3
Agentes e objetos

A engenharia de software baseada em agentes [93, 127, 128, 164] é uma área emergente cujo objetivo é oferecer suporte ao desenvolvimento de sistemas multiagentes. No entanto, não se chegou ainda a um acordo em relação a um grupo comum de conceitos que pode usado em diferentes metodologias e arquiteturas de agentes. A idéia dos agentes de software e os conceitos relacionados é muito vaga dentro da comunidade de agentes [65]. Cada pesquisador possui suas próprias definições.

Como resultado, entender os concerns essenciais que impulsionam o projeto e o desenvolvimento dos sistemas multiagentes é um grande desafio [65]. É muito difícil caracterizar e comparar arquiteturas e métodos orientados a agentes e promover efetivamente a separação de concerns de agentes. Além disso, ainda não há uma compreensão clara da interação entre as noções de agentes e objetos a partir de uma perspectiva da engenharia de software [33, 56, 91, 248].

Nesse contexto, há três perguntas básicas apresentadas neste capítulo:

(i) quais são os concerns básicos no desenvolvimento de agentes de software?

(ii) quais são as definições operacionais para esses concerns?

(iii) até que ponto as abstrações orientadas a objetos e as arquiteturas de software existentes oferecem suporte à separação de concerns de agentes?

Nesse contexto, este capítulo apresenta o TAO[3][223], um framework conceitual que oferece as bases para a compreensão dos concerns essenciais no

[3] TAO significa “Taming Agents and Objects”.

desenvolvimento de agentes de software. Ele define os conceitos (concerns) que fornecem uma terminologia consistente e a semântica básica para pensar a implementação e o projeto dos agentes em termos dos concerns centrais que caracterizam o software baseado em agentes. O objetivo não é apresentar uma pesquisa sobre os conceitos existentes usados por diferentes metodologias e arquiteturas orientadas a agentes , mas sim apresentar a definição de um framework conceitual que, de acordo com nosso ponto de vista, inclui os conceitos essenciais para o suporte ao desenvolvimento de métodos e arquiteturas.

Nessa definição informal, o framework consiste em uma descrição em linguagem natural de categorias dos conceitos centrais e seus relacionamentos, assim como um conjunto de modelos de características conceituais [130]. O TAO pode ser usado para facilitar a compreensão da interação de agentes e de outras abstrações da engenharia de software (por exemplo, objetos), avaliando as arquiteturas orientadas a agentes existentes, além de ser usado para impulsionar o desenvolvimento de novos métodos orientados a agentes.

No contexto desta tese, o framework conceitual é usado na discussão das limitações do paradigma de objetos na estruturação de concerns de agentes (Seção 3.6), para avaliar arquiteturas de agentes (Seção 3.7) e, em especial, para oferecer suporte ao método arquitetural orientado a aspectos (Capítulo 0) e a linguagem de padrões (Capítulo 5). O TAO também foi desenvolvido com o objetivo de fornecer um metamodelo para a modelagem de SMAs. Contudo, essa característica está fora do contexto deste trabalho e não será explicada. Ele é usado no contexto de outros trabalhos de pesquisa [222] desenvolvidos em nosso laboratório.

As contribuições deste capítulo foram descritas em [223]. O restante deste capítulo está organizado da seguinte forma. A Seção 3.1 apresenta o framework conceitual e uma breve descrição dos concerns do agente e seus relacionamentos. A Seção 3.2 apresenta os concerns fundamentais. Uma definição dos concerns de agência e de outros concerns é apresentada respectivamente nas Seções 3.3 e 3.4. A Seção 3.5 discute os relacionamentos entre as diferentes categorias de concerns. A Seção 3.6 ressalta as limitações da orientação a objetos para o projeto e a implementação de SMAs. A Seção 3.7 apresenta as arquiteturas de agentes existentes

cujo objetivo é o suporte à separação de concerns do agente. Finalmente, a Seção 3.8 apresenta as conclusões do capítulo.

3.1
TAO: o framework conceitual

Os conceitos associados a agentes de software não possuem bases sólidas. Eles são tão difusos quanto o conceito de agente propriamente dito. Os defensores da computação baseada em agentes não conseguem oferecer uma descrição satisfatória que diferencie software de agentes e software que não seja de agentes. Não houve exatamente uma tentativa real de fazer uma definição objetiva e operacional dos conceitos relacionados a agentes [65]. Muitos problemas encontram-se na natureza lingüística desses conceitos, o que dificulta sua definição e mais ainda sua descrição mais precisa [65].

