Linguagens de Programação Orientadas a Agentes: uma introdução baseada em AgentSpeak(L)

Rafael H. Bordini [1 2]
Renata Vieira [3]

Resumo: Este artigo apresenta uma introdução ao paradigma de programação orientada a agentes com base em uma linguagem particular chamada AgentSpeak(L). Exemplos de agentes programados nessa linguagem são fornecidos para facilitar a compreensão da linguagem. Conceitos básicos de sistemas multiagente e agentes inteligentes são dados de forma a fornecer ao leitor um material auto-contido. O artigo apresenta ainda um apanhado geral dos diversos trabalhos de pesquisa sobre AgentSpeak(L) que estão em desenvolvimento, bem como referências a diversas outras linguagens de programação orientadas a agentes.

Palavras-chave: Programação Orientada a Agents, Agentes Cognitivos, Arquitetura BDI, AgentSpeak(L).

Abstract: This paper presents an introduction to the paradigm of agent-oriented programming based on a particular language called AgentSpeak(L). Examples of agents programmed in that language are given in order to facilitate the understanding of the language. Basic concepts on multi-agent systems and intelligent agents are given so as to provide the reader with a self-contained material. The paper also presents an overview of the various research projects on AgentSpeak(L) currently under development, and it points to several other agent-oriented programming languages.

Keywords: Agent-Oriented Programming, Cognitive Agents, BDI Architecture, AgentSpeak(L).

1 Introdução

A idéia de Linguagens de Programação Orientadas a Agentes teve seu início marcado pelo importante artigo de Yoav Shoham, intitulado "*Agent-Oriented Programming*", que foi publicado em 1993 [55]. A proposta deste novo paradigma de programação foi inspirada na visão social de computação, discutida em profundidade pela comunidade de Inteligência Artificial Distribuída, e Sistemas Multiagente (SMA) em particular [65, 62]. O trabalho

[1]Department of Computer Science, University of Liverpool, United Kingdom
{R.Bordini@csc.liv.ac.uk}
[2]Programa de Pós-Graduação em Ciência da Computação, UFRGS, Brasil
{bordini@inf.ufrgs.br}
[3]Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, UNISINOS, Brasil
{renata@exatas.unisinos.br}

na área de SMA, em particular a arquitetura BDI (*Beliefs-Desires-Intentions*, ou crenças-desejos-intenções), introduzida na seção 2, é a base da proposta de Shoham. Sua proposta defende que da mesma forma que noções mentalísticas (tais como crenças, desejos e intenções) auxiliam o nosso raciocínio quotidiano sobre as atividades dos seres humanos, uma certa antropomorfização da maneira como os programadores descrevem componentes (chamados "agentes") de um sistema computacional deve, em princípio, facilitar a concepção de sistemas complexos. De forma geral, os SMA têm como um de seus objetivos a concepção de sistemas complexos, naturalmente distribuídos, situados em ambientes dinâmicos, imprevisíveis e parcialmente observáveis. As vantagens desta forma de se conceber sistemas computacionais é discutida, por exemplo, em [68]; uma discussão mais específica sobre linguagens de programação orientadas a agentes pode ser encontrada em [43].

A antropomorfização da concepção de sistemas computacionais proposta pela área de SMA não para no nível de um agente individual dentro do sistema, atribuindo-se noções mentalísticas a estes. No nível social (i.e, do conjunto de agentes), as analogias continuam, de forma que noções de sociedades humanas como papéis, grupos, organizações, e assim por diante [34] também devem ser utilizadas no projeto de sistemas computacionais. Isto sugere a necessidade de uma completa reformulação do processo de engenharia de software, e não somente das linguagens de programação a serem utilizadas. Com isto, muitos trabalhos acadêmicos tem recentemente se concentrado no que se chama "engenharia de software orientada a agentes"[4]. Este artigo não abrange esse contexto, concentrando-se especificamente em linguagens de programação orientadas a agentes. No entanto, vale mencionar que a proposta mais conhecida para engenharia de software orientada a agentes é a apresentada em [67].

Este artigo irá apresentar, em particular, a linguagem AgentSpeak(L), proposta inicialmente por [48] e posteriormente desenvolvida numa série de trabalhos (mencionados na Seção 5.1). Essa linguagem é inspirada na arquitetura BDI, e os conceitos utilizados na linguagem são bastante fiéis aos elementos dessa arquitetura, bem como às idéias da teoria filosófica que a embasam. A sintaxe da linguagem é bastante elegante, consistindo de uma extensão do paradigma de programação em lógica[5] através uma notação muito clara. A linguagem AgentSpeak(L) é usada aqui para introduzir conceitos que são utilizados, com algumas variações, na maioria das linguagens de programação orientadas a agentes. Ao apresentar uma linguagem particular em detalhes, é possível mostrar exemplos de programas utilizando essa linguagem. Este artigo apresenta dois exemplos de programas AgentSpeak(L); os exemplos são simples, servindo apenas para facilitar a compreensão das construções básicas da linguagem.

[4]Existe uma série de *workshops* com esse tema, cuja página *web* é <`http://www.csc.liv.ac.uk/~mjw/aose/`>.

[5]Este texto assume uma certa familiaridade do leitor com linguagens de programação em lógica, como Prolog por exemplo.

Este texto está organizado da seguinte maneira. A próxima seção introduz alguns conceitos básicos sobre agentes BDI, o tipo de agente cognitivo para o qual é voltada a linguagem AgentSpeak(L), apresentada na Seção 3. A Seção 4 mostra dois exemplos de programas em AgentSpeak(L), com o objetivo de facilitar a compreensão dos componentes básicos da linguagem. Antes de uma breve conclusão, a Seção 5 sumariza vários trabalhos recentes sobre a linguagem AgentSpeak(L), bem como fornece referências para diversas outras linguagens de programação orientadas a agentes.

2 Agentes BDI

A linguagem AgentSpeak(L) foi projetada para a programação de agentes BDI. Esta seção introduz os conceitos básicos da área de agentes e sistemas multiagente e depois concentra-se em agentes BDI, provendo o conhecimento básico da área para a compreensão de algumas noções da linguagem AgentSpeak(L). Uma versão bem mais detalhada desta introdução pode ser encontrada em [11].

2.1 Conceitos Básicos

A área de sistemas multiagente ocupa-se da construção de sistemas computacionais a partir da criação de entidades de software autônomas, denominadas *agentes*. Os agentes interagem através de um *ambiente* compartilhado por outros agentes de uma *sociedade*, e atuam sobre esse ambiente, alterando seu estado.

Como cada agente possui um conjunto limitado de capacidades específicas e objetivos próprios em relação aos estados do ambiente que quer atingir, mecanismos para a interação e coordenação dessas entidades são necessários, pois, em geral, os agentes não conseguem atingir todos seus objetivos isoladamente. Por exemplo, atingir um acordo, para que dois ou mais agentes cooperem na resolução de um problema comum entre eles, é uma forma de coordenar as atividades para atingir o objetivo comum da forma mais eficaz possível. Dessa forma, é possível para os projetistas de sistemas computacionais a criação de sistemas complexos de forma naturalmente distribuída e *bottom-up*. Contudo, criar mecanismos genéricos para a coordenação de tais agentes para que o sistema como um todo (em geral chamado de uma sociedade de agentes) funcione de forma adequada e eficiente é um dos grandes desafios da área. Outro grande desafio é a especificação interna de um agente, em que tipicamente se deseja uma representação simbólica daquilo que o agente sabe sobre o ambiente (e sobre os outros agentes naquele ambiente), bem como daquilo que o agente pretende atingir (sua motivação).

Existem dois grandes tipos de SMA: os reativos [23] e os cognitivos (como os que serão abordados ao longo deste texto). SMA reativos seguem a idéia de que um comporta-

mento inteligente em um sistema emerge da interação entre um grande número de agentes muito simples; a principal influência nesse tipo de trabalho vem da entomologia (estudo dos insetos). Nesses sistemas, os agentes não possuem uma representação explícita do estado do ambiente e dos outros agentes, nem da história de suas ações anteriores; eles têm comportamentos que podem ser descritos como autômatos finitos simples, e possuem um conjunto de regras que mapeiam percepções do ambiente diretamente em ações sobre este (os agentes agem sob um esquema estímulo-resposta). Já os SMA cognitivos em geral possuem tipicamente poucos agentes, dado que cada agente é um sistema sofisticado e computacionalmente complexo.

Outra divisão bem clara na comunidade de SMA está nos formalismos empregados pelos pesquisadores. De um lado, estão os pesquisadores que utilizam Teoria do Jogos para o projeto e coordenação de agentes. De outro lado, existem os pesquisadores que usam lógica matemática (ou representações baseadas em lógica) como a base de seus trabalhos. Teoria dos Jogos é uma área da matemática que possui ferramentas para a decisão de qual estratégia em um jogo abstrato de dois ou mais jogadores é a melhor para os jogadores. Essas ferramentas são muito usadas em Teoria da Decisão, na área de Economia, e de lá foram trazidas para o contexto de coordenação de agentes em SMA [54]. Por outro lado, na abordagem de SMA baseada em lógica, utiliza-se, em geral, lógicas modais bastante complexas (veja, por exemplo, [64]) para o projeto e desenvolvimento de agentes. Nessa abordagem a coordenação pode ser obtida através de um mecanismo de raciocínio social. Raciocínio social é qualquer raciocínio realizado por um agente a cerca dos outros agentes na sociedade (um trabalho recente na área é [53]). Os trabalhos sobre raciocínio social normalmente tem como base o trabalho sobre dependência social de Castelfranchi [15, 16].

A pesquisa em SMA é essencialmente de natureza multidisciplinar. Cada um destes campos de trabalho em SMA obteve inspiração em uma ou mais disciplinas além de Ciência da Computação, tais como: psicologia, ciência cognitiva, sociologia, entomologia, economia, teoria das organizações, teoria dramática, antropologia (veja [10]), entre outras. Em particular, a inspiração em sociologia marca a transição de sistemas de inteligência artificial tradicionais (que são monolíticos) para sistemas multiagente. A inspiração deve-se à corrente filosófica denominada pragmatismo filosófico que permeia todo o trabalho contemporâneo das ciências sociais, e que concebe toda atividade humana, inclusive a cognição, como uma prática social, ou seja, algo que somente se dá em um processo de interação entre indivíduos [39].

