge 2011). De acordo com (Woolridge 2011), um agente é um sistema de computador que está situado em algum ambiente, e é capaz de realizar comportamentos autônomos neste ambiente a fim de atingir seus objetivos.

Tal definição aborda uma característica importante que é *autonomia*. Alguns modelos de agente consideram *autonomia* apenas no que diz respeito à liberdade de realizar comportamentos. Assim, um agente é livre para realizar comportamentos de acordo com as circunstâncias, o que permite agentes atuarem em ambientes dinâmicos e flexíveis (Pollack 1992). *Autonomia* também tem sido relacionada à capacidade de atingir objetivos sem intervenção humana (Castelfranchi 1995). Apesar de diferentes concepções, nesta tese adotaremos o conceito de autonomia definido em (Woolridge 2011), onde *autonomia* é considerada uma propriedade que possibilita agentes tomarem decisões e, para isto, eles consideram os seus interesses em realizar determinados comportamentos, ou seja, consideram a sua motivação para atingir determinados objetivos ou para executar determinadas ações (López 2003).

Um dos primeiros esforços para ressaltar a importância de motivação foi realizado por (Slowman 1987)(Slowan and Croucher 1981) que trabalham com

PUC-Rio - Certificação Digital Nº 1012680/CA

a idéia de motivações como mecanismo para decidir o que o agente deve fazer. Em (Slowman 1987)(Slowan and Croucher 1981) motivações representam desejos, gostos ou preferências que podem ser classificadas como segue: Motivações de primeira ordem são aquelas que especificam objetivos; e Motivações de segunda ordem que são as motivações para gerar novas motivações e para atribuir prioridades a motivações conflitantes. (Slowman 1987) afirma que os seguintes parâmetros devem ser levados em consideração para que uma motivação gere um objetivo: importância, intensidade e urgência. Assim, somente os objetivos com alta intensidade são considerados por um agente para serem atingidos, no entanto, apenas os objetivos mais importantes serão atingidos. A urgência de cada objetivo determina o quão rápido um objetivo deve ser satisfeito.

Já de acordo com (Luck and d'Inverno 1998a), motivações estão associadas a objetivos. Cada motivação tem uma força (ou valor) que varia de acordo com o estado do agente. Este valor é utilizado para determinar quais objetivos controlam o comportamento do agente num determinado instante. Quando este valor excede um limite, o agente é dito ser motivado para fazer alguma coisa e um conjunto de objetivos é gerado. Cada agente autônomo possui um conjunto de motivações cujo objetivos dependem do tipo de agente sendo representado. Desta forma, objetivos são criados e destruídos a fim de mitigar as motivações do agente. Em (Luck and d'Inverno 1998a), motivações também são utilizadas para solucionar problema de objetivos conflitantes. Isto é, agentes autônomos sempre selecionam um conjunto de objetivos com as maiores motivações. Nesta tese adotamos a idéia de motivação adotada por (Luck and d'Inverno 1998a).

A fim de tornar o conceito de agente mais útil no sentido de permitir construir sistemas complexos, é necessário a elaboração de arquiteturas bem definidas que permitam projetar e ter um entendimento comum do funcionamento de agentes. Para tanto, uma variedade de arquiteturas para projetar agentes têm sido propostas variando de arquiteturas reativas (Woolridge 2011), onde decisões são tomadas através de alguma forma de mapeamento direto entre a situação atual e a realização de um comportamento, à arquitetura BDI, onde decisões são tomadas como resultado da manipulação de estrutura de dados representando estados mentais em um agente, especialmente crenças, desejos e intenções. Considerando que a arquitetura proposta neste trabalho é uma extensão da arquitetura BDI, na Seção 2.2 descreveremos tal arquitetura.

2.2

PUC-Rio - Certificação Digital Nº 1012680/CA

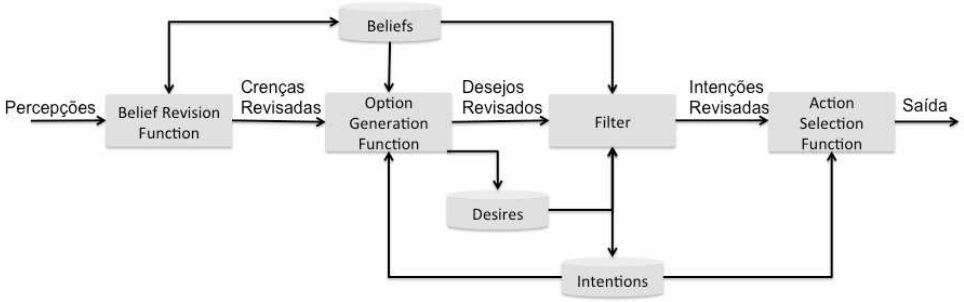

Figura 2.1: Arquitetura BDI (reproduzida de (Weiss 1999))

