2
Agentes Inteligentes

2.1
Introdução

Computadores não são muito bons em saber o que fazer: toda ação que o computador faz deve ser explicitamente antecipada, planejada e codificada por um programador. Se um programa de computador encontrar uma situação que seu criador não previu, o resultado não costuma ser bom: na melhor das hipóteses, o sistema pode falhar e, na pior, vidas podem ser perdidas (Woo99).

Para a maioria das tarefas, pode-se aceitar os computadores como empregados obedientes, literais e sem imaginação, como em aplicações que processam folhas de pagamento. Entretanto, para um número crescente de aplicações, são necessários sistemas que podem decidir por eles mesmos o que eles precisam fazer para atingir os objetivos para os quais foram construídos. Portanto, um objetivo da ciência da computação moderna é construir programas de computadores que podem agir autonomamente, *softwares* que podem tomar boas decisões sobre quais ações realizar em nome do usuário e executar essas ações. As aplicações variam de pequenos programas que pesquisam a Internet de maneira inteligente comprando e vendendo bens através do comércio eletrônico a módulos espaciais autônomos (Woo00).

Tais sistemas computacionais são conhecidos como agentes. Agentes que devem operar de maneira robusta em ambientes abertos, imprevisíveis e que se modificam rapidamente, onde há grandes possibilidades de que as ações falhem, são chamados de agentes inteligentes ou autônomos (Woo99).

2.2
O que São Agentes

A definição para o termo agente é muito difícil. Não há um consenso universal sobre a definição do termo agente, havendo um grande debate e muita controvérsia sobre o tema. Enquanto há um consenso de que autonomia é algo central para a noção de agente, não há muita concordância sobre outros aspectos de um agente. Parte dessa dificuldade procede do fato de que vários atributos associados com agentes possuírem importância diferente

em domínios diferentes (Woo99, Nwa96, Woo95). Por exemplo, para algumas aplicações, a habilidade de aprendizado através da experiência é de fundamental importância. Já para outras, o aprendizado não só não é importante como é indesejado (Woo99).

Entretanto, algum tipo de definição é importante. Em (Woo95), um agente é definido como um sistema computacional, que está situado em algum ambiente e é capaz de agir de maneira autônoma para atingir os objetivos para os quais foi construído. Note-se que essa definição não diz nada sobre o tipo de ambiente que o agente ocupa: agentes podem ocupar vários tipos de ambientes. E, assim como o conceito de agente, o de autonomia também é difícil de se definir precisamente. Para este trabalho, isso significa que os agentes são capazes de atuar sem intervenção humana ou de outros sistemas: eles possuem controle tanto sobre seus estados internos como sobre seus comportamentos (Woo95, Woo99).

É importante observar que a definição para o termo agente, por não ser universal, varia entre os pesquisadores da área. Por exemplo, a definição em (Nwa96) diz que um agente é um componente de *hardware* e/ou *software* que é capaz de agir de maneira precisa para concluir tarefas para o seu usuário. Note-se que nada é dito sobre a necessidade de o agente ser autônomo.

A figura 2.1 mostra uma visão abstrata de um agente. Neste diagrama, pode-se ver a ação de saída gerada por um agente com o objetivo de afetar o seu ambiente. Na maioria dos domínios de razoável complexidade, um agente não terá controle completo de seu ambiente. Na melhor das hipóteses, ele terá controle parcial, ou seja, poderá influenciá-lo. Do ponto de vista de um agente, isso significa que a mesma ação executada duas vezes em circunstâncias aparentemente idênticas pode vir a ter efeitos completamente diferentes e, em particular, pode não conseguir ter o efeito desejado. Portanto, agentes em ambientes não-triviais devem estar preparados para a possibilidade de falhas. Para completar, os ambientes podem ser (e geralmente são) não-determinísticos (Woo99).

A figura também mostra que o agente coleta uma análise sensorial do ambiente e produz uma ação de saída que o afeta. A interação, representada pelas setas, usualmente é contínua e infinita (Woo99).

Normalmente, um agente tem um repertório de ações disponíveis. Esse conjunto de possíveis ações representa a capacidade efetiva dos agentes, suas habilidades para modificar os seus ambientes. As ações têm pré-condições associadas às mesmas que definem as situações nas quais elas podem ser aplicadas (Woo99).

O problema principal que um agente enfrenta é decidir quais de suas

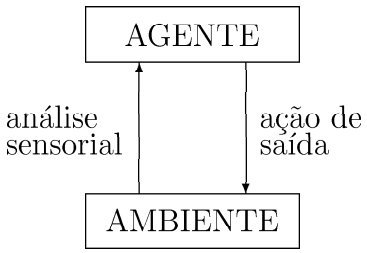

Figura 2.1: Um agente e seu ambiente

ações ele deve executar de modo a melhor satisfazer os objetivos para o qual foi criado. A complexidade do processo de tomada de decisão pode ser afetada por uma variedade de propriedades do ambiente (Woo99). Em (Rus95a), é sugerida uma classificação de propriedades do ambiente:

- Acessível vs inacessível
  Um ambiente acessível é aquele no qual o agente pode obter informações completas, precisas e atualizadas sobre o estado do ambiente. A maioria dos ambientes razoavelmente complexos são inacessíveis (incluindo, por exemplo, o mundo real e a Internet). Quanto mais acessível o ambiente, mais simples fica a construção de agentes para operar nele.
- Determinístico vs não-determinístico
  Um ambiente determinístico é aquele no qual qualquer ação de um agente tem um único efeito garantido; não há incertezas sobre o estado resultante a partir da realização da ação. O mundo real pode ser considerado não-determinístico. Ambientes não-determinísticos apresentam grandes problemas para os construtores de agentes.
- Episódico vs não-episódico
  Em um ambiente episódico, o desempenho de um agente é dependente de um número discreto de episódios, o que não tem ligação com o desempenho de um agente em diferentes cenários. Um exemplo de ambiente episódico seria o de um sistema de ordenação de correspondência (Rus95b). Ambientes episódicos são mais simples da perspectiva do desenvolvedor de agentes porque o agente pode decidir que ação realizar baseado somente no episódio atual; ele não precisa se preocupar com as interações entre o episódio corrente e os futuros.
- Estático vs dinâmico
  Um ambiente estático é aquele em que se considera que não foi modificado exceto pela execução de ações do agente. Um ambiente dinâmico é aquele onde há outros processos afetando-o, e, por isso, se modifica de maneira

fora do controle do agente. O mundo real é um ambiente altamente dinâmico. Também é dinâmico um sistema multi-agentes, pois há vários agentes afetando o ambiente.

– Discreto vs contínuo
Um ambiente é discreto se nele houver um número de ações e percepções finitos e fixos. Um jogo de xadrez é um exemplo de um ambiente discreto e a direção de um táxi é um exemplo de um que é contínuo.

Algo a ser notado é que se o ambiente for suficientemente complexo, o fato de ele ser determinístico não é de muita ajuda: para todos os efeitos, ele pode muito bem ser considerado não-determinístico (Rus95a). A classe mais complexa de ambientes gerais é aquelas com ambientes inacessíveis, não-determinísticos, não-episódicos, dinâmicos e contínuos (Woo99).

Alguns exemplos de agentes (Woo99) que podem ser citados são os controles de sistemas (como um termostato) e a maioria dos *software daemons*.

Para concluir, para este trabalho, agentes são sistemas computacionais que são capazes de agir autonomamente em algum ambiente para atingir seus objetivos para os quais foram construídos. Um agente tipicamente irá sentir o seu ambiente (através de sensores físicos no caso de agentes situados no mundo real ou por sensores de *software* no caso de agentes de *software*), e terá disponível para si um repertório de ações que podem ser executadas para modificar o ambiente, o qual pode responder não-deterministicamente à execução dessas ações (Woo99).

2.3 O que São Agentes Inteligentes

Não se costuma considerar termostatos e *software deamons* como agentes e certamente não como agentes inteligentes. Para considerar um agente como sendo inteligente, é preciso definir o que é inteligência. Novamente, também não é fácil definir esse termo, mas para esse trabalho, um agente inteligente é aquele que é capaz de ter ação autônoma *flexível* de modo a cumprir os objetivos para os quais foi criado (Woo99). O termo flexibilidade significa (Woo98, Woo99):

– Pró-atividade
Agentes inteligentes são capazes de exibir comportamento direcionado a objetivos, tomando a iniciativa para satisfazer os objetivos para os quais foram criados;

– Reatividade
Agentes inteligentes são capazes de perceber seus ambientes (eles podem pertencer a mais de um ambiente simultaneamente) e responder de

uma maneira temporal às mudanças que ocorrem nele para satisfazer os objetivos para os quais foram criados, e

– Habilidade social
Agentes inteligentes são capazes de interagir com outros agentes (e, possivelmente, seres humanos) para satisfazer os objetivos para os quais foram criados, resolvendo seus próprios problemas e ajudando os outros em suas atividades (Bor02).