Se do ponto de vista filosófico a motivação para os SMA é a linha do pragmatismo filosófico, de um ponto de vista de engenharia de sistemas, a motivação para os SMA é a forma natural para o projeto de grandes sistemas computacionais que resulta das metáforas já introduzidas e que permitem analogias diretas com o mundo real e a forma como raciocinamos sobre ele. Assim, sistemas computacionais são vistos como formados de ambientes

(em analogia a ambientes físicos) onde os agentes atuam, havendo possivelmente a presença de objetos[6]. Nesses ambientes os agentes podem formar grupos, em um nível mais alto de abstração formando comunidades ou sociedades de agentes, e assim por diante.

Do ponto de vista interno do agente também se buscam diversas metáforas com *estados mentais*, tais como crenças, objetivos, intenções e compromissos. Agentes deliberam sobre que objetivos vão atingir, baseados em suas próprias motivações, e com base nesses objetivos e observações do ambiente decidem que ações tomar, e que requisições feitas por outros agentes aceitar (essas são algumas das características que diferenciam agentes de objetos). Um agente de um sistema pode ser programado através de múltiplos sub-agentes, assim como agentes formam grupos, grupos formam sociedades e assim por diante. A essa propriedade, Demazeau [19] se refere como o princípio da recursividade em SMA.

Note que o projeto de um sistema computacional complexo utilizando essas metáforas, faz uso de técnicas automáticas de decomposição de tarefas e computação distribuída, permitindo a implementação de forma mais direta e eficiente para aplicações naturalmente distribuídas (em geral não existe controle centralizado em sistemas multiagente). Outra característica de aplicações para as quais os SMA representam uma alternativa interessante de implementação são sistemas dinâmicos, em que é impossível prever todos os casos que devem ser tratados. Nesse aspecto, a característica de autonomia dos agentes é fundamental: as pesquisas em SMA procuram mecanismos para que os agentes gerem e persigam seus próprios objetivos, permitindo assim a adaptação a situações imprevistas.

Um agente é dito *autônomo* se ele somente realiza tarefas por decisão própria, com base em uma representação de seus interesses individuais[7]. Uma classe de sistemas de inteligência artificial distribuída muito difundida na década de 80 chamada de resolução de problemas distribuída cooperativa (*cooperative distributed problem solving*) partia do princípio da benevolência de agentes: como todos agentes tem um objetivo comum nesses sistemas, sempre que um agente solicita uma tarefa a outro, esse aceita e procura realizar a tarefa da melhor maneira possível dentro de suas capacidades. Isto não é necessariamente verdade em SMA, onde agentes são considerados autônomos e têm interesses próprios.

Outro aspecto importante de agentes em SMA é que os agentes possuem conhecimento incompleto do seu ambiente e seguem o princípio da racionalidade limitada (*bounded rationality*); ou seja, os agentes não são necessariamente capazes de perceber tudo o que é verdade no estado atual do ambiente (o que para algumas aplicações é essencial), e possuem uma capacidade e tempo de processamento limitado (i.e., não podem ficar raciocinando indefinidamente sobre o problema até achar a melhor solução). Esses são aspectos fundamentais que permitem a utilização de SMA para implementação de soluções para problemas comple-

[6] Ao contrário dos agentes, objetos não são entidades autônomas, nem auto-motivadas.
[7] Essa é a noção mais simples de autonomia. O conceito de autonomia possui diversas conotações diferentes, e é muito debatido na literatura da área.

xos e dinâmicos.

Dada essa breve introdução à área de SMA, apresentamos agora uma definição de "agente" no contexto deste artigo. A Figura 1 permite a visualização de aspectos importantes de um agente, apresentando o modelo geral de agente proposto por Wooldridge [63].

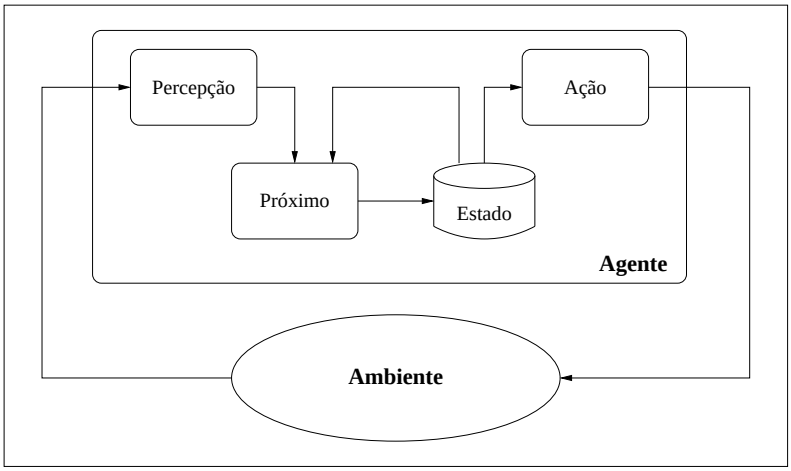

Figura 1. Modelo Geral de Agente (traduzido de [63])

É preciso explicar logo de início que os pesquisadores da área não possuem uma definição de consenso para o que é um agente. Algumas definições ficaram bem conhecidas, como a de Wooldridge e Jennings [66] e a de Franklin [27]. Ao invés de citar essas definições que são as mais aceitas, preferiu-se aqui mencionar algumas características que são importantes (do ponto de vista dos autores) para o conceito de agente na concepção de SMA e que estão relacionadas com a linguagem AgentSpeak(L) apresentada a seguir. Note que, como uma área da inteligência artificial distribuída, trata-se aqui de sistemas que usam técnicas de inteligência artificial. Devido ao sucesso e grande atenção dada a essa área na segunda metade da década de 90, o termo *agente* se difundiu amplamente em diversas áreas da Ciência da Computação. Nessa perspectiva, criou-se o termo *agentes de software* [28], em que praticamente qualquer processo comunicante passa a ser denominado agente.

São mencionados a seguir alguns aspectos importantes para a compreensão do que é um agente no contexto de SMA cognitivos. Um agente, nesta concepção, é um sistema computacional que é capaz de:

percepção: o agente é capaz de perceber alterações no ambiente (conforme ilustrado na Figura 1);

ação: as alterações no ambiente são provenientes das ações que os agentes realizam constantemente no ambiente; um agente age sempre com o intuito de atingir seus objetivos (motivação), ou seja, com o intuito de transformar o ambiente de seu estado atual em um outro estado desejado pelo agente (veja *motivação* abaixo);

comunicação: umas das ações possíveis de um agente é comunicar-se com outros agentes da sociedade (i.e., que compartilham o mesmo ambiente); como os agentes precisam coordenar suas ações, a comunicação entre eles é essencial[8];

representação: o agente possui uma representação simbólica explícita daquilo que acredita ser verdade em relação ao ambiente e aos outros agentes que compartilham aquele ambiente;

motivação: como em SMA os agentes são (ou podem ser) autônomos, é essencial que exista não só uma representação do conhecimento do agente, mas também uma representação dos desejos ou objetivos (i.e., aspectos motivacionais) daquele agente; em termos práticos, isto significa ter uma representação de estados do ambiente que o agente almeja alcançar; como conseqüência, o agente age sobre o ambiente por iniciativa própria para satisfazer esses objetivos;

deliberação: dada uma motivação e uma representação do estado atual do ambiente em que se encontra o agente, esse tem que ser capaz de decidir, dentre os estados de ambiente possíveis de ocorrerem no futuro, quais de fato serão os objetivos a serem seguidos por ele;

raciocínio e aprendizagem: técnicas de inteligência artificial clássica para, por exemplo, raciocínio e aprendizagem podem ser estendidas para múltiplos agentes, aumentando significativamente o desempenho desses, por exemplo no aspecto de deliberação; note que nem sempre se esperam essas características de raciocínio e aprendizagem de quaisquer agentes; a criação de mecanismos de aprendizagem específicos para ambientes multiagente é uma área de pesquisa que ainda requer bastante investigação (existem, contudo, vários trabalhos que aplicam aprendizagem por reforço em SMA [61]).

A seguir, apresenta-se a arquitetura BDI, um modelo específico para o desenvolvimento de agentes inteligentes que satisfazem os critérios usados na definição acima.

[8] A comunicação é uma característica essencial em SMA cognitivos baseados em lógica. No enfoque de SMA baseado em Teoria dos Jogos (mencionado anteriormente), é possível que os agentes prescindam de comunicação para obter coordenação.

2.2 A Arquitetura BDI

As mais importantes arquiteturas de agentes deliberativos são baseadas em um modelo de cognição fundamentado em três principais atitudes mentais que são as crenças, os desejos, e as intenções (abreviadas por BDI, *beliefs*, *desires* e *intentions*). A fundamentação filosófica para esta concepção de agentes vem do trabalho de Dennett [20] sobre sistemas intencionais e de Bratman [12] sobre raciocínio prático. A primeira implementação de um sistema baseado nessas idéias foi o IRMA (*Intelligent Resource-bounded Machine Architecture*) [13]. Com base nessa experiência, foi criado outro sistema utilizando a arquitetura BDI, chamado PRS (*Procedural Reasoning System*) [29]; o PRS, por sua vez, tem como sucessor um sistema chamado dMARS [35, 21]. Baseando-se nesses trabalhos, Rao criou a linguagem de programação AgentSpeak(L). Georgeff e Rao foram os principais autores na elaboração de arquiteturas abstratas de agente baseadas no modelo BDI [51, 50], bem como na concepção de lógicas BDI [52, 49]. Lógicas BDI são lógicas multimodais com um operador modal para cada uma das atitudes mentais do modelo BDI, além dos operadores usuais da lógica temporal de tempo ramificado e da lógica de ação. Para uma apresentação sucinta de uma lógica BDI com referências para as lógicas modais nas quais ela se baseia, bem como para a semântica de mundos possíveis normalmente utilizada para elas, veja [56].