Arquitetura BDI

O modelo BDI (do inglês *Belief-Desire-Intention*) baseia-se nas noções de crenças, desejos e intenções como atitudes mentais que orientam a realização de comportamentos. O processo de raciocínio de um agente projetado a partir do modelo BDI é ilustrado na Figura 2.1. Como esta Figura 2.1 ilustra, o modelo BDI possui sete componentes:

(Beliefs) Representando as crenças do agente, tais crenças representam o conhecimento sobre o mundo;

(Belief Revision Function) Esta função revisa as crenças do agente tomando como base as percepções do ambiente e crenças já existentes;

(Option Generation Function) Esta função toma como base as crenças e intenções do agente para produzir um conjunto de opções (ou desejos);

(Desires) O conjunto de desejos do agente;

(Filter) Esta função determina um novo conjunto de intenções com base nas crenças, desejos e intenções já existentes do agente;

(Intentions) Um conjunto de intenções, representando o foco atual do agente, ou seja, os estados que o agente selecionou para serem alcançados;

(Action Selection Function) Esta função seleciona as ações a serem executadas com base nas intenções.

O modelo BDI é importante pois explicita como os componentes de um agente trabalho em conjunto. Uma das mais adotadas abordagens para implementar o modelo BDI é o sistema de raciocínio procedudal (PRS) (Georgeff and Lansky 1987). No PRS, como apresentado na Figura 2.2, um agente é equipado com uma biblioteca de planos (representada por *Plan Library*). Estes planos são manualmente construídos pelo programador do agente. Cada plano no PRS possui os seguintes componentes:

PUC-Rio - Certificação Digital Nº 1012680/CA

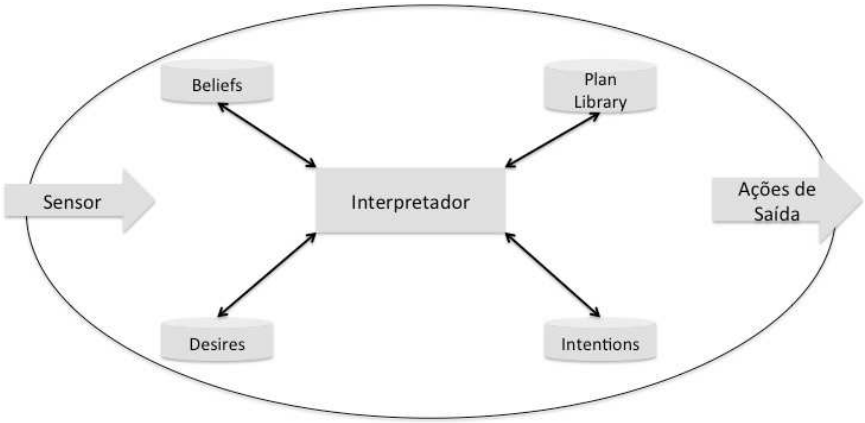

Figura 2.2: Sistema de Raciocínio Procedural (reproduzida de (Bordini et al. 2007))

- um objetivo representando a *condição de invocação* do plano;
- um *contexto* representando a condição para que um plano possa ser executado;
- um *corpo* representando a sequência de comportamentos que serão realizados pelo plano.

O interpretador PRS funciona como segue: Inicialmente um agente PRS possui uma coleção de planos e algumas crenças iniciais sobre o mundo. Quando o agente inicia a execução, o objetivo para ser atingido é colocado numa pilha. Esta pilha contém todos os objetivos que devem ser atingidos. Depois o agente busca na biblioteca de planos aqueles cuja *condição de invocação* é igual ao objetivo no topo da pilha, tais planos são conhecidos como *planos relevantes*, a partir dos *planos relevantes* é verificado quais deles o *contexto* é satisfeito a partir das crenças do agente, tais planos são conhecidos como *planos aplicáveis*. Um destes planos é escolhido para se tornar uma intenção. Uma intenção é selecionada para ser executada, que pode levar a geração de outros objetivos.