Essas propriedades são mais complicadas do que podem parecer à primeira vista. Considerando a pró-atividade, não é difícil construir um sistema que exiba um comportamento direcionado a atingir um objetivo. Por exemplo, isso é feito sempre que se escreve um procedimento, uma função ou um método em uma linguagem de programação. Esse tipo de sistema pode ser descrito em termos das hipóteses nas quais ele se baseia (pré-condições) e no efeito que ele acarreta se as hipóteses forem válidas (pós-condições). Os efeitos são seus objetivos. Esse modelo é bom para muitos ambientes. Por exemplo, ele funciona bem quando se considera sistemas funcionais, onde se tem uma entrada x e se produz como saída alguma função $f(x)$ da entrada (Woo99). Nesses casos, tem-se uma pré-condição φ representando o que deve ser verdadeiro no ambiente do programa para que o programa opere corretamente. Tem-se também uma pós-condição ψ que representa o que será verdadeiro no ambiente do programa após o seu término, assumindo que a pré-condição foi satisfeita quando a execução do programa começou (Hoa69, Woo02). Há ainda a hipótese de que o ambiente do programa, caracterizado pela sua pré-condição φ, só é modificado pelas ações do próprio programa. Como pode ser notado, essa hipótese não vale para muitos ambientes.

Porém, para ambientes não-funcionais, esse modelo orientado a objetivos simples, visando a satisfazer uma pós-condição dada uma pré-condição, não é aceitável, já que ele assume hipóteses limitantes importantes. Em particular, ele assume que o ambiente não se modifica durante a execução do sistema. Se o ambiente se modificar e, em particular, se as hipóteses na qual o agente se apóia se tornarem falsas durante a execução, o comportamento do agente pode ser indefinido. Mais ainda, assume-se que o objetivo permanece válido durante a execução. Se o objetivo não for mais válido, então simplesmente não há mais razão para continuar a execução do sistema (Woo99). Adicionalmente, tem-se que muitas das execuções em sistemas não-funcionais não terminam (Woo02).

Em muitos ambientes, nada do que foi assumido acima (as hipóteses e os objetivos do agente se manterem válidos durante a execução) é válido. Principalmente em domínios que são muito complexos para um agente observar completamente, que são multi-agentes ou onde há incerteza no ambiente,

essas hipóteses não são razoáveis. Em tais ambientes, executar cegamente um procedimento sem questionar as hipóteses nas quais o procedimento se baseia (pré-condições) não é uma boa estratégia. Em tais ambientes dinâmicos, um agente deve ser reativo. Ou seja, ele deve responder aos eventos que ocorrem no seu ambiente, onde esses eventos podem afetar tanto os objetivos do agente como as hipóteses nas quais os procedimentos que o agente está executando se baseiam para atingir os seus objetivos (Woo99).

Também não é difícil construir um agente puramente reativo, assim como construir agentes puramente direcionados a objetivos, que são aqueles que continuamente respondem aos seus ambientes. Entretanto, é difícil construir um sistema que atinja um balanço entre o comportamento direcionado a objetivos e o reativo. Deseja-se agentes que tentarão atingir seus objetivos sistematicamente, talvez fazendo uso de padrões de ações semelhantes a procedimentos complexos em linguagens de programação. Mas não se deseja que os agentes continuem executando esses procedimentos cegamente em uma tentativa de atingir um objetivo quando está claro que o procedimento não irá funcionar ou quando o objetivo deixa de ser válido por algum motivo (o que, na maioria dos casos, não é fácil de ser detectado por ser um processo indecidível). Em tais circunstâncias, deseja-se que o agente seja capaz de reagir à nova situação em um tempo tal de forma que a sua reação seja útil. Entretanto, não se quer que o agente esteja reagindo continuamente e, portanto, nunca focando em um objetivo por tempo suficiente para conseguir atingi-lo (Woo99).

Não é surpreendente o fato de ser difícil construir um sistema bem balanceado entre o direcionado a objetivos e o reativo, já que com pessoas isso também acontece (Woo99).

Considera-se agora a habilidade social. Aparentemente isso é trivial: diariamente, milhões de computadores em todo o mundo trocam informações tanto com seres humanos como com outros computadores rotineiramente. Todavia, a habilidade de se trocar seqüências de bits não pode ser considerada de fato habilidade social. O significado de habilidade social inclui a capacidade de negociação e cooperação entre os agentes. E para atingir certos objetivos, os agentes precisam negociar e cooperar uns com os outros (Woo99).

A definição de agente inteligente acima pode ter sua noção incrementada levando-se em conta atributos humanos. Muitos agentes são conceitualizados ou implementados usando conceitos que normalmente são mais aplicados a seres humanos. Por exemplo, pode-se caracterizar um agente utilizando-se noções mentalísticas, tais como conhecimento, crença, intenção e obrigação (Woo95, Sho93). Neste trabalho, como já ficou claro no capítulo anterior, são usados os conceitos de crença, desejo, intenção (modelo BDI (Woo99,

Woo00, Woo95)) e confiança. Também pode-se ter agentes emocionais, nos quais é possível se modelar emoções humanas em agentes computacionais (Woo95, Bat92a, Bat94).

Finalizando, agentes inteligentes podem ainda possuir outros atributos tais como (Woo95):

- Mobilidade
  É a habilidade de um agente de se mover em uma rede (Whi94).
- Veracidade
  É a hipótese de que um agente não irá enviar informações que ele sabe que são falsas (Gal88).
- Benevolência
  É a hipótese de que os agentes não têm objetivos conflitantes e que todo agente irá, portanto, sempre tentar fazer o que lhe for pedido (Ros85).
- Racionalidade
  É a hipótese de que um agente irá agir de forma a atingir os seus objetivos e que não irá agir de maneira tal que impeça seus objetivos de serem alcançados, pelo menos enquanto as suas crenças permitirem (Gal88).
- Aprendizado
  Para um sistema de agentes ser realmente inteligente, ele deve aprender a medida que ele reage ao ou interage com o seu ambiente. Afinal, considera-se que um atributo chave de qualquer ser que seja inteligente é a sua habilidade de aprender (Nwa96).

2.4 Arquiteturas para Agentes Inteligentes

2.4.1 Arquiteturas Abstratas

Para se formalizar uma visão abstrata de agentes, primeiramente deve-se assumir que o estado do ambiente do agente pode ser caracterizado como um conjunto $S = \{s_1, s_2, ...\}$ de *estados de ambiente*. Em um dado instante, assume-se que o ambiente está em algum desses estados. Assume-se também que a capacidade efetiva de um agente é representada por um conjunto $A = \{a_1, a_2, ...\}$ de *ações*. Portanto, um agente pode ser visto como uma função abstrata (Woo99):

$$acao : S^* \rightarrow A$$

que mapeia seqüências de estados de ambientes em ações. Essa função é determinística porque o agente, por ser computacional, responderá de apenas

a uma maneira a um determinado estímulo do seu ambiente. Um agente modelado por uma função dessa forma será considerado um agente padrão. A intuição é que um agente decida que ação fazer com base em sua história, sua experiência até então. Essas experiências são representadas como uma seqüência de estados de ambiente, aqueles que o agente encontrou até então (Woo99).

O comportamento (não-determinístico) de um ambiente pode ser modelado como uma função:

$$amb : S \times A \rightarrow \mathcal{P}(S)$$

que recebe o estado atual do ambiente $s \in S$ e uma ação $a \in A$ (executada por um agente), e as mapeia em um conjunto de estados de ambientes $amb(s, a)$ possivelmente resultantes da execução da ação a no estado s ($\mathcal{P}(S)$ representa o conjunto das partes do conjunto S). Se todos os conjuntos no contra-domínio de $amb()$ forem unitários, então o ambiente é determinístico e seu comportamento pode ser previsto de maneira precisa (Woo99). Nesse caso, a função $amb()$ pode ser descrita sem usar o conjunto das partes:

$$amb : S \times A \rightarrow S$$

Pode-se representar a interação entre agente e ambiente em uma *história*. Uma história h é uma seqüência;

$$h : s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \xrightarrow{a_{u-1}} s_u \xrightarrow{a_u} ...$$

onde s_0 é o estado inicial do ambiente (quando o agente começa a sua execução), a_u é a u-ésima ação que o agente escolheu executar e s_u é o u-ésimo estado do ambiente (o qual é um dos resultados possíveis da execução da ação a_{u-1} no estado s_{u-1}). Se $acao()$ for um agente; $amb()$, um ambiente, e s_0, o estado inicial do ambiente, então a seqüência h irá representar uma história possível do agente no ambiente se as seguintes condições valerem:

$\forall u \in \mathbb{N}$, $a_u = acao((s_0, s_1, ..., s_u))$ e $\forall u \in \mathbb{N}$, tal que $u > 0$, $s_u \in amb(s_{u-1}, a_{u-1})$.