De forma esquemática, a arquitetura BDI genérica está apresentada na Figura 2, conforme proposto em [63].

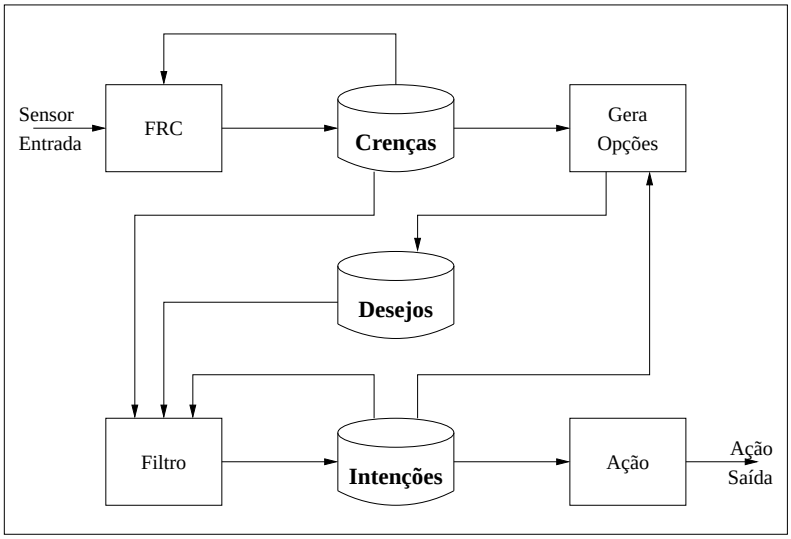

Figura 2. Arquitetura BDI Genérica (adaptado de [63])

Resumidamente, esta arquitetura de agente está estruturada da seguinte forma. As *crenças* representam aquilo que o agente sabe sobre o estado do ambiente e dos agentes naquele ambiente (inclusive sobre si mesmo). Os *desejos* representam estados do mundo que o agente quer atingir (dito de outra forma, são representações daquilo que ele quer que passe a ser verdadeiro no ambiente). Em tese, desejos podem ser contraditórios, ou seja, pode-se desejar coisas que são mutuamente exclusivas do ponto de vista de ação prática. Normalmente se refere a *objetivos* como um subconjunto dos desejos que são todos compatíveis entre si. As *intenções* representam seqüências de ações específicas que um agente se compromete a executar para atingir determinados objetivos.

A *função de revisão de crenças* (referenciada por *FRC* na figura) recebe a informação sensória (i.e., percebe propriedades do ambiente) e, consultando as crenças anteriores do agente, atualiza essas crenças para que elas reflitam o novo estado do ambiente[9]. Com essa nova representação do estado do ambiente, é possível que novas opções fiquem disponíveis (opções de estados a serem atingidos). A função denominada na figura como *Gera Opções* verifica quais as novas alternativas de estados a serem atingidos, que são relevantes para os interesses particulares daquele agente. Isto deve ser feito com base no estado atual do mundo (conforme as crenças do agente) e nas intenções com que o agente já está comprometido. A atualização dos objetivos se dá, então, de duas formas: as observações do ambiente possivelmente determinam novos objetivos do agente, e a execução de intenções de mais alto nível pode gerar a necessidade de que objetivos mais específicos sejam atingidos.

Uma vez atualizado o conhecimento e a motivação do agente, é preciso, em seguida, decidir que curso de ações específico será usado para alcançar os objetivos atuais do agente (para isso é preciso levar em conta os outros cursos de ações com os quais o agente já se comprometeu, para evitar ações incoerentes, bem como eliminar intenções que já foram atingidas ou que se tornaram impossíveis de ser atingidas). Esse é o papel da função *Filtro*, que atualiza o conjunto de intenções do agente, com base nas crenças e desejos atualizados e nas intenções já existentes. Esse processo realizado pela função Filtro para determinar como atualizar o conjunto de intenções do agente é normalmente chamado de *deliberação*[10]. Com o conjunto de intenções já atualizado, a escolha de qual ação específica, entre aquelas pretendidas, será a próxima a ser realizada pelo agente no ambiente é feita pela função *Ação*. Em certos casos, em que não é necessário priorização entre múltiplas intenções, a escolha pode ser simples; ou seja, basta escolher qualquer uma entre as intenções ativas, desde que se garanta que todas as intenções terão, em algum momento, a oportunidade de serem escolhidas para execução. Porém, alguns agentes podem precisar usar escolha de intenções baseadas em

[9]Note que os mecanismos reais de percepção normalmente são imprecisos e/ou incompletos; ou seja, a percepção pode ser falha, no sentido de não refletir a realidade do ambiente, ou pode não ser total, perdendo alguma informação sobre o ambiente.

[10]Essa é a origem do termo *deliberativo* muitas vezes utilizado para se referir a agentes cognitivos, já que a habilidade de realizar essa deliberação é uma característica muito importante para esse tipo de agente.

critérios mais sofisticados para garantir que certas intenções sejam priorizadas em relação a outras em certas circunstâncias.

3 A Linguagem AgentSpeak(L)

3.1 Noções Gerais

A linguagem AgentSpeak(L) foi projetada para a programação de agentes BDI na forma de sistemas de planejamento reativos (*reactive planning systems*). Sistemas de planejamento reativos são sistemas que estão permanentemente em execução, reagindo a eventos que acontecem no ambiente em que estão situados através da execução de planos que se encontram em uma biblioteca de planos parcialmente instanciados.

A linguagem de programação AgentSpeak(L) foi primeiramente apresentada em [48]. A linguagem é uma extensão natural e elegante de programação em lógica para a arquitetura de agentes BDI, que representa um modelo abstrato para a programação de agentes e tem sido a abordagem predominante na implementação de agentes inteligentes ou "racionais" [64]. Um agente AgentSpeak(L) corresponde à especificação de um conjunto de crenças que formarão a base de crenças inicial e um conjunto de planos. Um *átomo de crença* é um predicado de primeira ordem na notação lógica usual, e *literais de crença* são átomos de crenças ou suas negações. A *base de crenças* de um agente é uma coleção de átomos de crença.

AgentSpeak(L) distingue dois tipos de objetivos: *objetivos de realização* (*achievement goals*) e *objetivos de teste* (*test goals*). Objetivos de realização e teste são predicados, tais como crenças, porém com operadores prefixados '!' e '?', respectivamente. Objetivos de realização expressam que o agente quer alcançar um estado no ambiente onde o predicado associado ao objetivo é verdadeiro. Na prática, esses objetivos iniciam a execução de *subplanos*. Um objetivo de teste retorna a unificação do predicado de teste com uma crença do agente, ou falha caso não seja possível a unificação com nenhuma crença do agente. Um *evento ativador* (*triggering event*) define quais eventos podem iniciar a execução de um plano. Um *evento* pode ser interno, quando gerado pela execução de um plano em que um subobjetivo precisa ser alcançado, ou externo, quando gerado pelas atualizações de crenças que resultam da percepção do ambiente. Eventos ativadores são relacionados com a *adição* e *remoção* de atitudes mentais (crenças ou objetivos). Adição e remoção de atitudes mentais são representadas pelos operadores prefixados ('+') e ('-').

Planos fazem referência a *ações básicas* que um agente é capaz de executar em seu ambiente. Essas ações são definidas por predicados com símbolos predicativos especiais (chamados símbolos de ação) usados para distinguir ações de outros predicados. Um plano é formado por um evento ativador (denotando o propósito do plano), seguido de uma conjunção

de literais de crença representando um *contexto*. O contexto deve ser conseqüência lógica do conjunto de crenças do agente no momento em que o evento é selecionado pelo agente para o plano ser considerado *aplicável*. O resto do plano é uma seqüência de ações básicas ou subobjetivos que o agente deve atingir ou testar quando uma instância do plano é selecionada para execução.

```
+concert(A,V) : likes(A)
   ← !book_tickets(A,V).

+!book_tickets(A,V) : ¬busy(phone)
   ← call(V);
      ...;
      !choose_seats(A,V).
```

Figura 3. Exemplos de planos AgentSpeak(L)

A Figura 3 apresenta exemplos de planos AgentSpeak(L). O primeiro plano especifica que ao anúncio de um concerto a ser realizado pelo artista `A` no local `V` (do Inglês *venue*), correspondendo a adição de uma crença `concert(A,V)` como conseqüência da percepção do ambiente. Se for o caso de o agente gostar do artista `A`, então o agente terá como objetivo a reserva dos ingressos para esse concerto. O segundo plano especifica que ao adotar o objetivo de reservar ingressos, se for o caso de a linha telefônica não estar ocupada, então o agente pode executar o plano que consiste de: executar a ação básica de fazer contato telefônico com o local do concerto `V`, seguido de um determinado protocolo de reserva de ingressos (indicado por '...'), e que termina com a execução de um subplano para a escolha de acentos em eventos desse tipo naquele local. (Assume-se que fazer contato telefônico é uma ação básica que o agente é capaz de executar no ambiente em questão).

3.2 Sintaxe Abstrata

A especificação de um agente ag em AgentSpeak(L) deve ser feita de acordo com a gramática apresentada a seguir (adaptada de [45]). Em AgentSpeak(L), um agente é especificado por um conjunto de crenças *bs* (*beliefs*) correspondendo à base de crenças inicial do agente, e um conjunto de planos *ps* que forma a biblioteca de planos do agente.

$$
\begin{array}{lcll}
\mathit{ag} & ::= & \mathit{bs} \quad \mathit{ps} & \\
\mathit{bs} & ::= & b_1 \ldots b_n & (n \geq 0) \\
\mathit{at} & ::= & \mathsf{P}(t_1, \ldots, t_n) & (n \geq 0) \\
\mathit{ps} & ::= & p_1 \ldots p_n & (n \geq 1) \\
p & ::= & \mathit{te} : \mathit{ct} \leftarrow h & \\
\mathit{te} & ::= & +\mathit{at} \mid -\mathit{at} \mid +g \mid -g & \\
\mathit{ct} & ::= & \mathit{at} \mid \neg \mathit{at} \mid \mathit{ct} \wedge \mathit{ct} \mid \mathsf{T} & \\
h & ::= & a \mid g \mid u \mid h; h & \\
a & ::= & \mathsf{A}(t_1, \ldots, t_n) & (n \geq 0) \\
g & ::= & !\mathit{at} \mid ?\mathit{at} & \\
u & ::= & +\mathit{at} \mid -\mathit{at} &
\end{array}
$$

As fórmulas atômicas *at* da linguagem são predicados onde P é um símbolo predicativo e $t_1, \ldots, t_n$ são termos padrão da lógica de primeira ordem. Chamamos de *crença* uma fórmula atômica *at* sem variáveis e b é meta-variável para crenças. O conjunto inicial de crenças de um programa AgentSpeak(L) é uma seqüência de crenças *bs*.