A linguagem *AgentSpeak*, introduzida por (Rao 1996), representa uma tentativa de refinar as principais características do PRS em uma linguagem de programação simples e unificada. Forneceremos mais detalhes da linguagem *AgentSpeak* na próxima Seção.

2.3 AgentSpeak(L)

O *AgentSpeak*(L) é uma linguagem de programação baseada em lógica de primeira ordem com as abstrações do modelo BDI onde um agente é definido em função de suas crenças iniciais e uma biblioteca de planos. Agentes

PUC-Rio - Certificação Digital Nº 1012680/CA

AgentSpeak(L) reagem a eventos, gerados a partir de mudanças no ambiente ou na sua estrutura interna, utilizando a biblioteca de planos.

$$
\begin{aligned}
&agent ::= beliefs\ plans\\
&beliefs ::= (literal\,'.')*\\
&literal ::= ['not']\ atomic\_formula\\
&atomic\_formula ::= < ATOM >'\ ('list\_of\_terms')'\\
&list\_of\_terms ::= term\ (',' term)*\\
&term ::= atomic\_formula \mid list \mid < VAR > \mid < STRING > \mid < NUMBER >\\
&list ::= '['[term\ (',' term) * ['\ |'\ (list \mid < VAR >)]]']'\\
&plans ::= (plan)*\\
&plan ::= trigger\_event\ ':'\ context\ '<-'\ body'.'\\
&trigger\_event ::= symbol\ trigger\\
&symbol ::= '-' \mid '+'\\
&trigger ::= belief \mid goal\\
&belief ::= literal\\
&goal ::= '!'\ literal\\
&\qquad \mid '?'\ literal
\end{aligned}
$$

$$
\begin{aligned}
&context ::= literal \mid <\text{VAR}> \mid\\
&\qquad 'not'\ context \mid\\
&\qquad context\ '\&'\ context \mid\\
&\qquad context\ '\mid'\ context \mid\\
&\qquad 'true'\\
&body ::= body\_formula('; 'body\_formula)*\\
&body\_formula ::= goal\\
&\qquad \mid action\\
&\qquad \mid symbol\ belief\\
&action ::= atomic\_formula
\end{aligned}
$$

Acima apresentamos a linguagem *AgentSpeak*[1]. No *AgentSpeak(L)* um agente é especificado por um conjunto de crenças, representando a base de crenças do agente, e um conjunto de planos, representando a biblioteca de planos do agente.

A base de crenças do agente é representada por *beliefs* que é um conjunto de literais (representado por *literal*), onde cada literal é uma fórmula atômica (representada por *atomic_formula*) ou sua negacão. Uma fórmula atômica é composta por um símbolo identificador iniciando com uma letra minúscula

[1]Esta é uma versão simplificada do AgentSpeak(L) entendido pelo interpretador *Jason*

PUC-Rio - Certificação Digital Nº 1012680/CA

(representado por $< ATOM >$) e uma lista de termos (representada por *list_of_terms*). Cada termo é uma fórmula atômica, uma lista (representada por *list*), uma variável (representada por $< VAR >$ que é um identificador iniciando com uma letra maiúscula), um número (representado por $< NUMBER >$ é um número inteiro ou ponto flutuante), ou uma *string* (representada por $< STRING >$).

A base de planos é representada por *plans*. Onde cada plano (representado por *plan*) é composto por uma condição de invocação (representada por *triggering_event*) indicando os eventos que o plano é capaz de lidar, um contexto (representado por *context*) indicando a circunstância para que um plano possa ser considerado aplicável e um corpo (representado por *body*), que é uma sequência de ações, objetivos ou atualizações das crenças.

Uma condição de invocação *triggering_event* é composta por um símbolo (representado por *symbol*) indicando o tipo de operação realizada ("+"indica que ocorreu uma adição e "−"indica que ocorreu uma remoção) e um *trigger* representando o elemento que sofreu a operação. Tal elemento pode ser uma crença (representada por *belief*) ou um objetivo (representado por *goal*), sendo que um objetivo pode ser do tipo *achievement* (representado por *! literal*) onde o agente deseja atingir um determinado estado do ambiente ou *test* (representado por *? literal*) onde o agente verifica se um determinado estado do ambiente é satisfeito.