O *comportamento característico* de um agente $acao()$ em um ambiente $amb()$ é o conjunto de todas as histórias que satisfazem essas propriedades. Se alguma propriedade ϕ satisfizer todas essas histórias, então essa propriedade pode ser considerada como uma propriedade invariante do agente no ambiente. Seja a função

$$hist : A_g \times A_b \rightarrow \mathcal{P}(H),$$

onde A_g representa um conjunto de agentes, A_b um conjunto de ambientes e H um conjunto de históricos. Então, $hist(agente, ambiente)$ denotará o conjunto de todas as histórias do *agente* no *ambiente*. Dois agentes ag_1 e ag_2 são ditos possuidores de *comportamento equivalente* com respeito ao ambiente *amb* se e somente se:

$$hist(ag_1, amb) = hist(ag_2, amb),$$

e simplesmente possuidores de comportamento equivalente se e somente se eles possuírem comportamento equivalente com respeito a todos os possíveis ambientes:

$$\forall amb\ hist(ag_1, amb) = hist(ag_2, amb).$$

Em geral, está-se interessado em agentes cujas interações com seus ambientes não tenham fim, ou seja, são infinitas. Em tais casos, as histórias são infinitas (Woo99).

Ver os agentes nesse nível abstrato contribui para uma análise simples dos mesmos. Todavia, não ajuda a construi-los, pois não dá nenhuma informação sobre a construção da função de decisão *acao*(). Por essa razão, o modelo abstrato de agentes é refinado, quebrando-o em sub-sistemas. Enquanto o modelo de agentes é refinado, faz-se escolhas para a construção que se relacionam com os sub-sistemas que compõem um agente (quais estruturas de dados e de controle estarão presentes). Uma *arquitetura de agente* é, essencialmente, um mapa das entranhas de um agente (suas estruturas de dados, as operações que podem ser feitas com elas e o fluxo de controle entre as mesmas).

Nas seções a seguir são descritas algumas arquiteturas abstratas para agentes inteligentes. Primeiramente, descreve-se os agentes puramente reativos. Posteriormente, descreve-se agentes com percepção e os com estados, os quais possuem decisões de *design* em alto-nível (Woo99).

Agentes Puramente Reativos

Certos tipos de agentes decidem o que fazer sem levar em conta as suas histórias. Eles baseiam as suas tomadas de decisão inteiramente no presente, com absolutamente nenhuma referência ao passado. Esses agentes são chamados de *puramente reativos*, já que eles simplesmente respondem diretamente aos seus ambientes. Formalmente, o comportamento de um agente puramente reativo pode ser representado por uma função:

$$acao : S \rightarrow A.$$

É fácil notar que para todo agente puramente reativo há um agente equivalente padrão; entretanto, o contrário não necessariamente é verdade. Ou seja, um agente reativo é um caso particular de um agente padrão.

Agentes com Percepção

Nesta sub-seção separa-se a função de decisão de um agente nos sub-sistemas de *ação* e de *percepção*, como mostrado na figura 2.2, onde novamente as setas representam a interação do agente com o ambiente. A seta que entra em *visao* representa a captura do estado do ambiente (entrada), e a seta que sai em *acao* representa a ação do agente no ambiente (saída). Resumindo, o agente executa uma ação em resposta a um estímulo do ambiente.

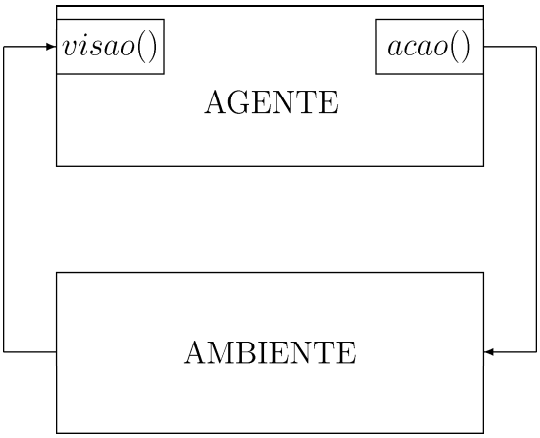

Figura 2.2: Sub-sistemas de ação e percepção

A idéia é a de que a função *visao*() captura a habilidade do agente de observar o seu ambiente, enquanto a função *acao*() representa o processo de tomada de decisão do agente. A saída da função *visao*() é uma *percepção*, que é o que o agente capta do ambiente através da sua visão. Seja P um conjunto não vazio de percepções. Então *visao*() é uma função:

$$visao : S \rightarrow P$$

que mapeia estados de ambiente em percepções. E *acao*() é agora uma função:

$$acao : P^* \rightarrow A$$

que mapeia seqüências de percepções em ações.

Essas simples definições permitem que algumas propriedades interessantes de agentes e percepções sejam exploradas. Sejam dois estados de ambiente, $s_1 \in S$ e $s_2 \in S$, tal que $s_1 \neq s_2$, mas que $visao(s_1) = visao(s_2)$. Ou seja, dois estados de ambiente diferentes são mapeados na mesma percepção e, portanto, o agente tem a mesma percepção a partir de estados de ambiente diferentes. Logo, na visão do agente, s_1 e s_2 são *indistinguíveis*.

Para o agente, isso significa que $s_1 \equiv s_2$. Não é difícil ver que $\equiv$ é uma relação de equivalência entre os estados de ambiente, que particiona S

em conjuntos de estados mutuamente indistinguíveis. Intuitivamente, quanto maiores essas classes de equivalência forem, menos efetiva é a percepção do agente. Se $|\equiv| = |S|$, ou seja, se o número de percepções distintas for igual ao de estados de ambiente diferentes, o agente pode distinguir todos os estados, o agente percebe perfeitamente o seu ambiente, sendo onisciente. Por outro lado, se $|\equiv| = 1$, a habilidade de percepção do agente é nula, ou seja, ele não pode distinguir nenhum estado diferente. Nesse caso, na visão do agente, todos os estados de ambiente são iguais.

Agentes com Estados

Até agora a função de decisão *acao*() foi modelada como mapeando seqüências de estados de ambiente ou de percepções em ações. Isso permite a representação de agentes cujos processos de tomada de decisão é influenciado pelo histórico. Porém, essa não é uma representação muito intuitiva, e, portanto, será substituída por um esquema equivalente, que é mais natural. A idéia é a de que os agentes mantenham estados, como mostrado na figura 2.3.

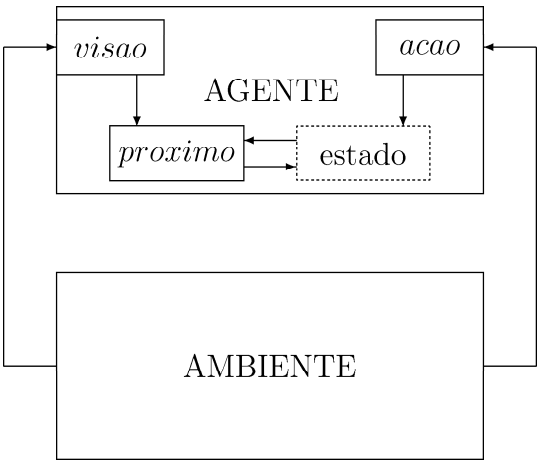

Figura 2.3: Agentes que contêm estados

Esses agentes têm alguma estrutura de dados interna, que é tipicamente usada para armazenar informações sobre o estado de ambiente e o histórico. Seja I o conjunto de todos os estados internos do agente. O processo de decisão de um agente é então baseado, pelo menos em parte, nessa informação. A função de percepção *visao*() para um agente com estados, mapeando estados de ambiente a percepções, não precisa ser modificada. Mas considerando-se que a maneira com a qual o agente percebe o seu ambiente pode ser afetada pelo estado interno do agente, tem-se que:

$$visao : S \times I \rightarrow P.$$

Já a função de seleção de ações $acao()$ agora é definida como sendo de estados internos para ações:

$$acao : I \rightarrow A.$$

Uma função adicional chamada $proximo()$ é introduzida, que mapeia um estado interno e uma percepção em um estado interno, ou seja, calcula o novo estado interno a partir do estado interno atual e da percepção capturada do ambiente:

$$proximo : I \times P \rightarrow I.$$

O comportamento de um agente com estados pode ser resumido da maneira a seguir:

- O agente começa a executar em algum estado inicial i_0;
- Ele observa o estado de seu ambiente s_0 e gera uma percepção $visao(s_0)$;
- O estado interno do agente é atualizado através da função $proximo()$, passando a ser $proximo(i_0, visao(s_0))$;
- A ação escolhida pelo agente é $acao(proximo(i_0, visao(s_0)))$;
- Essa ação é executada; e
- O agente entra em um novo ciclo, percebendo o ambiente através de $visao()$, atualizando o seu estado via $proximo()$ e escolhendo uma ação a ser executada via $acao()$.