Um plano em AgentSpeak(L) é dado por p acima, onde *te* (*triggering event*) é o evento ativador, *ct* é o contexto do plano (uma conjunção de literais de crença) e h é uma seqüência de ações, objetivos ou atualizações de crenças. A construção *te* : *ct* é dita a *cabeça* do plano, e h o *corpo* do plano. O conjunto de planos de um agente é dado por *ps* como uma lista de planos.

Um evento ativador *te* corresponde à adição/remoção de crenças da base de crenças do agente ($+$*at* e $-$*at*, respectivamente), ou à adição/remoção de objetivos ($+g$ e $-g$, respectivamente). O agente possui um conjunto de *ações básicas* que utiliza para atuar sobre o ambiente. Ações são referidas por predicados usuais com a exceção de que um símbolo de ação A é usado no lugar do símbolo predicativo. Objetivos g podem ser objetivos de realização ($!$*at*) ou de teste ($?$*at*). Finalmente, $+$*at* e $-$*at* (no corpo de um plano) representam operações de atualização (*update*) da base de crença u, através da adição ou remoção de crenças respectivamente.

Note que uma fórmula $!g$ no corpo de um plano gera um evento cujo evento ativador é $+!g$ (este assunto será tratado em maiores detalhes na próxima seção). Portanto, planos escritos pelo programador que tenha um evento ativador que possa ser unificado com $+!g$ representam alternativas de planos que devem ser considerados no tratamento de tal evento. Planos com evento ativador do tipo $+$*at* e $-$*at* são utilizados no tratamento de eventos que são gerados quando crenças são adicionadas ou removidas (tanto como conseqüência da percepção do ambiente, como devido a alterações de crenças explicitamente requisitadas no corpo de um plano). Eventos ativadores do tipo $-!g$ são usados para o tratamento de falhas de pla-

nos (mas não serão abordados neste texto), e eventos ativadores do tipo $+?g$ e $-?g$ não são utilizados na implementação atual da linguagem (maiores detalhes na Seção 5.1). A seção a seguir fornece mais detalhes sobre a geração de eventos, e outros aspectos importantes da interpretação de programas AgentSpeak(L).

3.3 Semântica Informal

Um interpretador abstrato para a linguagem AgentSpeak(L) precisa ter acesso à base de crenças e à biblioteca de planos, e gerenciar um conjunto de *eventos* e um conjunto de *intenções*. Seu funcionamento requer três *funções de seleção*: a função de seleção de eventos (S_E) seleciona um único evento do conjunto de eventos; uma outra função (S_{Ap}) seleciona uma "opção" (um plano aplicável) entre o conjunto de planos aplicáveis para um evento selecionado; e a terceira função (S_I) seleciona uma intenção do conjunto de intenções. As funções de seleção são específicas para cada agente, sendo responsáveis por parte significativa do comportamento do agente. Essas funções, apesar de fazerem parte do interpretador, devem ser fornecidas pelo projetista do agente juntamente com o programa AgentSpeak(L), exceto nas situações em que funções de seleção *default*[11], que são muito simples, possam ser utilizadas.

Os trabalhos anteriores sobre AgentSpeak(L) não discutem como projetistas podem especificar essas funções. Em princípio linguagens de programação tradicionais devem ser usadas para a definição de funções de seleção *ad hoc*. A extensão de AgentSpeak(L) apresentada em [3] visava justamente esse problema, propondo a geração automática e eficiente de funções de seleção de intenções. A linguagem estendida permite expressar relações entre planos, bem como critérios quantitativos para sua execução. Usa-se, então, escalonamento de tarefas baseado em teoria de decisão para guiar as escolhas feitas pela função de seleção de intenções.

Como dito acima, duas estruturas importantes para o interpretador abstrato são o conjunto de eventos e o conjunto de intenções. *Intenções* são cursos de ações com os quais um agente se compromete para tratar certos eventos. Cada intenção é uma pilha de planos parcialmente instanciados.

Eventos causam o início da execução de planos que tem eventos ativadores relevantes. Eventos podem ser *externos*, quando originados da percepção do ambiente (i.e., adição ou remoção de crenças resultantes do processo de revisão de crenças); ou *internos*, quando gerados pela execução de planos do agente (um subobjetivo em um plano gera um evento do tipo "adição de objetivo de realização"). No último caso, o evento é acompanhado da intenção

[11] As funções de seleção *default* são as seguintes: S_E seleciona eventos sem nenhuma prioridade, utilizando a ordem em que eles são inseridos no conjunto de eventos; a função S_{Ap} utiliza a ordem em que os planos foram escritos no programa fonte AgentSpeak(L) para a escolha entre diversos planos aplicáveis; e a função S_I funciona, por assim dizer, como um escalonador *round-robin*, executando uma ação de cada uma das intenções ativas por vez.

que o gerou (o plano escolhido para aquele evento será colocado no topo daquela intenção). Eventos externos criam novas intenções representando diferentes focos de atenção na atuação do agente no ambiente.

A seguir o funcionamento de um interpretador AgentSpeak(L) é apresentado em detalhe, com o auxílio da Figura 4 (reproduzida de [42]). Na figura conjuntos de crenças, eventos, planos e intenções são representados por retângulos. Losangos representam a seleção de um elemento de um conjunto. Círculos representam alguns dos processos envolvidos na interpretação de programas AgentSpeak(L).

A cada ciclo de interpretação de um programa AgentSpeak(L), a lista de eventos é atualizada com o resultado do processo de revisão de crenças. Assume-se que as crenças são atualizadas pela percepção e que sempre que houverem mudanças na base de crenças do agente, isso implica na inserção de um evento no conjunto de eventos. Essa função de revisão de crenças não é parte de um interpretador AgentSpeak(L), mas é um componente que deve estar presente na arquitetura geral do agente (implementações de interpretadores AgentSpeak(L) tipicamente fornecem uma função de revisão de crenças simples utilizada como *default*).

Depois de S_E selecionar um evento, o interpretador AgentSpeak(L) tem que unificar aquele evento com eventos ativadores nas cabeças dos planos presentes na biblioteca de planos. Isso gera um conjunto de todos os *planos relevantes* para o evento escolhido. Pela verificação dos contextos de planos que seguem logicamente das crenças do agente, AgentSpeak(L) determina o conjunto de *planos aplicáveis* (planos que podem ser usados, na situação presente, para tratar o evento selecionado naquele ciclo). Depois, S_{Ap} escolhe, entre os planos do conjunto de planos aplicáveis, um único plano aplicável que se torna o *meio pretendido* para o tratamento daquele evento, e coloca o plano no topo de uma intenção existente (se o evento for interno), ou cria uma nova intenção no conjunto de intenções (se o evento for externo, i.e., gerado por percepção do ambiente), definindo um novo "foco de atenção" do agente.

Nesse estágio, resta apenas a seleção de uma única intenção para ser executada no ciclo. A função S_I seleciona uma intenção do agente (i.e. uma das pilhas de planos parcialmente instanciados que se encontram dentro do conjunto de intenções, cada uma representando um dos focos de atenção do agente). No topo dessa intenção existe uma instância de um plano da biblioteca de planos, e a fórmula no início do corpo do plano é executada. Isso implica em uma ação básica a ser realizada pelo agente no ambiente, na geração de um evento interno (se a fórmula selecionada for um objetivo de realização) ou na execução de um objetivo de teste (através do acesso à base de crenças). Caso a fórmula seja um objetivo de realização, simplesmente um evento do tipo "adição de objetivo de realização" é adicionado ao conjunto de eventos, acompanhado da intenção que gerou o evento. Essa intenção tem que ser removida do conjunto de intenções, pois ela fica suspensa até que o evento interno seja

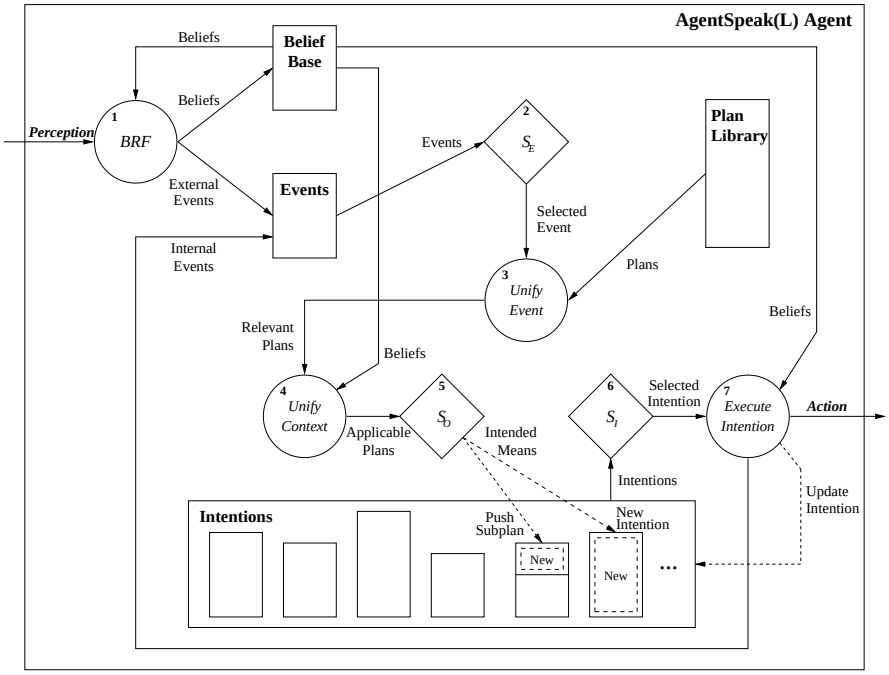

Figura 4. Ciclo de interpretação de um programa AgentSpeak(L) [42].

escolhido pela função S_E. Quando uma instância de plano for escolhida como meio pretendido para tratar este evento, o plano é colocado no topo da pilha de planos daquela intenção, e ela é retornada para o conjunto de intenções (podendo novamente ser selecionada por S_I).