Desta forma, é possível realizar as seguintes operações: (*"+"literal*) Uma crença foi adicionada a base de crenças do agente; ("−"literal) Uma crença foi removida da base de crenças do agente; ("+"!literal) Um novo objetivo do tipo *achievement* foi adotado pelo agente; ("−"!literal) Um objetivo do tipo *achievement* foi removido pelo agente; ("+"?literal) Um novo objetivo do tipo *test* foi adotado pelo agente; e, ("−"?literal) Um objetivo do tipo *test* foi removido pelo agente.

O contexto do plano (representado pelo tipo *context*) pode ser um literal, uma variável, a negação de um contexto, a conjunção ou disjunção de dois contextos, ou ainda sempre verdade.

O corpo do plano (representado pelo tipo *body*) é uma sequência de fórmulas do tipo *body_formula*, onde cada fórmula pode ser um objetivo do tipo *goal*, uma ação do tipo *action* ou a adição ou remoção de uma crença (representado por símbolo do tipo *symbol* e uma crença do tipo *belief*).

2.4

PUC-Rio - Certificação Digital Nº 1012680/CA

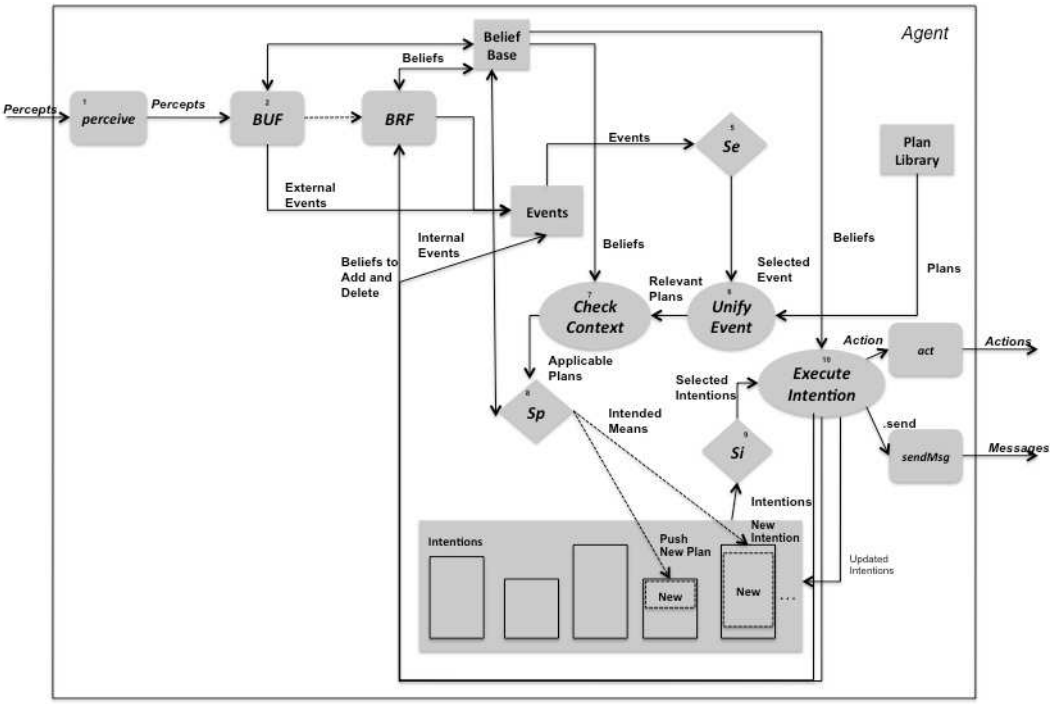

Figura 2.3: Interpretador Jason

Interpretador *Jason*

Jason é um interpretador para o *AgentSpeak*(L) que fornece suporte à criação de agentes BDI. Figura 2.3 (reproduzida a partir de (Bordini et al. 2007)) ilustra a arquitetura do *Jason*. Conjuntos (tais como, crenças-representadas por *Belief Base*-, eventos-representados por *Events*-, planos-representados por *Plan Library*- e intenções-representadas por *Intentions*-) são representados como retângulos, os losangos são usados para representar funções de seleção de um elemento a partir de um determinado conjunto e círculos para representar alguns dos processos envolvidos na interpretação de agentes *AgentSpeak*(L).

Cada ciclo de interpretação atualiza a lista de eventos de acordo com a percepção proveniente do ambiente e das informações advindas da própria execução do agente. Na função *Belief Update (BUF)* as percepções e as crenças do agente são utilizadas para atualizar a base de crenças do agente e o conjunto de eventos.