É importante observar que os agentes com estados como os definidos acima não são mais poderosos que os agentes padrões introduzidos na seção 2.4.1. Na verdade, o poder de expressão deles é idêntico, ou seja, cada agente com estados pode ser transformado em um agente padrão com comportamento equivalente.

2.4.2 Arquiteturas Concretas

Até agora, os agentes só foram analisados de maneira abstrata. Portanto, por enquanto, foram examinadas as propriedades de agentes que contém ou não estados; não se considerando como são esses estados (suas estruturas). De maneira similar, a tomada de decisão de um agente foi modelada como uma função abstrata *acao*(), que de alguma forma consegue indicar que ação executar, mas não se discutiu como essa função pode ser implementada. Nesta seção isso será abordado em quatro classes de agentes (Woo99):

- Agentes lógicos
  A tomada de decisão é feita através de dedução lógica;
- Agentes reativos
  A tomada de decisão é implementada em alguma forma de mapeamento direto da situação para a ação;
- Agentes com crenças, desejos e intenções (BDI)
  A tomada de decisão depende da manipulação de estruturas de dados representando as crenças, os desejos e as intenções do agente, e
- Agentes com camadas
  A tomada de decisão é feita via várias camadas de *software*, cada qual estando em maior ou menor grau analisando o ambiente em diferentes níveis de abstração.

Em cada um desses casos, deixa-se a visão abstrata de agentes e começa-se a se descrever a estrutura interna e a operação dos agentes. Em cada subseção a seguir são explicadas a natureza desses agentes, as hipóteses nas quais cada arquitetura depende e as vantagens e desvantagens relativas de cada uma (Woo99).

Arquiteturas Baseadas em Lógica

A abordagem tradicional para a construção de sistemas inteligentes artificialmente, conhecida como IA simbólico, sugere que o comportamento inteligente pode ser gerado em um sistema através de uma representação simbólica do seu ambiente e de seu comportamento desejado, e manipulando sintaticamente essa representação. Nesta seção, foca-se nessa tradição, na qual essas representações simbólicas são fórmulas lógicas e a manipulação sintática corresponde à dedução lógica ou à prova de teoremas.

A idéia de agentes como provadores de teoremas é sedutora. Suponha-se alguma teoria de agentes que explique como um agente inteligente deve

se comportar. Essa teoria pode explicar, por exemplo, como um agente gera objetivos de forma a satisfazer os objetivos para os quais foi construído, como ele varia entre os comportamentos reativo e direcionado a objetivos de modo a atingir esses objetivos e assim por diante. Então, essa teoria ϕ pode ser considerada como sendo uma especificação de como o agente deve se comportar. A abordagem tradicional para implementar um sistema que satisfaça essa especificação envolveria o refinamento da mesma através de vários estágios, que cada vez a tornam mais concreta, até que a implementação finalmente seja feita. Todavia, quando se vê agentes como provadores de teoremas, tal refinamento não ocorre. Ao invés disso, ϕ é vista como uma especificação executável: ela é executada diretamente para produzir o comportamento do agente.

Para ver como tal idéia funciona, desenvolve-se um modelo simples de agentes lógicos, que serão chamados de agentes *deliberativos*. Em tais agentes, o estado interno é assumido como sendo um banco de dados de fórmulas da lógica clássica predicativa de primeira ordem (FOL - *First Order Logic*). Essas fórmulas podem ser usadas para representar as propriedades de um ambiente. Nesse caso, o banco de dados é a informação que o agente tem de seu ambiente. Logo, o banco de dados faz um papel semelhante ao de *crenças* em seres humanos. Assim como acontece com os homens, os agentes podem estar errados, ou seja, uma fórmula do banco de dados pode não ser verdadeira. A percepção ou o raciocínio do agente podem estar com problemas, a informação pode estar desatualizada ou a interpretação da fórmula desejada pelo criador do agente pode ser algo inteiramente diferente.

Seja L o conjunto de sentenças da lógica predicativa clássica de primeira ordem e seja $D = \mathcal{P}(L)$ o conjunto de bancos de dados de L, ou seja, o conjunto de conjuntos de fórmulas-L. O estado interno de um agente é então um elemento de D, onde $\Delta_0, \Delta_1, ... \in D$. O processo de tomada de decisão de um agente é modelado usando um conjunto de regras de dedução, ρ, as quais são simplesmente regras de dedução para a lógica de primeira ordem, neste caso. Escreve-se $\Delta \vdash_\rho \phi$ se a fórmula ϕ puder ser provada a partir do banco de dados Δ usando somente as regras de dedução ρ.

As funções *visao*(), *proximo*() e *acao*() continuam as mesmas, sendo que D toma o lugar de I. No caso de *acao*(), ela agora é definida em termos de suas regras de dedução. A definição algorítmica dessa função é mostrada na figura 2.4, onde $Executar(a)$ é um predicado de primeira ordem que significa a execução da ação a e *nada* é uma ação nula, semelhante a $NULL$ na linguagem de programação C e a *null* em Java.

A idéia é a de que o programador do agente irá codificar as regras de

```
1.    funcao acao(Δ : D) : A
2.    inicio
3.        para todo a ∈ A faca
4.            se Δ ⊢_ρ Executar(a) entao retorne a fim_se
5.        fim_para
6.        para todo a ∈ A faca
7.            se Δ ⊬_ρ ¬Executar(a) entao retorne a fim_se
8.        fim_para
9.        retorne nada
10.   fim_acao
```

Figura 2.4: Função *acao*() de um agente lógico

dedução ρ e um banco de dados Δ de tal forma que se a fórmula $Executar(a)$ puder ser derivada, onde a é um termo que denota uma ação, então a é a melhor ação a ser executada. Portanto, na primeira parte da função, (linhas (3)-(5)), o agente, para cada uma das possíveis ações a, tenta provar a fórmula $Executar(a)$ a partir do seu banco de dados (passado como um parâmetro para a função) usando as regras de dedução ρ. Se o agente tiver sucesso na prova de $Executar(a)$, a é retornada como a ação a ser executada.

Se o agente não conseguir provar $Executar(a)$ para todas as ações $a \in A$, ele tenta encontrar uma ação que seja consistente com suas regras de dedução e seu banco de dados, ou seja, uma que não seja explicitamente proibida. Portanto, nas linhas (6)-(8), o agente tenta achar uma ação $a \in A$ tal que $\neg Executar(a)$ não possa ser derivada a partir do banco de dados usando as regras de dedução. Isso é, se ele não conseguir provar que ele não pode executar a ação α, então ela não é proibida e, portanto, é consistente com as regras de dedução e o banco de dados. Se ele puder encontrar tal ação, então a mesma é retornada como sendo a ação a ser executada. Entretanto, se o agente não achar uma ação que seja ao menos consistente, ele não retorna ação nenhuma, indicando que nenhuma ação foi selecionada.

Portanto, o comportamento do agente é determinado pelas suas regras de dedução (seu programa) e seu banco de dados atual (representando as informações que o agente tem de seu ambiente). Ele primeiramente tenta encontrar a melhor ação a ser executada para um determinado estado do ambiente. Se ele não conseguir fazê-lo, ele tenta encontrar uma ação possível, não necessariamente a melhor. Se ele também não encontrar uma ação possível, ele não faz nada.

Examinando-se as pragmáticas da abordagem lógica de agentes, um ponto importante a se destacar é que uma construção literal, ingênua de agentes dessa maneira é absolutamente impraticável. Para ver o porquê disso, suponha que o conjunto de regras ρ tenha sido designado de tal forma que,

para qualquer banco de dados Δ, se for possível provar $Executar(a)$, então a é uma ação ótima, ou seja a é a melhor ação que poderia ser executada quando o ambiente estiver em um estado que é descrito por Δ. Então, ao começar a execução do agente, no tempo t_1, o agente gerou um banco de dados Δ_1 e começou a aplicar as suas regras ρ para encontrar a ação a ser executada. Algum tempo depois, em t_2, ele consegue estabelecer $\Delta_1 \vdash_\rho Executar(a)$ para algum $a \in A$ e, portanto, a é a melhor ação que o agente poderia executar em t_1. Porém, se o ambiente tiver se modificado durante esse intervalo, não há mais a garantia de que a ainda seja ótima. Ela pode ser muito aquém da ótima, particularmente se muito tempo tiver se passado entre t_1 e t_2. Se esse intervalo for infinitesimal, ou seja, se o processo de tomada de decisão for efetivamente instantâneo, então pode-se descartar esse problema. Todavia, sabe-se que que o problema de se saber se uma fórmula pode ser provada ou não é bem longe de ser resolvido instantaneamente, por ser um problema, no mínimo, NP-Completo (Mac78, Pap93, Cor90, Aho95). Para piorar, se o agente usar lógica clássica predicativa de primeira ordem para representar o seu ambiente, e suas regras forem completas e corretas, não há sequer garantias de que o processo terminará, por esse ser indecidível (Mac78, Pap93, End02, Car00). Considera-se que um agente possui a propriedade de racionalidade calculada se e somente se seu aparato de tomada de decisão sugerir uma ação que era ótima quando o processo de tomada de decisão começou. A racionalidade calculada é claramente inaceitável em ambientes que mudam mais rapidamente do que a velocidade de tomada de decisão do agente.