Se a fórmula a ser executada é a realização de uma ação básica ou a execução de um objetivo de teste, a fórmula deve ser removida do corpo da instância de plano que se encontra no topo da intenção selecionada. No caso da execução de um objetivo de teste, a base de crenças será inspecionada para encontrar um átomo de crença que unifica com o predicado de teste. Se uma unificação for possível, instanciações de variáveis podem ocorrer no plano parcialmente instanciado; após isto, o objetivo de teste pode ser removido do conjunto de intenções, pois já foi realizado. No caso de uma ação básica a ser executada, o interpretador simplesmente informa ao componente da arquitetura do agente que é responsável pela atuação sobre o ambiente qual ação é requerida, podendo também remover a ação[12] do conjunto de intenções. Quando todas as fórmulas no corpo de um plano forem removidas (i.e., tiverem

[12]Em algumas implementações, e.g. [3], o interpretador espera um retorno do ambiente simulado para saber se a ação pode ser executada ou não. O plano falha caso o retorno do ambiente não seja positivo.

sido executadas), o plano é removido da intenção, tal como o objetivo de realização que o gerou, se esse for o caso, é removido do início do corpo do plano abaixo daquele na pilha de planos daquele foco de atenção. O ciclo de execução termina com a execução de uma fórmula do corpo de um plano pretendido, e AgentSpeak(L) começa um novo ciclo, com a verificação do estado do ambiente após a ação do agente sobre ele, a geração dos eventos adequados, e continuando a execução de um ciclo de raciocínio do agente como descrito acima.

O objetivo desta seção foi apresentar informalmente alguns aspectos importantes da semântica da linguagem AgentSpeak(L). A semântica formal da linguagem pode ser encontrada nos seguintes artigos [45, 7, 46] (o último deles trata especificamente de aspectos de comunicação entre agentes).

4 Estudos de Caso

Esta seção mostra dois exemplos bem simples de programas escritos em AgentSpeak(L). O fato de os programas serem simples é uma vantagem para a assimilação dos conceitos básicos da linguagem, porém dificultam a compreensão da importância das abstrações da arquitetura BDI em sistemas mais complexos. O objetivo aqui não é mostrar o tipo de sistema em que este paradigma é particularmente apropriado, mas especificamente mostrar exemplos das construções básicas da linguagem AgentSpeak(L). Ambos exemplos foram utilizados em trabalhos que propõem o uso da técnica de *model checking* (verificação de modelos) para a verificação formal de sistemas programados em AgentSpeak(L) (este assunto é discutido na Seção 5.1).

4.1 Robôs Coletores de Lixo em Marte

O cenário utilizado nesta seção foi apresentado em [14] e lembra também o cenário utilizado em [48]; o código apresentado aqui para este cenário foi introduzido anteriormente em [5]. Dois robôs estão coletando lixo no planeta Marte. O robô `r1` procura por lixos depositados no solo do planeta e quando algum lixo é encontrado, o robô coleta o lixo e o leva para o local onde `r2` encontra-se, larga o lixo lá e retorna ao local onde o lixo foi encontrado para continuar sua busca a partir daquela posição anterior. O robô `r2` está posicionado ao lado de um incinerador; todo o lixo levado por `r1` é colocado no incinerador. Os pedaços de lixo são colocados randomicamente em uma grade que define o território do planeta[13]. Outra fonte de não-determinismo é a imprecisão do braço do robô em pegar os pedaços de lixo. A ação de pegar o lixo pode falhar, mas o mecanismo é bom o suficiente para não falhar mais que duas vezes seguidas; ou seja, no pior caso `r1` irá tentar apanhar o lixo três vezes.

[13]Cada posição da grade representa uma área de território de tamanho limitado pelo alcance dos sensores do robô. Qualquer lixo dentro da área de território representado por uma posição na grade é instantaneamente percebido pelo agente.

O código AgentSpeak(L) para `r1` é dado abaixo. No código, cada plano é anotado com um *label* para que se possa referir ao plano no texto que segue. As ações que possuem um ponto (‘.’) em seu nome, denotam ações internas, uma noção introduzida em [3]. Essas ação são executadas internamente pelo agente, e não afetam o ambiente como as ações básicas que o agente executa.

Agent `r1`

Beliefs

```
pos(r2,2,2).
checking(slots).
```

Plans

```
+pos(r1,X1,Y1) : checking(slots) & not(garbage(r1))        (p1)
   <- next(slot).

+garbage(r1) : checking(slots)                             (p2)
   <- !stop(check);
      !take(garb,r2);
      !continue(check).

+!stop(check) : true                                       (p3)
   <- ?pos(r1,X1,Y1);
      +pos(back,X1,Y1);
      -checking(slots).

+!take(S,L) : true                                         (p4)
   <- !ensure_pick(S);
      !go(L);
      drop(S).

+!ensure_pick(S) : garbage(r1)                             (p5)
   <- pick(garb);
      !ensure_pick(S).

+!ensure_pick(S) : true <- true.                           (p6)

+!continue(check) : true                                   (p7)
   <- !go(back);
      -pos(back,X1,Y1);
      +checking(slots);
      next(slot).

+!go(L) : pos(L,Xl,Yl) & pos(r1,Xl,Yl)                     (p8)
   <- true.
```

```
+!go(L) : true                                                    (p9)
   <- ?pos(L,Xl,Yl);
      moveTowards(Xl,Yl);
      !go(L).
```

A crença inicial do agente é sobre a posição do agente `r2` na grade que define o território do planeta em que `r1` deve procurar lixo, e que sua tarefa inicial será verificar os diversos pontos do território, procurando lixo. Todos os planos são explicados abaixo.

O plano `p1` é usado quando o agente percebe que ele está numa nova posição e está procurando por lixo. Se não há lixo percebido naquele ponto, só o que deve ser feito é a ação básica `next(slot)` que move o robô para o próximo ponto na grade (com exceção do local do incinerador, pois o lixo nessa posição é tratado por `r2`). Note que essa é uma ação básica do ponto de vista do agente: assume-se que o robô possui os mecanismos físicos necessários para se mover a uma *próxima* posição no território, seguindo algum caminho pré-determinado de tal forma que todas as posições sejam ordenadas e o mecanismo pode então detectar quando não há mais nenhuma posição seguinte a ser verificada.

O ambiente simulado provê a informação sobre a existência de lixo nas posições dos robôs `r1` e `r2`, quando o agente realiza percepção do ambiente. Quando `r1` percebe lixo em sua posição, a crença `garbage(r1)` é adicionada à base de crenças de tal maneira que o plano `p2` pode ser então usado. A tarefa de lidar com um pedaço de lixo percebido é decomposta em três partes, envolvendo subobjetivos de realização que garantem que: (i) o robô irá parar de procurar por lixo de maneira consistente (lembrando seu último ponto de procura para que a tarefa possa ser continuada daquele ponto); (ii) o lixo seja levado até a posição de `r2`; e (iii) a tarefa de procurar por lixo será, então, retomada. Cada um desses objetivos são realizados respectivamente pela execução dos três planos que seguem.

Quando o agente pretende atingir o subobjetivo (i) acima, o plano `p3` é sua única opção, e ele é sempre aplicável (já que seu contexto é vazio). O agente recupera da sua base de crenças sua posição atual (a posição é inserida na base de crenças através da percepção do ambiente). O agente então toma nota da informação de para onde deve retornar para a continuação da sua busca por lixos. Isso é feito pela adição de uma crença na base de crenças: `+pos(back,X1,Y1)`. O agente remove da sua base de crenças a informação de que está procurando por lixo, e após isso o agente terá o objetivo de levar o lixo até a posição de `r2` e retornar.

O subobjetivo (ii) é tratado pelo plano `p4`, que diz que para o robô levar o lixo até a posição de `r2`, ele deve coletar o lixo e atingir o subobjetivo de ir para a posição de `r2` e, quando chegar lá, ele poderá finalmente largar o lixo. Note que `pick(garb)` e `drop(garb)` são ações básicas, ou seja, são coisas que o robô pode realizar fisicamente no ambiente por meio de seu hardware.

Os planos `p5` e `p6` juntos asseguram que o robô irá continuar tentando pegar um pedaço de lixo até que ele não possa mais perceber o lixo na grade (i.e., até que a ação de coleta seja realizada com sucesso). Lembre que o mecanismo de coleta é impreciso, e o robô pode ter que tentar algumas vezes até obter êxito na execução da ação.

O plano `p7` é usado para que o agente continue a tarefa de verificar a existência de lixo na grade que representa o espaço de território a ser limpo. O agente precisa atingir o subobjetivo de retornar para sua posição anterior (`!go(back)`), e uma vez lá ele poderá remover a nota que fez sobre aquela posição, lembrando que voltou ao estado de procura por lixo e, assim, procedendo para o próximo ponto na grade.

Os últimos dois planos são usados para atingir o objetivo de ir para uma posição específica na grade (representada pela constante com a qual a variável `L` está instanciada). O plano `p9` recupera a crença que o agente possui sobre a posição na grade em que fica a localidade `L`, move-se uma posição na grade em direção ao ponto referente às coordenadas em questão `moveTowards(Xl,Yl)`, e volta a ter o objetivo de mover-se (continuar movendo-se) em direção a `L`; note que o plano é recursivo. O plano `p8` estabelece o fim da recursão dizendo que não há mais o que fazer para atingir o objetivo de ir em direção a `L` se o agente já está naquela posição (o plano anterior não mais será aplicável neste caso).

O agente `r2` é definido pelo código AgentSpeak(L) abaixo. Tudo o que ele faz é queimar os pedaços de lixo (`burn(garb)`) quando ele percebe que há lixo nessa posição (`+garbage(r2)`).