A função *Belief Review* (BRF) revisa a base de crenças com um literal para ser adicionado ou removido, e a intenção que requisitou a mudança da crença. Um único evento é selecionado pela função *Event Selection (Se)*, tal evento é unificado com as condições de invocação dos planos armazenados na *Plan Library* utilizando o processo *Unify Event* gerando um conjunto de *planos relevantes*. O contexto de tais planos são verificados de acordo com a base de crenças utilizando o processo *Check Context* gerando um conjunto de *planos*

PUC-Rio - Certificação Digital Nº 1012680/CA

aplicáveis.

A função *Plan Selection (Sp)* seleciona um dos *planos aplicáveis* para lidar com o evento selecionado. Tal plano é adicionado ao topo de uma intenção (se o evento foi gerado a partir de tal intenção), ou cria uma nova intenção no conjunto de intenções se o evento foi gerado a partir de percepções do ambiente. Neste ponto, vale ressaltar que uma intenção é uma sequência de planos. A função *Intention Selection (Si)* seleciona uma das intenções do agente para ser executada pelo processo *Execute Intention*. Quando todas as fórmulas no corpo de um plano são executadas, o plano é removido da lista de uma intenção, e se ainda existe planos na intenção o próximo plano é selecionado para ser executado, caso contrário a intenção é removida do conjunto de intenções. E o processo inicia novamente.

2.5 Normas e Agentes Normativos

Vimos na Seção 2.1 que agentes são livres para decidir qual comportamento realizar. Entretanto, em algumas situações tamanha liberdade possibilita que agentes realizem comportamentos não desejáveis e até impossibilita que os mesmos trabalhem em conjunto.

A fim de regular o comportamento de agentes e ainda manter a autonomia dos mesmos, o uso de normas tem ganhado espaço na comunidade como meio para definir proibições e responsabilidades num sistema multiagentes (da Silva 2008). A introdução de normas em sistemas multiagentes tem sido considerada como um fator importante para garantir a eficácia dos agentes (López 2003).

Normas especificam padrões de comportamento para um conjunto de agentes, papéis assumidos por eles ou grupos que os mesmos fazem parte (da Silva 2008) (López 2003). Tais padrões são às vezes representados como ações a serem executadas (da Silva 2008), ou restrições a serem impostas sobre as ações de um agente (da Silva 2008). Em outras ocasiões, padrões de comportamento são especificados através de objetivos que devem ser satisfeitos ou evitados por agentes (López 2003).

Em geral, normas não são aplicadas o tempo todo, mas apenas em circunstâncias especiais ou dentro de um contexto específico. Assim, as normas devem especificar as situações onde os responsáveis devem cumpri-las ou as situações onde os responsáveis podem desconsiderá-las (López 2003).

Dado que normas regulam o comportamento de agentes autônomos, portanto, livres para decidir por cumprir ou violar cada norma. A fim de influenciar o seu cumprimento, recompensas são utilizadas como forma de promover o

PUC-Rio - Certificação Digital Nº 1012680/CA

cumprimento das normas e punições são utilizadas como meio para inibir a violação. Tais recompensas e punições podem está associadas ao atingimento de objetivos (López 2003), a realização de ações ou ao estabelecimento de outras normas (da Silva 2008).

Também é necessário considerar que as normas de um sistema não são isoladas uma das outras. As vezes, a ativação, desativação, cumprimento ou violação de uma norma pode levar a ativação, desativação, cumprimento ou violação de outras normas (López 2003). Na literatura tais relacionamentos são conhecidos como *interlocking* entre normas. Neste contexto, vale ressaltar a possibilidade de existir conflitos entre normas, que acontece quando duas normas regulando o mesmo comportamento, estão ativadas mas, uma delas obriga a realização do comportamento enquanto a outra proíbe a realização do comportamento (Vasconcelos et al. 2008).

Embora uso de normas seja um mecanismo promissor para regular o comportamento de agentes, eles precisam ser capazes de lidar com tais normas de forma a manter sua autonomia. Em (Dignum et al. 2002) é discutido que para um agente capaz de lidar com normas (conhecido como *agentes normativos*) ser verdadeiramente autônomo, ele deve ser capaz de raciocinar sobre as normas que são de sua responsabilidade, e, ocasionalmente, violá-las se elas estão indo de encontro com os interesses individuais do agente.