Pode-se dizer que esse problema é conseqüência da abordagem puramente lógica dada aqui. Isso é parcialmente verdadeiro. Afastando-se do rigor das linguagens de representação lógica e de conjuntos completos de regras de dedução, pode-se construir agentes com desempenho respeitável. Contudo, perde-se o que é argumentado como sendo a principal vantagem que a abordagem lógica fornece: uma semântica simples e elegante.

Há vários outros problemas associados à abordagem lógica de agentes. Primeiramente, considera-se a função *visao*(), que mapeia o ambiente em uma percepção. Em um agente baseado em lógica, a percepção geralmente é simbólica, sendo tipicamente um conjunto de fórmulas na linguagem de representação do agente. Entretanto, para muitos ambientes, não é óbvio como o mapeamento do ambiente para uma percepção simbólica pode ser feito. Isso é um problema se algo a ser percebido no ambiente for contínuo e não discreto, por exemplo. Outro problema é que a representação de propriedades de ambientes dinâmicos e do mundo-real é extremamente difícil. Por exemplo, representar e raciocinar sobre informações temporais (como uma situação

se modifica através do tempo) é extraordinariamente complicado. Por fim, representar informações procedurais simples (informações sobre "o que fazer") em lógica tradicional pode ser um tanto não-intuitivo e enfadonho.

Resumindo, em abordagens baseadas em lógica para a construção de agentes, a tomada de decisão é vista como uma dedução. O "programa" de um agente, ou seja, sua estratégia de tomada de decisão, é codificado como uma teoria lógica e o processo de seleção de uma ação se reduz ao problema da prova. Abordagens lógicas são elegantes e têm uma semântica lógica enxuta. Porém, têm muitas desvantagens. Em particular, a complexidade computacional inerente do processo de prova de teoremas torna questionável a operação efetiva de agentes como provadores de teoremas em ambientes com restrição de tempo. A tomada de decisão em tais agentes é baseada na hipótese da racionalidade calculada (se um agente tomasse uma decisão em um tempo infinitamente rápido ela seria perfeitamente racional) (Rus97) e de que uma ação que seja racional quando a tomada de decisão começa também será racional quando ela for concluída. O problema associado com a representação e o raciocínio em ambientes físicos, dinâmicos e complexos também estão essencialmente em aberto.

Arquiteturas Reativas

Por causa dos muitos problemas da abordagem lógica para agentes, outras abordagens surgiram baseadas nos seguintes temas:

- A idéia de que o comportamento inteligente e racional é visto como ligado de maneira inata ao ambiente que um agente ocupa, ou seja, o comportamento do agente não é isolado, mas sim um produto da interação que o agente mantém com o seu ambiente; e
- A idéia de que o comportamento inteligente emerge da interação de vários comportamentos simples.

As abordagens alternativas à lógica para agentes podem ser referidas como *comportamental* (já que é baseada na idéia do desenvolvimento e combinação de comportamentos individuais), *situada* (já que é baseada na idéia de que os agentes são situados em algum ambiente ao invés de serem isolados dele) e, finalmente, reativa (porque tais sistemas são geralmente vistos como simplesmente reagindo a um ambiente, sem raciocinar sobre isso). Esta seção apresenta uma discussão sobre a arquitetura de subsunção, a arquitetura reativa mais conhecida.

Há duas características que definem a arquitetura de subsunção. A primeira é que a tomada de decisão do agente é feita através de um conjunto de

comportamentos para a conclusão de tarefas; cada comportamento pode ser visto como uma função *acao*() individual, como definido acima, que continuamente faz uma análise sensorial e a mapeia em uma ação a ser executada. Cada um desses módulos de comportamento foi criado para cumprir uma determinada tarefa. Um ponto importante a se levantar é que se assume que esses módulos para o cumprimento de tarefas não incluem nenhuma representação simbólica complexa e não fazem fazem nenhum raciocínio simbólico. Em muitas implementações, esses comportamentos são implementados como regras da forma:

$$situacao \rightarrow acao$$

que simplesmente mapeia análises sensoriais diretamente em ações.

A segunda característica que define a arquitetura de subsunção é que vários comportamentos podem ser disparados ao mesmo tempo. Obviamente, deve haver um mecanismo para escolher entre as diferentes ações selecionadas por essas ações múltiplas. Uma proposta possível é arrumar os módulos em uma hierarquia de subsunção, com os comportamentos organizados em camadas. As camadas mais baixas na hierarquia são capazes de inibir camadas mais acima: quanto mais baixa uma camada é, maior é a sua prioridade. A idéia é que camadas mais acima representam comportamentos mais abstratos.

Assume-se que a função *visao*(), que representa a habilidade de percepção do agente, permanece a mesma. Entretanto, em sistemas implementados com a arquitetura de subsunção, assume-se uma ligação estreita entre ação e percepção, já que a entrada dada pela análise sensorial não é muito processada ou transformada.

A função de decisão *acao*() é composta de um conjunto de comportamentos, juntamente com uma relação de *inibição* entre esses comportamentos. Um comportamento é um par (c, a), onde $c \subseteq P$ é um conjunto de percepções chamado de *condições* e $a \in A$ é uma ação. Um comportamento (c, a) irá ser disparado quando o ambiente estiver no estado $s \in S$ se e somente se $visao(s) \in c$. $Comp = \{(c, a) \mid c \subseteq P \text{ e } a \in A\}$ é o conjunto de todas as regras desse tipo.

Uma relação binária de inibição está associada com o conjunto de regras de comportamento de um agente $R \subseteq Comp$: $\prec \subseteq R \times R$. Assume-se que essa relação é uma ordem total em R (transitiva, irreflexiva e anti-simétrica). Escreve-se $b_1 \prec b_2$ se $(b_1, b_2) \in \prec$ e lê-se "b_1 inibe b_2", ou seja, b_1 está mais abaixo na hierarquia do que b_2 e, portanto, terá prioridade sobre b_2. A função de ação é definida na figura 2.5.

Logo, a ação de seleção começa computando, em primeiro lugar, o conjunto *disparado* de todos os comportamentos que podem ser disparados (5).

1. $funcao\ acao(p : P) : A$
2. $var\ disparado : \mathcal{P}(\mathcal{R})$
3. $var\ selecionada : A$
4. $inicio$
5. $\quad disparado := \{(c, a) \mid (c, a) \in R\ e\ p \in c\}$
6. $\quad para\ todo\ (c, a) \in disparado\ faca$
7. $\quad\quad se\ \neg(\exists(c', a') \in disparado\ tal\ que\ (c', a') \prec (c, a))\ entao$
8. $\quad\quad\quad retorne\ a$
9. $\quad\quad fim\_se$
10. $\quad fim\_para$
11. $\quad retorne\ nada$
12. fim_acao

Figura 2.5: Função *acao*() de um agente reativo

Posteriormente, cada comportamento (c, a) que pode ser disparado é verificado para determinar se há algum outro comportamento de maior prioridade que possa ser disparado (8). Se nenhum comportamento puder ser disparado, então nenhuma ação é retornada, indicando que nenhuma ação foi escolhida.

Considerando-se que uma das principais preocupações com a tomada de decisão baseada em lógica era a sua complexidade teórica, deve-se examinar o quão bom é o desempenho desse simples sistema baseado em comportamento. A complexidade temporal da função de ação dessa arquitetura não é pior do que $\mathcal{O}(n^2)$, onde n é máximo entre o número de comportamentos e o de percepções. Portanto, mesmo com o algoritmo ingênuo acima a tomada de decisão é tratável. Na prática, é possível se ter resultados muito melhores: a lógica da tomada de decisão pode ser codificada em *hardware*, resultando em um tempo de decisão constante.