Agent `r2`

```
+garbage(r2) : true
   <- burn(garb).
```

Crenças `garbage(r2)` são adicionadas à base de crenças do agente a partir da percepção do ambiente, sempre que existir lixo no ponto da grade onde `r2` se encontra.

4.2 Um Modelo Abstrato de Leilões

Nessa seção será descrito um cenário simplificado de leilões, utilizado anteriormente em [4]. O ambiente anuncia 10 leilões e designa quem é o vencedor em cada um (aquele com o lance mais alto). Existem três agentes participando desses leilões com estratégias de lance simplificadas.

Agent `ag1`

```
+auction(N) : true
   <- place_bid(N,6).
```

O agente `ag1` é um agente que oferece 6 como lance sempre que o ambiente anuncia um novo leilão.

Agent `ag2`

```
myself(ag2).
bid(ag2,4).
ally(ag3).

+auction(N) : myself(I) & ally(A) & not(alliance(A,I))
   <- ?bid(I,B); place_bid(N,B).

+auction(N) : alliance(A,I)
   <- place_bid(N,0).

+alliance(A,I) : myself(I) & ally(A)
   <- ?bid(I,B);
      .send(A,tell,bid(I,B));
      .send(A,tell,alliance(A,I)).
```

O agente `ag2` oferece 4, caso não tenha combinado uma aliança com `ag3`, caso em que ele oferece 0 (já que `ag3` fará um lance em nome da aliança entre eles). Quando `ag2` recebe uma mensagem de `ag3` propondo uma aliança, a crença `alliance(ag3,ag2)` é adicionada à base de crenças de `ag2`; uma função de confiança (*trust*) *default* é utilizada. Isso constitui um evento ativador para um último plano que informa ao `ag3` o quanto `ag2` costuma oferecer como lance e confirma que `ag2` concorda em formar uma aliança com `ag3`.

Agent `ag3`

```
myself(ag3).
bid(ag3,3).
ally(ag2).
threshold(3).

+auction(N) : threshold(T) & .gte(T,N)
   <- !bid_normally(N).
```

```
+auction(N) : myself(I) & winner(I)
              & ally(A) & not(alliance(I,A))
   <- !bid_normally(N).

+auction(N) : myself(I) & not(winner(I))
              & ally(A) & not(alliance(I,A))
   <- !alliance(I,A);
      !bid_normally(N).

+auction(N) : alliance(I,A)
   <- ?bid(I,B); ?bid(A,C);
      .plus(B,C,D); place_bid(N,D).

+!bid_normally(N) : true
   <- ?bid(I,B); place_bid(N,B).

+!alliance(I,A) : true
   <- .send(A,tell,alliance(I,A)).
```

O agente `ag3` tenta ganhar os primeiros `T` leilões, onde `T` é um limiar (*threshold*) armazenado em sua base de crenças. Se ele não ganha nenhum leilão até aquele limiar, ele tenta então fazer uma aliança com `ag2` (através de um envio de mensagem). Quando `ag2` confirma a aliança, então `ag3` apresenta em nome dos dois um lance com a soma de suas lances usuais.

5 Trabalhos Relacionados

5.1 Recentes Desenvolvimentos em AgentSpeak(L)

A partir da proposta original apresentada por Rao [48], vários outros autores têm levado adiante a investigação de diferentes aspectos de AgentSpeak(L). Em [22] um interpretador abstrato para essa linguagem foi formalmente especificado através da linguagem Z. Muitos dos elementos nessa formalização foram primeiramente apresentados em [21], que apresenta uma especificação formal para dMARS. Isso deve-se ao fato de AgentSpeak(L) ser fortemente baseada nas experiências com o sistema dMARS [35], sucessor do PRS [29], ambos sistemas que utilizam a arquitetura BDI.

Algumas extensões de AgentSpeak(L) foram propostas em [3], e um interpretador para a versão estendida foi apresentado. As extensões tinham por objetivo a obtenção de uma linguagem de programação que fosse mais prática. A linguagem estendida permite a especificação de relações entre planos e critérios quantitativos para sua execução. O interpretador usa, então, um escalonador de tarefas baseado em teoria de decisão para guiar, automatica-

mente, as escolhas relacionadas à função de seleção de intenções de um agente.

Em [45], uma semântica operacional para AgentSpeak(L) foi dada com base na abordagem estrutural de Plotkin [47], uma notação mais usual do que Z para dar semântica a linguagens de programação. Posteriormente, essa semântica operacional foi utilizada na especificação de um *framework* para a construção de provas de propriedades BDI da linguagem AgentSpeak(L) [6]. A combinação dos princípios da "tese da assimetria" (*assymetry thesis*) satisfeitos por um agente AgentSpeak(L) foi apresentado inicialmente em [6]; as provas detalhadas dessas propriedades foram apresentadas em [7]. Esses princípios são relevantes para se assegurar a racionalidade dos agentes programados em AgentSpeak(L) com base nos princípios de racionalidade segundo a teoria BDI. Em [46], a semântica operacional de AgentSpeak(L) foi estendida para dar semântica à comunicação entre agentes baseada na teoria dos atos de fala.

Em [9] foi apresentado o uso de técnicas de *model checking* para a verificação formal de programas AgentSpeak(L) através de um conjunto de ferramentas chamado CASP. CASP traduz uma versão simplificada de AgentSpeak(L) em linguagens que podem ser utilizadas em verificadores de modelos para a lógica temporal linear [1]. A tradução para Promela foi apresentada em [4], e a tradução para Java em [5]. A verificação de modelo é então feita com o uso de Spin [31] no primeiro caso, e com JPF2 [60] no segundo. Esse trabalho em verificação de modelos para programas AgentSpeak(L) usa as definições de modalidades BDI em termos da semântica operacional de AgentSpeak(L) apresentadas em [7] para mostrar de forma precisa como as especificações BDI (a serem verificadas para sistemas com múltiplos agentes AgentSpeak(L)) são interpretadas.

Poucas aplicações foram desenvolvidas com AgentSpeak(L) até o momento, dado que a sua implementação prática é muito recente e ainda requer um trabalho mais aprofundado de experimentação para que se torne uma linguagem de programação poderosa. O mesmo se aplica a outras linguagens baseadas em agentes (veja a próxima seção). Apesar do trabalho na área datar do início dos anos 90, um grande número de questões está ainda em aberto. Entre as aplicações existentes, desenvolvidas em AgentSpeak(L), podemos mencionar uma simulação de aspectos sociais do crescimento urbano [8] e um robô carregador encarregado do armazém de uma fábrica, que funciona em um ambiente de realidade virtual [58]. O primeiro trabalho foi desenvolvido como um estudo de caso para a plataforma MAS-SOC[14] de simulação social baseada em agentes em que cada agente de uma simulação é implementado em AgentSpeak(L). O segundo trabalho apresenta uma arquitetura de duas camadas com o interpretador AgentSpeak(L) em um nível, e, em outro nível, um sistema articulado para a modelagem de personagens 3D em ambientes virtuais. Tal arquitetura tem por objetivo permitir o uso de agentes autônomos sofisticados em sistemas de realidade virtual ou outras

[14] A URL do projeto MAS-SOC é <`http://www.inf.ufrgs.br/~massoc/`>. Nesta página também podem ser encontrados *links* para outros recursos relacionados à linguagem AgentSpeak(L).

aplicações baseadas em animações[15] tridimensionais.

5.2 Outras Linguagens de Programação Orientadas a Agentes

Desde o artigo seminal em programação orientada a agentes de Shoham [55], várias outras linguagens nesse paradigma foram propostas, seguindo diferentes abordagens e influências. Naquele artigo, foi apresentada a linguagem Agent0, que é inspirada na arquitetura BDI com sintaxe baseada em LISP. A seguir, serão mencionadas algumas das outras linguagens orientada a agentes que apareceram na literatura de sistemas multiagente desde então.

Concurrent METATEM [24] é baseada em lógica temporal executável (essa linguagem, de fato, é anterior ao trabalho de Shoham). Originalmente, a idéia era de que cada agente fosse executado diretamente de uma especificação em lógica temporal de tempo linear. Trabalhos mais recentes adicionam outras modalidades à linguagem, permitindo aos desenvolvedores projetar agentes baseados em abstrações antropomórficas, de maneira similar a agentes BDI [25]. Outros trabalhos em Concurrent METATEM têm permitido aos usuários a especificação de grupos (aninhados) de agentes, com características interessantes de comunicação e acesso para os agentes em diferentes grupos [26].

Uma linguagem recentemente introduzida na literatura da área chama-se STAPLE [38]. Essa linguagem é baseada na teoria de intenções conjuntas de Cohen e Levesque, que é formalizada na lógica apresentada em [17]. A plataforma IMPACT [57] tem como objetivo permitir a "agentificação" de código legado, e a semântica de agentes IMPACT é baseada em lógica deontológica [2]. Algumas linguagens de programação apresentadas na literatura estendem sistemas de programação que tem demonstrado grande aplicação em inteligência artificial de modo geral ao conceito de sistemas de múltiplos agentes situados. Um exemplo é a linguagem ConGolog [18], uma linguagem de programação concorrente baseada em cálculo de situações. Outro exemplo é $\mathcal{MINERVA}$, uma arquitetura de agente [41, 40] baseada em programação em lógica dinâmica.

Outras linguagens BDI, tais como 3APL [30], foram inspiradas em AgentSpeak(L) modificando-a de alguma maneira. 3APL, por exemplo, não inclui o conceito de evento ou de uma estrutura intencional (como o conjunto de intenções em AgentSpeak(L)); isso faz com que 3APL se distancie um pouco da concepção original da arquitetura BDI. Por outro lado, a linguagem tem outras características, tais como regras especiais que operam nos objetivos do agente durante sua execução, que são úteis, por exemplo, no tratamento de planos que falham. Em trabalho recente, a linguagem Dribble [59] foi proposta, entendendo 3APL com a noção de objetivos declarativos. Outra abordagem a agentes BDI é o cálculo Ψ [36], baseado no cálculo π [44], com o objetivo de dar suporte teórico para linguagens de programação

[15] Exemplos de animações realizadas com esta arquitetura podem ser encontrados na URL <`http://www.inf.ufrgs.br/cg/ras/`>.

visuais [36, 37].