Para tanto, em (Lopez and Marquez 2004) é definido que um agente normativo deve ser capaz de realizar dois importantes processos: *deliberação* e *cumprimento*. O primeiro diz respeito a decisão por selecionar uma norma para ser cumprida ou violada, e tal decisão pode ser realizada de diferentes maneiras, tais como: simplesmente decidir por cumprir todas as normas ou violar qualquer norma do sistema. O segundo diz respeito em como tal decisão por cumprir ou violar uma norma influenciará no raciocínio do agente, por exemplo, objetivos podem ser deixados de serem atingidos pela decisão por cumprir uma norma de proibição.

No Capítulo 3 apresentaremos algumas abordagens já existentes na literatura para o projeto de agentes normativos que adotam diferentes estratégias de *deliberação* e *cumprimento*.

2.6 Linguagem de Especificação Formal Z

Inúmeras técnicas e linguagens formais estão disponíveis para especificar propriedades de sistemas de software (D'inverno et al. 1997) . Ao escolher a linguagem Z adota-se uma linguagem que permite projetar formalmente sistemas a serem desenvolvidos. Z vem sendo cada vez mais uti-

PUC-Rio - Certificação Digital Nº 1012680/CA

lizada tanto na indústria quanto na academia, como um forte e elegante meio de especificação formal, e é apoiado por uma variedade de livros (por exemplo, (Bowen 1996)(Hayes and Flinn 1992)(Spivey 1988)), estudos de casos industriais (por exemplo, (Craigen et al. 1992)(Wezeman 1995)) e ferramentas para verificação de tipo, testes e realizações de animação (por exemplo, (Hewitt et al. 1997)(Saaltink 1997)). Desta forma, Z oferece os benefícios que precisamos, ajudando-nos a manter a crítica conexão entre modelos formais e sistemas implementados. Além disto, Z possui outros benefícios:

– É mais acessível do que muitos outros formalismos, uma vez que é baseada em componentes elementares já existentes, tais como a teoria dos conjuntos e lógica de primeira ordem;
– É uma linguagem extremamente expressiva, permitindo uma formalização consistente e estruturada de um sistema de computador e suas operações;

Agora descreveremos um pouco mais sobre a sintaxe de Z, cujo principal elemento sintático é o *esquema* que permite as especificações serem estruturadas em componentes modulares gerenciáveis. Esquemas Z consistem de duas partes: *parte superior*, onde reside a declaração de variáveis e seus tipos, e *parte inferior* que relaciona e condiciona tais variáveis.

Por exemplo, considere o esquema *Pair* apresentado abaixo, *first* e *second* são números naturais e a condição $first \geq second$ define que *first* é sempre maior que *second*.

Pair

$$first : \mathbb{N}$$
$$second : \mathbb{N}$$

$$first \geq second$$

Z permite que esquemas sejam incluídos dentro de outros esquemas. Por exemplo, se desejássemos incluir todas as informações do esquema,*Pair*, em outro esquema, T, juntamente com um outro conjunto de declarações, d e condições, p, poderíamos escrever o esquema como apresentado a seguir.

T

$$Pair$$
$$d$$

$$p$$

PUC-Rio - Certificação Digital Nº 1012680/CA

Para introduzir um tipo em Z, onde nenhuma informação sobre os elementos dentro deste tipo é conhecida, um conjunto específico é utilizado. Por exemplo, $[TREE]$ representa o tipo de todas as árvores sem dizer nada sobre a natureza dos elementos individuais dentro do tipo. Se desejarmos afirmar que uma variável assume um valor ou um conjunto de valores, podemos escrever $x : TREE$ e $x : \mathbb{P}\, TREE$, respectivamente.

Uma relação expressa algum relacionamento entre dois tipos existentes, conhecidos como *source* e *target*. Quando nenhum elemento do tipo *source* esta relacionado a dois ou mais elementos do tipo *target*, a relação é uma função. Se todo elemento no *source* está relacionado, então a função é *total*; se nem todos os elementos do *source* estão relacionados, então a função é *parcial*. Funções totais são representadas por $\rightarrow$ e funções parciais por $\nrightarrow$. Sequências são tipos especiais de funções onde o domínio consisti de números naturais contínuos.