Há vantagens óbvias nas abordagens reativas, tal como na arquitetura de subsunção: simplicidade, economia, ser computacionalmente tratável, ser robusta contra falhas e elegante. Todos esses atributos tornam tais arquiteturas atraentes. Porém, há também problemas não resolvidos fundamentais, não somente na arquitetura de subsunção mas também em outras arquiteturas puramente reativas:

- Se os agentes não empregarem modelos de seus ambientes, eles precisam ter disponíveis informações suficientes sobre os seus ambientes locais para eles determinarem uma ação aceitável;
- Como os agentes puramente reativos tomam decisões baseadas em informações locais (ou seja, informações sobre o estado atual dos agentes), é difícil ver com tal tomada de decisão poderia levar em conta informações não-locais – inerentemente é preciso ter uma visão a curto prazo;

- É difícil ver como agentes puramente reativos podem ser construídos para aprender através da experiência, melhorando seus desempenhos ao longo do tempo;
- Uma grande vantagem de sistemas puramente reativos é que o comportamento geral emerge da interação dos comportamentos dos componentes quando o agente é posto no ambiente. Todavia, o termo "emerge" sugere uma relação entre os comportamentos individuais, o ambiente e o comportamento geral que não é capaz de ser entendida. Isso necessariamente torna muito difícil a construção de agentes para resolver tarefas específicas. Finalmente, não existe nenhuma metodologia para a construção de tais agentes: deve-se usar um processo trabalhoso de experimentações, tentativas e erros para construir um agente, e
- Enquanto agentes efetivos podem ser gerados com um número pequeno de comportamentos (em geral, com menos de dez camadas), é muito mais difícil construir agentes com muitas camadas. As dinâmicas das interações entre os diferentes comportamentos se tornam muito complexas para serem entendidas.

Várias soluções para esses problemas têm sido propostas. Uma das mais populares é a idéia de desenvolver agentes para realizar tarefas específicas.

Arquiteturas com Crenças, Desejos e Intenções (BDI)

Nesta seção, discute-se as arquiteturas com crenças (*beliefs*), desejos (*desires*) e intenções (*intentions*), que é usada como base neste trabalho. Essas arquiteturas têm suas raízes na tradição filosófica do entendimento do raciocínio prático – o processo de decidir, a cada momento, qual ação realizar de acordo com os objetivos.

Intuitivamente, as crenças de um agente correspondem às informações que o agente tem sobre o ambiente, as quais podem estar incompletas ou incorretas dependendo da capacidade de o agente perceber o mundo. Os desejos representam aquilo que o agente gostaria de fazer em um mundo ideal. Sistemas BDI implementados requerem que os desejos sejam consistentes entre si. Ou seja, dado um conjunto de desejos, não é possível derivar uma contradição a partir deles. Finalmente, as intenções representam desejos com os quais o agente se comprometeu a atingir.

O raciocínio prático envolve dois processos importantes: decidir que objetivos se quer alcançar e como se atingir esses objetivos. O primeiro processo é conhecido como deliberação e o último como raciocínio *means-ends*. Em outras palavras, primeiramente toma-se uma decisão, onde esse

processo tipicamente começa com a tentativa de se entender que opções estão disponíveis. Depois de gerar o conjunto de alternativas, deve-se escolher umas e se dedicar às mesmas. As opções escolhidas se tornam intenções, que então determinam as ações do agente. As intenções, por sua vez, retroalimentam o futuro raciocínio prático do agente na forma de novos desejos.

As intenções têm um papel crucial no processo do raciocínio prático. Uma das mais óbvias propriedades das intenções é que elas tendem a levar à ação, ou seja, se existir uma intenção de atingir um objetivo, deve-se lutar para alcançá-lo de maneira racional. Com isso, quer-se dizer que se deve tomar uma ação que se acredita que será a melhor maneira de satisfazer a intenção. Mais ainda, se a ação em andamento falhar em satisfazer a intenção, espera-se que se tente novamente outra ação e não simplesmente abandone a intenção.

Adicionalmente, o fato de se ter uma intenção restringe o raciocínio prático futuro. Por exemplo, enquanto se tem uma intenção, não se deve fazer opções que são inconsistentes com a intenção. Tem-se ainda que as intenções são persistentes. Se as intenções forem abandonadas sem se devotar recursos para atingi-las, nunca se atingirá nada. Contudo, não se deve persistir em uma intenção por tempo demais. De modo semelhante, se a razão para a intenção deixar de existir, então é racional desistir da intenção. Finalmente, as intenções estão intimamente ligadas com as crenças sobre o futuro. Ou seja, não se deve ter uma intenção se não se acredita que elas sejam alcançáveis. Portanto, nada impede também que uma crença seja sobre o futuro, algo que o agente acredita que seja verdade no futuro. Por exemplo, um agente, ao perceber o seu ambiente, pode concluir que no futuro determinado fato será verdade. Como ilustração, lidando com seres humanos, durante a noite, um homem tem a crença de que no futuro próximo o Sol vai voltar a nascer.

Dessa discussão, pode-se ver que as intenções possuem vários papéis importantes no raciocínio prático:

- As intenções direcionam o raciocínio *means-ends*;
- As intenções restringem as deliberações futuras;
- As intenções são persistentes; e
- As intenções influenciam as crenças nas quais o raciocínio prático futuro é baseado.

Um problema chave na criação de agentes com raciocínio prático é achar um balanço entre esses diferentes pontos. Especificamente, parece claro que um agente deve abandonar algumas intenções (porque ele acredita que elas nunca serão alcançadas ou já foram alcançadas, ou a razão para ter a intenção não existe mais). Logo, de tempos em tempos, vale a pena ao

agente parar para reconsiderar as suas intenções. Porém a reconsideração tem um custo em termos de tempo e de recursos computacionais. Portanto, isso apresenta o seguinte dilema, que é, essencialmente, o problema de balancear os comportamentos pró-ativo (direcionado a objetivos) e reativo (direcionado por eventos):

- Um agente que não pára para reconsiderar o suficiente freqüentemente continuará tentando atingir as suas intenções mesmo quando estiver claro que elas não podem ser atingidas ou quando não há mais razão para atingi-las, e
- Um agente que constantemente reconsidera as suas intenções pode gastar tempo insuficiente na tentativa de atingi-las e, portanto, corre o risco de nunca atingi-las.

Claramente há um balanço a ser atingido entre o grau de comprometimento e o de reconsideração. A natureza desse balanço foi estudada de maneira empírica em (Kin91). Seus autores investigaram como agentes *teimosos* (que nunca param para reconsiderar) e agentes *cautelosos* (que estão constantemente parando para reconsiderar) se comportam em vários tipos de ambientes. O parâmetro mais importante nesses experimentos foi a taxa de mudança do mundo, γ. Os principais resultados de (Kin91) foram:

- Se γ for baixo (o ambiente não se modifica rapidamente), os agentes teimosos se comportaram melhor comparados aos cautelosos, porque os últimos perdem tempo reconsiderando seus comprometimentos enquanto os agentes teimosos estão ocupados trabalhando em direção (e atingindo) os seus objetivos, e
- Se γ for alto (o ambiente se modifica rapidamente), os agentes cautelosos tendem a ter um desempenho melhor do que os agentes teimosos, porque eles são capazes de reconhecer quando as intenções não são mais atingíveis, e também quando levam vantagem em situações com novidades boas inesperadas e novas oportunidades.

Com isso, tem-se que diferentes tipos de ambientes requerem diferentes tipos de estratégias de decisão. Em ambientes estáticos que não se modificam, o comportamento puramente pró-ativo, direcionado a objetivos, é adequado. Todavia, em ambientes mais dinâmicos, a habilidade de se reagir às mudanças modificando as intenções se torna mais importante.

O processo de raciocínio prático em um agente BDI é resumido na figura 2.6. Como esta figura ilustra, há sete componentes principais em um agente BDI:

- Um conjunto de *crenças* atuais, representando as informações que o agente tem de seu ambiente atual;
- Uma *função de revisão de crenças* ($frcr()$), que recebe uma análise sensorial e as crenças atuais do agente, e, com base nelas, determina o novo conjunto de crenças;
- Uma *função de geração de opções* ($opcoes()$), que determina as opções disponíveis para o agente (seus desejos), com base nas suas crenças sobre o ambiente e suas intenções atuais;
- Um conjunto de *opções atuais* (desejos), representando as possíveis ações disponíveis ao agente;
- Uma função de *filtro* ($filtro()$), que representa o processo de deliberação do agente e que determina as intenções do agente baseando-se nas crenças, nos desejos e nas intenções atuais;
- Um conjunto de *intenções* atuais, representando o foco atual do agente, metas com as quais o agente se comprometeu em atingir, e
- Uma *função de seleção de ação* ($execucao()$), que determina uma ação a ser realizada com base nas intenções atuais.

A definição formal desses componentes é direta. Primeiramente, seja Bel o conjunto de todas as crenças possíveis; Des, o conjunto de todos os desejos possíveis, e Int, o conjunto de todas as intenções possíveis. Geralmente, as crenças, os desejos e as intenções são representados por fórmulas lógicas, possivelmente em lógica de primeira ordem. Portanto, pode-se afirmar que um agente BDI não deixa de ser um agente lógico. Quaisquer que sejam os conteúdos desses conjuntos, é importante notar que eles devem ter alguma noção de consistência definida sobre eles, de forma que alguém possa responder, por exemplo, a questão: ter uma intenção de atingir x é consistente com a crença em y? A representação de crenças, desejos e intenções como fórmulas lógicas permite que se formulem essas questões como questões sobre a determinação da consistência de uma fórmula lógica, um problema bem conhecido e bem entendido, embora seja indecidível no caso geral. O estado de um agente BDI em um dado momento é, obviamente, uma tripla (B, D, I), onde $B \subseteq Bel$, $D \subseteq Des$ e $I \subseteq Int$.