Uma linguagem fortemente baseada em AgentSpeak(L), chamada AgentTalk, foi recentemente disponibilizada pela Internet[16]. Também encontram-se disponíveis na Internet, plataformas JAVA para o desenvolvimento de agentes BDI, tais como JAM[17] [33] e JACK[18] [32]. A grande diferença entre essas ferramentas e linguagens BDI como AgentSpeak(L) e 3APL é que essas últimas têm semântica formal, o que possibilita a verificação formal de sistemas programados com estas linguagens; além disto, pode-se estabelecer, com rigorismo ainda maior, uma relação formal com a teoria BDI original, vinda da literatura de filosofia sobre raciocínio prático e que influencia todo o trabalho em agentes racionais (através de trabalhos teóricos que estão em desenvolvimento).

6 Conclusão

Após uma década desde a primeira publicação utilizando o termo "programação orientada a agentes" [55], pode-se dizer que as linguagens de programação orientadas a agentes encontram-se ainda em sua infância. Muita pesquisa em aspectos formais dessas linguagens já foi desenvolvida, mas ainda há muitas questões em aberto. Além disto, só recentemente estas linguagem com base formal e realmente substanciadas nas idéias da área de sistemas multiagente tiveram interpretadores disponibilizados publicamente. Com isto, pouca experimentação prática, com sistemas de grande porte, foi desenvolvida até o momento. As relações com os demais níveis do processo de engenharia de software orientada a agente também não estão bem consolidadas ainda.

De toda a forma, dado o grande sucesso recente da área de sistemas multiagente, influenciando o trabalho nas mais diversas área da computação, é muito provável que o próximo grande paradigma de programação seja o paradigma de orientação a agentes, alterando substancialmente a forma como sistemas computacionais serão concebidos. As linguagens de programação com esta orientação certamente terão um papel importante nesse processo, mas exatamente a forma como isto acontecerá é difícil prever. O objetivo deste texto foi dar uma visão geral de alguns conceitos que serão importantes nesse paradigma, e fornecer referências para os trabalhos na área, esperando que as idéias apresentadas aqui ajudem também na compreensão da significativa diferença desta forma de concepção de sistemas computacionais, desde o uso de noções mentalísticas até o uso de comunicação de alto nível (baseada na teoria dos atos de fala), na concepção de entidades de software auto-motivadas e autônomas.

[16] Essa linguagem tem um interpretador escrito na linguagem funcional Scheme. A única referência a esse trabalho é a URL: <`http://www.cs.rmit.edu.au/~winikoff/agenttalk/`>.
[17] URL: <`http://www.marcush.net`>.
[18] URL: <`http://www.agent-software.com`>.

Agradecimentos

Como pode ser observado pelas referências relacionadas a AgentSpeak(L), este artigo deriva de vários trabalhos (anteriores ou em andamento) realizados com a cooperação de diversos pesquisadores e alunos, e parcialmente financiados pelo CNPq e pela FAPERGS. Registramos nosso sincero agradecimento aos seguintes co-autores, em ordem alfabética: Marcelo G. de Azambuja, Deniel M. Basso, Ana L.C. Bazzan, Antônio Carlos da Rocha Costa, Guilherme Drehmer, Michael Fisher, Jomi F. Hübner, Rafael de O. Jannone, Romulo Krafta, Victor Lesser, Rodrigo Machado, Álvaro F. Moreira, Fabio Y. Okuyama, Denise de Oliveira, Carmen Pardavila, Maíra R. Rodrigues, Rosa M. Vicari, Willem Visser, Michael Wooldridge.

Referências

[1] E Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 997–1072. Elsevier Science, Amsterdam, 1990.

[2] Lennart Åqvist. Deontic logic. In Dov M. Gabbay and Franz Günthner, editors, *Handbook of Philosophical Logic: Volume II: Extensions of Classical Logic*, chapter II.11, pages 605–714. D. Reidel Publishing Company, 1984.

[3] Rafael H. Bordini, Ana L. C. Bazzan, Rafael O. Jannone, Daniel M. Basso, Rosa M. Vicari, and Victor R. Lesser. AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In Cristiano Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002), 15–19 July, Bologna, Italy*, pages 1294–1302, New York, NY, 2002. ACM Press.

[4] Rafael H. Bordini, Michael Fisher, Carmen Pardavila, and Michael Wooldridge. Model checking AgentSpeak. In Jeffrey S. Rosenschein, Tuomas Sandholm, Wooldridge Michael, and Makoto Yokoo, editors, *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2003), Melbourne, Australia, 14–18 July*, pages 409–416, New York, NY, 2003. ACM Press.

[5] Rafael H. Bordini, Michael Fisher, Willem Visser, and Michael Wooldridge. Verifiable multi-agent programs. In *Proceedings of the First International Workshop on Programming Multiagent Systems: languages, frameworks, techniques and tools (PROMAS-03), to be held with AAMAS-03, 15 July, 2003, Melbourne, Australia*, 2003. To appear.

[6] Rafael H. Bordini and Álvaro F. Moreira. Proving the asymmetry thesis principles for a BDI agent-oriented programming language. In Jürgen Dix, João Alexandre Leite,

and Ken Satoh, editors, *Proceedings of the Third International Workshop on Computational Logic in Multi-Agent Systems (CLIMA-02), 1st August, Copenhagen, Denmark*, Electronic Notes in Theoretical Computer Science 70(5). Elsevier, 2002. URL: <http://www.elsevier.nl/locate/entcs/volume70.html>. CLIMA-02 was held as part of FLoC-02. This paper was originally published in Datalogiske Skrifter number 93, Roskilde University, Denmark, pages 94–108.

[7] Rafael H. Bordini and Álvaro F. Moreira. Proving BDI properties of agent-oriented programming languages: The asymmetry thesis principles in AgentSpeak(L). *Annals of Mathematics and Artificial Intelligence*, 2003. Accepted for publication in a Special Issue on Computational Logic in Multi-Agent Systems.

[8] Rafael H. Bordini, Fabio Y. Okuyama, Denise de Oliveira, Guilherme Drehmer, and Romulo C. Krafta. The MAS-SOC approach to multi-agent based simulation. In Gabriela Lindemann, Daniel Moldt, Mario Paolucci, and Bin Yu, editors, *Proceedings of the First International Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA'02), 16 July, 2002, Bologna, Italy, held with AAMAS02*, Lecture Notes in Artificial Intelligence, Berlin, 2003. Springer-Verlag. To appear.

[9] Rafael H. Bordini, Willem Visser, Michael Fisher, Carmen Pardavila, and Michael Wooldridge. Model checking multi-agent programs with CASP. In *Proceedgins of the Fifteenth Conference on Computer-Aided Verification (CAV-2003), Boulder, CO, 8–12 July*, 2003. Tool description. To appear.

[10] Rafael Heitor Bordini. *Contributions to an Anthropological Approach to the Cultural Adaptation of Migrant Agents*. PhD thesis, University of London, 1999.

[11] Rafael Heitor Bordini, Renata Vieira, and Álvaro Freitas Moreira. Fundamentos de sistemas multiagentes. In Carlos Eduardo Ferreira, editor, *Anais do XXI Congresso da Sociedade Brasileira de Computação (SBC2001), Volume 2, XX Jornada de Atualização em Informática (JAI), 30 de julho – 3 de agosto, Fortaleza-CE, Brasil*, chapter 1, pages 3–41. Sociedade Brasileira de Computação, 2001.

[12] Michael E. Bratman. *Intentions, Plans and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.

[13] Michael E. Bratman, David J. Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.

[14] John A. Campbell and Mark d'Inverno. Knowledge interchange protocols. In Yves Demazeau and Jean-Pierre Müller, editors, *Decentralized A.I.—Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'89), 16–18 August, Cambridge, 1989*, pages 63–80. Elsevier Science B.V., Amsterdam, 1990.

[15] Cristiano Castelfranchi. Social power: a point missed in multi-agent DAI and HCI. In Yves Demazeau and Jean-Pierre Müller, editors, *Decentralized A.I.—Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'89), 16–18 August, Cambridge, 1989*, pages 49–62. Elsevier Science B.V., Amsterdam, 1990.

[16] Cristiano Castelfranchi, Maria Miceli, and Amadeo Cesta. Dependence relations among autonomous agents. In Eric Werner and Yves Demazeau, editors, *Decentralized A.I. 3—Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World, 5–7 August, 1991, Kaiserslautern, Germany*, pages 215–227. Elsevier Science B.V., Amsterdam, 1992.

[17] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.

[18] Giuseppe de Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog: A concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169, 2000.

[19] Yves Demazeau. From cognitive interactions to collective behaviour in agent-based systems. In *Proceedings of the European Conference on Cognitive Science*, Saint-Malo, April 1995.

[20] Daniel C. Dennett. *The Intentional Stance*. The MIT Press, Cambridge, MA, 1987.

[21] Mark d'Inverno, David Kinny, Michael Luck, and Michael Wooldridge. A formal specification of dMARS. In Munindar P. Singh, Anand S. Rao, and Michael Wooldridge, editors, *Intelligent Agents IV—Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97), Providence, RI, 24–26 July, 1997*, number 1365 in Lecture Notes in Artificial Intelligence, pages 155–176. Springer-Verlag, Berlin, 1998.

[22] Mark d'Inverno and Michael Luck. Engineering AgentSpeak(L): A formal computational model. *Journal of Logic and Computation*, 8(3):1–27, 1998.

[23] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.

[24] Michael Fisher. A survey of concurrent METATEM—the language and its applications. In Dov M. Gabbay and H. J. Ohlbach, editors, *Temporal Logics—Proceedings of the First International Conference*, number 827 in Lecture Notes in Artificial Intelligence, pages 480–505. Springer-Verlag, Berlin, 1994.

[25] Michael Fisher and Chiara Ghidini. The ABC of rational agent modelling. In Cristiano Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002), 15–19 July, Bologna, Italy*, pages 849–856, New York, NY, 2002. ACM Press.