Considere dois exemplos de relações, *REL1* e *REL2*. *REL1* define uma função entre as *trees*, enquanto REL2 define uma seqüência de *trees*. O tamanho do conjunto *source* determina se *REL1* é *parcial* ou *total*. Se os únicos elementos do *source* são *tree1*, *tree2* e *tree3*, então a função é *total*, se não é *parcial*.

$$Rel1 = \{(tree1, tree2), (tree2, tree3), (tree3, tree2)\}$$
$$Rel2 = \{(1, tree4), (2, tree2), (3, tree3)\}$$

Geralmente, a sequência *REL2* é escrita em Z como:

$$\langle tree4;\ tree2;\ tree3\rangle$$

Operações em sequências incluem obter o *head*, *tail* e *concatenação*.

$$head\langle tree4, tree2, tree3\rangle = tree4$$
$$tail\langle tree4, tree2, tree3\rangle = \langle tree2, tree3\rangle$$
$$\langle tree4, tree2\rangle \frown \langle tree3\rangle = \langle tree4, tree2, tree3\rangle$$

O domínio de uma relação ou função é o conjunto de elementos do *source* que estão relacionados.

$$dom\ REL1 = \{tree1, tree2, tree3\}$$
$$dom\ REL2 = \{1, 2, 3\}$$

Da mesma forma, a imagem é o conjunto de elementos *target* que estão relacionados.

PUC-Rio - Certificação Digital Nº 1012680/CA

$ran\ REL1 = \{tree2, tree3\}$
$ran\ REL2 = \{tree2, tree3, tree4\}$

A inversa de uma relação é obtida através da inversão de cada um dos pares ordenados de modo que o domínio torna-se a imagem, e a imagem torna-se o domínio.

$$REL1^{\sim} = \{(tree2, tree1), (tree3, tree2), (tree2, tree3)\}$$

Uma relação pode ser restringida a um subconjunto específico de seu domínio utilizando restrições de domínio.

$$\{tree2, tree3, tree4\} \lhd REL1 = \{(tree2, tree3), (tree3, tree2)\}$$

Do mesmo modo uma relação pode ser anti-restringida por um conjunto de tal modo que a relação resultante não contém quaisquer pares ordenados cujo primeiro elemento está no conjunto de restrição.

$$\{tree1, tree2\} \lhd\!\!\!\!- REL1 = \{(tree3, tree2)\}$$

Isto é conhecido como restrição anti-domínio. Por último, uma relação pode ser atualizada por uma outra relação através de uma substituição relacional.

$$REL1 \oplus \{(tree1, tree3), (tree2, tree2), (tree2, tree3)\} =$$
$$\{(tree1, tree3), (tree2, tree2), (tree2, tree3), (tree3, tree2)\}$$

Em Z podemos definir conjuntos de elementos. Por exemplo, a seguinte expressão descreve o conjunto de raízes quadradas de um número inteiro maior que 10.

$$\{x : \mathbb{Z} \mid x > 10 \bullet x * x\}$$

Funções, relações e variáveis podem ser definidas fora de um esquema por meio de definições axiomáticas. Definições axiomáticas, similar aos esquemas, contém declarações e condições sobre tais declarações. O exemplo seguinte define uma função que fornece o quadrado de um número inteiro.

$$square : \mathbb{Z} \rightarrow \mathbb{Z}$$
$$\forall n : \mathbb{Z} \bullet squaren = n * n$$

PUC-Rio - Certificação Digital Nº 1012680/CA

Mais detalhes sobre Z pode ser encontrado em (Spivey 1988).

2.7 Considerações Finais

Este Capítulo definiu na Seção 2.1 o conceito de agentes e autonomia adotado neste trabalho. Nas Seção 2.2 foi apresentada a arquitetura BDI, tal arquitetura serviu como base para elaboração da arquitetura proposta neste trabalho. Na Seção 2.3 apresentamos a linguagem *AgentSpeak* e na Seção 2.4 apresentamos o interpretador *Jason* ambos serviram como base para o fornecimento de mecanismos que possibilitassem a implementação de agentes projetados a partir da arquitetura proposta neste trabalho.

Embora exista outras arquiteturas para projetar agentes (Woolridge 2011) e tecnologias para implementação dos mesmos (Bordini et al. 2010). Focamos nos trabalhos que foram utilizados nesta tese.

Finalmente, apresentamos uma visão geral sobre normas e agentes normativos na Seção 2.5 e uma visão geral da linguagem de especificação Z, utilizada na formalização da arquitetura proposta neste trabalho, foi apresentada na Seção 2.6.

PUC-Rio - Certificação Digital Nº 1012680/CA