A função de revisão de crenças de um agente é um mapeamento:

$$frcr : \mathcal{P}(Bel) \times P \rightarrow \mathcal{P}(Bel)$$

que com base na percepção e nas crenças atuais determina um novo conjunto de crenças. Deve-se enfatizar que se o agente perceber dois estados de ambiente

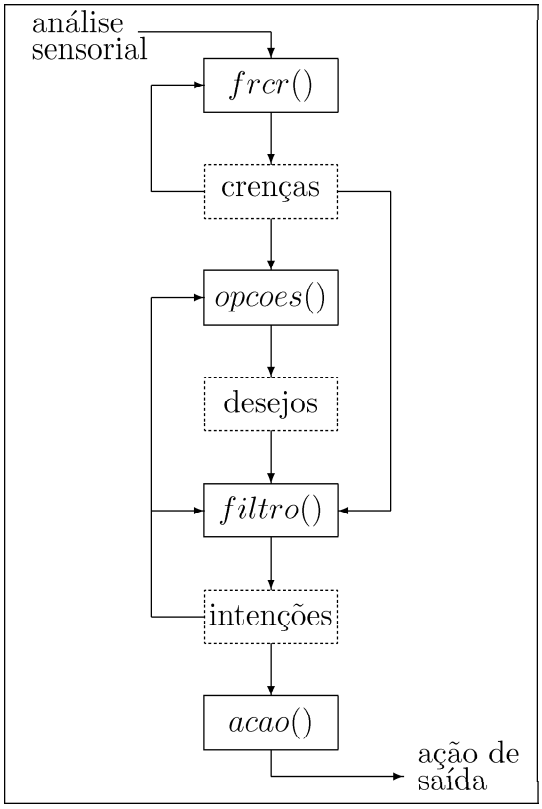

Figura 2.6: Diagrama esquemático de uma arquitetura de crenças, desejos e intenções genérica

diferentes como se fossem iguais, suas crenças com relação ao seu ambiente serão idênticas nas duas situações.

A função de geração de opções *opcoes*() mapeia um conjunto de crenças e um conjunto de intenções a um conjunto de desejos:

$$opcoes : \mathcal{P}(Bel) \times \mathcal{P}(Int) \rightarrow \mathcal{P}(Des)$$

Essa função exerce vários papéis. Primeiramente, ela deve ser responsável pelo raciocínio *means-ends* do agente, o processo de decidir como se atingir as intenções. Logo, assim que um agente formular uma intenção de atingir x, ele deve posteriormente considerar as opções para atingir x. Essas opções serão mais concretas (menos abstratas) do que x. Como algumas dessas opções (desejos) se tornam intenções posteriormente, elas irão retro-alimentar a geração de opções, resultando em opções ainda mais concretas sendo geradas. Portanto, pode-se ver o processo de geração de opções de um agente BDI como um processo de elaboração recursiva de uma estrutura de plano hierárquica, considerando e se comprometendo com intenções cada vez mais específicas, até finalmente atingir as intenções que correspondem a ações executáveis imediatamente.

Enquanto o propósito principal da função *opcoes*() é o raciocínio *means-ends*, ela deve, adicionalmente, satisfazer várias outras restrições. Em primeiro lugar, ela deve ser consistente: quaisquer opções geradas devem ser consistentes tanto com as crenças quanto com as intenções atuais. Em segundo lugar, ela deve ser oportunista, de forma que possa reconhecer quando as circunstâncias ambientais mudam de maneira vantajosa para o agente, para oferecer-lhe novas maneiras de se atingir as intenções, ou a possibilidade de atingir intenções que antes não eram atingíveis.

O processo de deliberação de um agente BDI (decisão do que fazer) é representado pela função de filtro:

$$filtro : \mathcal{P}(Bel) \times \mathcal{P}(Des) \times \mathcal{P}(Int) \rightarrow \mathcal{P}(Int)$$

que atualiza as intenções do agente baseada nas suas intenções previamente armazenadas, e nas suas crenças e nos seus desejos atuais. Essa função deve atender a dois papéis. Primeiramente, ela deve descartar quaisquer intenções que não forem mais atingíveis ou para as quais o custo de atingi-las excede o ganho esperado com o sucesso em atingi-las. Em segundo lugar, ela deve manter as intenções que ainda não foram atingidas e com as quais ainda se espera ter um benefício geral. Finalmente, ela deve adotar novas intenções, seja para atingir intenções existentes ou para explorar novas oportunidades.

Note-se que não se deve esperar que essa função introduza as intenções do nada. Portanto, $filtro()$ deve satisfazer a seguinte restrição:

$$\forall B \in \mathcal{P}(Bel), \forall D \in \mathcal{P}(Des), \forall I \in \mathcal{P}(Int)\ filtro(B, D, I) \subseteq I \cup D$$

Em outras palavras, as intenções atuais são intenções que já estavam armazenadas ou são novas opções adotadas.

Tem-se agora a função *execucao*(), na qual se assume que simplesmente retorne qualquer intenção executável, que é uma intenção que corresponde diretamente a uma ação executável:

$$execucao : \mathcal{P}(Int) \rightarrow A$$

A função de decisão de um agente BDI *acao*() é a seguinte:

$$acao : P \rightarrow A$$

Ela é definida pelo pseudo-código na figura 2.7.

Deve-se notar que a representação das intenções de um agente como um conjunto (ou seja, uma coleção não-estruturada) é muito simplista na prática. Uma alternativa simples é a associação de uma prioridade a cada intenção, indicando a sua importância relativa. Uma outra idéia natural é representar

```
1.  funcao acao(p : P) : A
2.  inicio
3.      B := frcr(B, P)
4.      D := opcoes(B, I)
5.      I := filtro(B, D, I)
6.      retorne execucao(I)
7.  fim_acao
```

Figura 2.7: Função *acao*() de um agente BDI

as intenções como uma pilha. Uma intenção é inserida (operação de *push*) na pilha quando é adotada e retirada (operação de *pop*) quando já foi atingida ou se tornar inatingível. As intenções mais abstratas tenderão a ficar no fundo da pilha e as mais concretas, no topo.

Finalizando, a cada ação executada, modifica-se o ambiente. O agente deve ser capaz de perceber corretamente a mudança que ele mesmo provocou e registrá-la em suas crenças, modificando-as adequadamente de forma que as crenças atuais mapeiem o ambiente modificado pelo próprio agente, para que suas crenças fiquem atualizadas com o estado atual do ambiente.

Resumindo, as arquiteturas BDI são arquiteturas de raciocínio prático, nas quais o processo de decisão do que fazer lembra o tipo de raciocínio prático que as pessoas aparentemente usam no cotidiano. Os componentes básicos de uma arquitetura BDI são as estruturas de dados que representam as crenças, os desejos e as intenções do agente, e funções que representam a sua deliberação (decidir que intenções ter, ou seja, decidir o que fazer) e o raciocínio *means-end* (decidir como fazer algo). As intenções fazem o papel central no modelo BDI: elas provêem estabilidade para o processo de decisão e agem para focar o raciocínio prático do agente. Um grande desafio nas arquiteturas BDI é o problema de se balancear reconsideração e comprometimento nas intenções: o processo de deliberação deve sofrer um ajuste fino para o seu ambiente, garantindo que em domínios dinâmicos e altamente imprevisíveis, ele reconsidere suas intenções freqüentemente; em ambientes mais estáticos, é necessário reconsiderações menos freqüentes.

O modelo BDI é atraente por várias razões. Primeiramente, ele é intuitivo: é facil se reconhecer os processos de decidir o que fazer e, posteriormente, como fazer o que foi decidido. Mais ainda, captura o entendimento informal das pessoas com relação às noções de crenças, desejos e intenções. Em segundo lugar, esse modelo fornece uma decomposição funcional clara, que indica que tipos de sub-sistemas são necessários para a construção de um agente. Porém, a maior dificuldade é, como sempre, saber implementar essas funções eficientemente.

Um agente BDI pode ser considerado um agente lógico, pois cada um dos seus estados mentais são usualmente formalizados em lógica modal. Também pode ser considerado um agente reativo, pois suas ações são baseadas de acordo com o que ele percebe no ambiente. Por fim, também pode ser considerado um agente com camadas como é mostrado na próxima sessão, pois cada componente pode ser considerado como uma camada.