[26] Michael Fisher, Chiara Ghidini, and Benjamin Hirsch. Organising logic-based agents. In *Proceedings of the Second NASA/IEEE Goddard Workshop on Formal Approaches to Agent Based Systems (FAABS 2002), Greenbelt, MD, October 29-31*, 2002.

[27] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In Jörg P. Müller, Michael J. Wooldridge, and Nicholas R. Jennings, editors, *Intelligent Agents III—Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96), ECAI'96 Workshop, Budapest, Hungary*, number 1193 in Lecture Notes in Artificial Intelligence, pages 21–35. Springer-Verlag, Berlin, 1997. URL: `http://www.msci.memphis.edu/~franklin/aagents.html`.

[28] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, July 1994. URL: `http://logic.stanford.edu/sharing/papers/`.

[29] Michael P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87), 13–17 July, 1987, Seattle, WA*, pages 677–682, Manlo Park, CA, 1987. AAAI Press / MIT Press.

[30] Koen V. Hindriks and Frank S. and de Boer. Formal semantics for an abstract agent programming language. In Munindar P. Singh, Anand S. Rao, and Michael Wooldridge, editors, *Intelligent Agents IV—Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97), Providence, RI, 24–26 July, 1997*, number 1365 in Lecture Notes in Artificial Intelligence, pages 215–229, Berlin, 1998. Springer-Verlag.

[31] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[32] Nick Howden, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. JACK intelligent agents™ — summary of an agent infrastructure. In *Proceedings of Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, held with the Fifth International Conference on Autonomous Agents (Agents 2001), 28 May – 1 June, Montreal, Canada*, 2001.

[33] Marcus J. Huber. JAM: A BDI-theoretic mobile agent architecture. In *Proceedings of the Third International Conference on Autonomous Agents (Agents-99), 1–5 May, Seattle, WA*, pages 236–243. ACM Press, 1999.

[34] Jomi Fred Hübner. *Um Modelo de Reorganização de Sistemas Multiagentes*. PhD thesis, Universidade de São Paulo, Escola Politécnica, 2003.

[35] David Kinny. The distributed multi-agent reasoning system architecture and language specification. Technical report, Australian Artificial Intelligence Institute, Melbourne, Australia, 1993.

[36] David Kinny. The Ψ calculus: An algebraic agent language. In John-Jules Meyer and Milind Tambe, editors, *Intelligent Agents VIII – Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001), August 1–3, 2001, Seattle, WA*, number 2333 in Lecture Notes in Artificial Intelligence, pages 32–50, Berlin, 2002. Springer-Verlag.

[37] David Kinny. ViP: A visual programming language for plan execution systems. In Cristiano Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002, featuring 6th AGENTS, 5th ICMAS and 9th ATAL), 15–19 July, Bologna, Italy*, pages 721–728, New York, NY, 2002. ACM Press.

[38] Sanjeev Kumar, Philip R. Cohen, and Marcus J. Huber. Direct execution of team specifications in STAPLE. In Cristiano Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002), 15–19 July, Bologna, Italy*, pages 567–568, New York, NY, 2002. ACM Press. Short paper.

[39] Jean Lave. *Cognition in Practice: Mind, Mathematics and Culture in Everyday Life*. Cambridge University Press, 1988.

[40] João Alexandre Leite. *Evolving Knowledge Bases: Specification and Semantics*, volume 81 of *Frontiers in Artificial Intelligence and Applications, Dissertations in Artificial Intelligence*. IOS Press/Ohmsha, Amsterdam, 2003.

[41] João Alexandre Leite, José Júlio Alferes, and Luís Moniz Pereira. $\mathcal{MINERVA}$—a dynamic logic programming agent architecture. In John-Jules Meyer and Milind Tambe, editors, *Intelligent Agents VIII – Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001), August 1–3, 2001, Seattle, WA*, number 2333 in Lecture Notes in Artificial Intelligence, pages 141–157, Berlin, 2002. Springer-Verlag.

[42] Rodrigo Machado and Rafael H. Bordini. Running AgentSpeak(L) agents on SIM_AGENT. In John-Jules Meyer and Milind Tambe, editors, *Intelligent Agents VIII – Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001), August 1–3, 2001, Seattle, WA*, number 2333 in Lecture Notes in Artificial Intelligence, pages 158–174, Berlin, 2002. Springer-Verlag.

[43] John-Jules Ch. Meyer. Agent languages and their relationship to other programming paradigms. In Jörg P. Müller, Munindar P. Singh, and Anand S. Rao, editors, *Intelligent Agents V—Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), held as part of the Agents' World, Paris, 4–7 July, 1998*, number 1555 in Lecture Notes in Artificial Intelligence, pages 113–131, Heidelberg, 1999. Springer-Verlag. UCL-CS [RN/98/29].

[44] Robin Milner, Joachim Parrow, and David Walker. A calculus for mobile processes (parts I and II). *Information and Computation*, 100(1):1–40 and 41–77, September 1992.

[45] Álvaro F. Moreira and Rafael H. Bordini. An operational semantics for a BDI agent-oriented programming language. In *Proceedings of the Workshop on Logics for Agent-Based Systems (LABS-02), held in conjunction with the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002), April 22–25, Toulouse, France*, pages 45–59, 2002.

[46] Álvaro F. Moreira, Renata Vieira, and Rafael H. Bordini. Extending the operational semantics of a BDI agent-oriented programming language for introducing speech-act based communication. In *Proceedings of the Workshop on Declarative Agent Languages and Technologies (DALT-03), held with AAMAS-03, 15 July, 2003, Melbourne, Australia*, 2003.

[47] Gordon D. Plotkin. A structural approach to operational semantics. Technical report, Computer Science Department, Aarhus University, Aarhus, 1981.

[48] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Walter Van de Velde and John Perram, editors, *Proceedings of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), 22–25 January, Eindhoven, The Netherlands*, number 1038 in Lecture Notes in Artificial Intelligence, pages 42–55, London, 1996. Springer-Verlag.

[49] Anand S. Rao. Decision procedures for propositional linear-time belief-desire-intention logics. In Michael Wooldridge, Jörg P. Müller, and Milind Tambe, editors, *Intelligent Agents II—Proceedings of the Second International Workshop on Agent Theories, Architectures, and Languages (ATAL'95), held as part of IJCAI'95, Montréal, Canada, August 1995*, number 1037 in Lecture Notes in Artificial Intelligence, pages 33–48, Berlin, 1996. Springer-Verlag.

[50] Anand S. Rao and Michael P. Georgeff. An abstract architecture for rational agents. In Charles Rich, William R. Swartout, and Bernhard Nebel, editors, *Proceedings of*

the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92), 25–29 October, Cambridge, MA, pages 439–449, San Mateo, CA, October 1992. Morgan Kaufmann.

[51] Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In Victor Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95), 12–14 June, San Francisco, CA*, pages 312–319, Menlo Park, CA, 1995. AAAI Press / MIT Press.

[52] Anand S. Rao and Michael P. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 8(3):293–343, 1998.

[53] Maíra Ribeiro Rodrigues, Antônio Carlos da Rocha Costa, and Rafael Heitor Bordini. A system of exchange values to support social interactions in artificial societies. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2003), Melbourne, Australia, 14–18 July*, 2003. To appear.

[54] J.S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, Cambridge, MA, 1994.

[55] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[56] Munindar P. Singh, Anand S. Rao, and Michael P. Georgeff. Formal methods in DAI: Logic-based representation and reasoning. In Gerhard Weiß, editor, *Multiagent Systems—A Modern Approach to Distributed Artificial Intelligence*, chapter 8, pages 331–376. MIT Press, Cambridge, MA, 1999.

[57] V. S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, and Robert Ross. *Heterogeneous Agent Systems*. The MIT Press, 2000.

[58] Jorge A. R. Torres, Luciana P. Nedel, and Rafael H. Bordini. Using the BDI architecture to produce autonomous characters in virtual worlds. In *Proceedings of the Fourth International Conference on Interactive Virtual Agents (IVA 2003), Irsee, Germany, 15–17 September*, Lecture Notes in Artificial Intelligence, Berlin, 2003. Springer-Verlag. Short paper, to appear.

[59] Birna van Riemsdijk, Wiebe van der Hoek, and John-Jules Ch. Meyer. Agent programming in dribble: from beliefs to goals using plans. In Jeffrey S. Rosenschein, Tuomas Sandholm, Wooldridge Michael, and Makoto Yokoo, editors, *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2003), Melbourne, Australia, 14–18 July*, pages 393–400, New York, NY, 2003. ACM Press.

[60] Willem Visser, Klaus Havelund, Guillaume Brat, and SeungJoon Park. Model checking programs. In *Proceedings of the Fifteenth International Conference on Automated Software Engineering (ASE'00), 11-15 September, Grenoble, France*, pages 3–12. IEEE Computer Society, 2000.

[61] Gerhard Weiß, editor. *Distributed Artificial Intelligence Meets Machine Learning—Learning in Multiagent Environments*. Number 1221 in Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 1997.

[62] Gerhard Weiß, editor. *Multiagent Systems—A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 1999.

[63] Michael Wooldridge. Intelligent agents. In Gerhard Weiß, editor, *Multiagent Systems—A Modern Approach to Distributed Artificial Intelligence*, chapter 1, pages 27–77. MIT Press, Cambridge, MA, 1999.

[64] Michael Wooldridge. *Reasoning about Rational Agents*. The MIT Press, Cambridge, MA, 2000.

[65] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2002.

[66] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995. URL: `http://www.elec.qmw.ac.uk/dai/pubs`.

[67] Michael J. Wooldridge, Nicholas R. Jennings, and David Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[68] Franco Zambonelli and H. Van Dyke Parunak. Signs of a revolution in computer science and software engineering. In Paolo Petta, Robert Tolksdorf, and Franco Zambonelli, editors, *Proceedings of the Third International Workshop on Engineering Societies in the Agents World (ESAW 2002), Madrid, Spain, 16–17 September*, number 2577 in Lecture Notes in Artificial Intelligence, pages 13–28, Berlin, 2003. Springer-Verlag.