Arquiteturas em Camadas

Considerando-se o requisito de que um agente deve ser capaz de ter comportamento reativo e pró-ativo, uma decomposição óbvia envolve a criação de sub-sistemas separados para lidar com esses diferentes tipos de comportamento. Essa idéia leva naturalmente a classe das arquiteturas nas quais os vários sub-sistemas são organizados em uma hierarquia de camadas interativas. Nesta sessão, serão considerados alguns aspectos dessas arquiteturas.

Tipicamente existirão pelo menos duas camadas para lidar com os comportamentos reativos e pró-ativos, respectivamente. Em princípio, não há razão para não se ter muito mais camadas. Qualquer que seja o número de camadas, uma tipologia útil para tais arquiteturas é através dos fluxos de informação e controle entre as camadas. Em termos gerais, pode-se identificar dois tipos de fluxo de controle nas arquiteturas em camadas (Mul95):

- Camadas dispostas horizontalmente:
  Nas arquiteturas com camadas dispostas horizontalmente (figura 2.8(a)), as camadas estão diretamente conectadas à análise sensorial e à ação de saída. De fato, cada camada em si age como um agente, produzindo sugestões para qual ação que deve ser realizada, e
- Camadas dispostas verticalmente:
  Nas arquiteturas com camadas dispostas verticalmente (figuras 2.8(b) e 2.8(c)), a análise sensorial e a ação de saída são lidados por no máximo uma camada em cada caso. Considerando um agente BDI como um agente em camadas, suas camadas seriam dispostas verticalmente.

A grande vantagem da disposição horizontal é a simplicidade conceitual da mesma: se for necessário um agente exibir n tipos diferentes de comportamento, então pode-se implementar n camadas. Entretanto, como as camadas estão na verdade competindo umas com as outras para gerar sugestões de ações, há o risco de o comportamento geral do agente não ser coerente. Para garantir que as arquiteturas com camadas dispostas horizontalmente sejam consistentes, elas geralmente possuem uma função *mediadora*, que decide qual camada tem "controle" do agente em um dado instante de tempo. A necessidade de

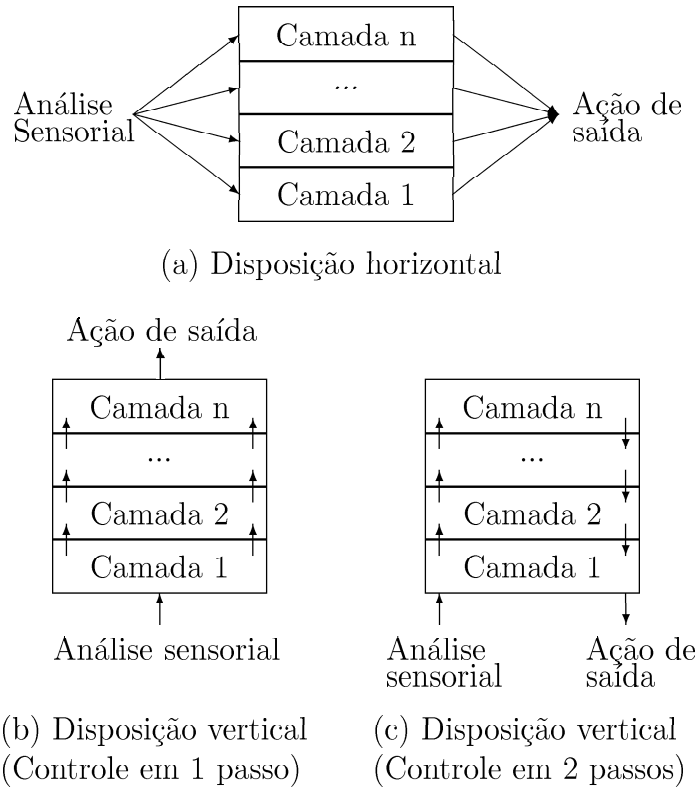

Figura 2.8: Fluxos de informação e controle em três tipos de arquiteturas de agentes em camadas

tal controle central é problemática, significando que, potencialmente, o criador do agente deve considerar todas as possíveis interações entre as camadas. Se houver n camadas em uma arquitetura e cada camada for capaz de sugerir m ações possíveis, m^n interações precisam ser consideradas. Isso é claramente difícil do ponto de vista de engenharia para qualquer sistema que não seja muito simples. A introdução de um sistema de controle central introduz um gargalo na tomada de decisão do agente.

Esses problemas são parcialmente resolvidos em uma arquitetura em camadas dispostas verticalmente. Pode-se subdividir as arquiteturas em camadas dispostas verticalmente em arquiteturas de um passo (figura 2.8(b)) e em arquiteturas de dois passos (figura 2.8(c)). Nas arquiteturas de um passo, o controle flui seqüencialmente através de cada camada, até que a camada final gere a ação de saída. Em arquiteturas de dois passos, a informação flui em direção ao topo da arquitetura (primeiro passo) e o controle posteriormente flui no caminho inverso. Em ambas as arquiteturas, a complexidade das interações entre as camadas é reduzida: como há $n-1$ interfaces entre n camadas, se cada camada for capaz de sugerir m ações, há no máximo $m^2(n-1)$ interações a serem consideradas entre as camadas. Isso claramente é mais simples que

o caso de camadas dispostas horizontalmente. Entretanto, com essa simplicidade há uma redução de flexibilidade: para que uma arquitetura em camadas dispostas verticalmente possa tomar uma decisão, o controle precisa passar por todas as camadas. Não há tolerância a falhas: uma falha em uma camada provavelmente resultará em sérios danos ao desempenho do agente.

A decomposição em camadas representa uma decomposição natural de funcionalidade: é fácil ver como os comportamentos reativo, pró-ativo e social podem ser gerados por camadas reativa, pró-ativa e social em uma arquitetura. O problema principal com as arquiteturas em camadas é que enquanto elas podem ser consideradas uma solução pragmática, elas não possuem uma semântica e um conceito claros presentes nas arquiteturas sem camadas. Em particular, enquanto as arquiteturas baseadas em lógica têm uma semântica lógica clara, é difícil ver como uma semântica desse tipo pode ser desenvolvida para uma arquitetura em camadas. Outro problema é o da interação entre camadas. Se cada camada for um processo de produção de ações independente, é necessário considerar todas as possíveis maneiras com as quais as camadas podem interagir umas com as outras. Esse problema é parcialmente resolvido com as arquiteturas em camadas dispostas verticalmente.

Por fim, tem-se que um agente BDI não deixa de ter uma arquitetura disposta em camadas, onde cada componente do agente tem uma função bem definida e interage com os outros passando-lhes as informações necessárias. Ou seja, há um fluxo de informações entre as crenças, os desejos e as intenções. Porém, sua arquitetura não é simplesmente vertical ou horizontal.

2.5 Aplicações para Agentes Inteligentes

Algumas áreas de aplicações para agentes inteligentes são (Woo99, Woo95):

- Quando um módulo espacial faz uma longa viagem da Terra para outros planetas mais distantes, um grupo na Terra em geral é necessário para acompanhar o progresso da missão continuamente e decidir como lidar com situações inesperadas. Isso é caro e, se as decisões precisarem ser rápidas, simplesmente impraticável. Por essas razões, as agências espaciais estão investigando seriamente na possibilidade de tornar os módulos mais autônomos, dando aos mesmos capacidades de tomada de decisão e responsabilidades maiores;
- Se um controle de tráfego aéreo falhar ou então um ser humano que está nesse controle falhar, um sistema autônomo em um outro controle aéreo próximo pode detectar a falha e corrigi-la a tempo, impedindo eventuais

acidentes com todos os vôos afetados. Com isso, uma potencial situação desastrosa acaba sem nenhum incidente (Woo02);

- Pesquisar na Internet pela resposta de uma busca específica pode ser um processo longo e entediante. Uma solução é ter um programa de computador (um agente) para fazer a busca. Tipicamente, seria dado ao agente uma busca que necessitaria da síntese de informações de várias fontes de informação na Internet;
- Agentes de interface, que são programas de computador que aplicam técnicas de inteligência artificial para prover assistência a um usuário que lida com uma aplicação em particular. A metáfora é a de um *assistente pessoal* que está colaborando com o usuário no mesmo ambiente de trabalho (Mae94); e
- Há um potencial óbvio para a união da tecnologia de agentes com computação gráfica e realidade virtual. O objetivo é desenvolver mundos simulados artisticamente interessantes e altamente interativos, para dar aos usuários a experiência de viver, e não somente ver, mundos ricos de emoções que incluam agentes emocionais razoavelmente competentes (Bat92b). Para construir tais mundos simulados, é necessário desenvolver agentes críveis; agentes que provêem a ilusão de vida (Bat94). Um componente chave em tais agentes é a emoção: eles precisam mostrar emoções, e agir e reagir de uma maneira que entrem em ressonância com a empatia e o entendimento do comportamento humano.