Por exemplo, notações diferentes da interação de agentes são capturadas por termos diferentes em diversos métodos e arquiteturas de agente, como “comunicação”, “cooperação”, “colaboração”, “percepção” etc. Além disso, esses conceitos são às vezes usados de forma intercambiável para o mesmo significado. Ocorre um problema similar com o conceito de autonomia. Para alguns pesquisadores, a autonomia é a capacidade do agente de ter suas próprias threads de controle [2, 119]. Outros defendem que a autonomia é a capacidade de tomar decisões [8, 180, 119]. Há ainda os que defendem que a autonomia é a capacidade proativa do agente [119, 128, 183]. Finalmente, alguns definem autonomia como a combinação de duas ou três características do agente [61, 119, 207, 247].

Entretanto, apesar de a idéia do software baseado em agentes ser muito vaga, mesmo dentro da comunidade de agentes, parece haver um consenso em relação às propriedades recorrentes e decisivas dos agentes de software [128, 183, 202, 248]. O principal propósito do TAO é fornecer um framework conceitual unificador para a compreensão de conceitos de agentes comuns (concerns) e seus relacionamentos. O uso consistente desses conceitos facilita o desenvolvimento e a comparação de métodos e arquiteturas orientados a agentes.

3.1.1 Estrutura

O framework define e conecta concerns importantes no domínio de SMAs. Esses concerns são a base do projeto e da implementação de SMAs. O TAO apresenta a definição de cada conceito e possibilita a criação de relacionamentos entre eles. Cada conceito do TAO representa um concern da engenharia de software (Capítulo 2), ou seja, uma característica de agente importante que os engenheiros de software precisam modularizar. Chamamos esses concerns de *concerns de agência* ou *concerns do agente.* O conjunto central de concerns de agência foi desenvolvido com base na investigação das metodologias orientadas a agentes existentes [66, 169, 244, 249, 251], frameworks e arquiteturas orientadas a agentes (Seção 3.7), linguagens de programação [142, 174, 216, 217] e metamodelos [70, 129]. O TAO é estruturado da seguinte forma:

- contém um conjunto restrito de concerns os quais recebem significados específicos;
- define cada concern em termos de subconceitos (ou subconcerns), que são, por sua vez, definidos usando uma terminologia consistente;
- define os relacionamentos entre concerns e subconcerns;
- modela os concerns e os subconcerns usando modelos de características [130].

O TAO usa modelos de características [130] (Tabela 2) porque, por um lado, são fáceis de entender e, por outro, permitem expressar relações relativamente complexas de uma maneira bem compacta. Além disso, eles facilitam a representação de concerns obrigatórios e opcionais no domínio de SMAs. Cada concern é uma característica nos modelos de características. Os concerns são apresentados em *itálico* ao longo do texto. Embora algumas definições do TAO possam parecer restritivas, nossa experiência permite a conclusão de que os concerns na engenharia de software baseada em agentes nunca serão claros a menos que sejam estreitados os conceitos desses agentes. O TAO não tem qualquer compromisso, de nenhuma forma, com o objetivo de completude. No entanto, ele fornece um primeiro passo em direção a uma melhor compreensão dos concerns de agentes.

3.1.2
Categorias de concerns

O framework define um total de quatorze concerns principais, resumidas na Tabela 3. O TAO classifica o conjunto de concerns de agentes em 3 categorias:

- *concerns fundamentais*: são a base para a construção de sistemas multiagentes, incluindo objetos, tipos de objetos, agentes, tipos de agentes, ambientes e eventos;
- *concerns de agência:* são os concerns obrigatórios na construção de uma arquitetura de agentes – incluindo o conhecimento intrínseco, a interação, a adaptação e a autonomia;
- *concerns adicionais* – são os concerns opcionais presentes em arquiteturas de agentes mais complexas – incorporam a colaboração, os papéis, a mobilidade e a aprendizagem.

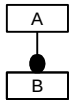
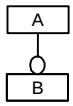
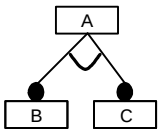
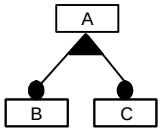
Feature Type	Graphical Representation
mandatory Mandatory feature B has to be included if its parent feature A is selected	
optional Optional feature B may be included if its parent feature A is selected.	
alternative Alternative features are organized in alternative groups. Exactly one feature of such the group B,C has to be selected if the group's parent feature A is selected.	
or *Or* features are organized in *or* groups. At least one feature of such the group B,C has to be selected if the group's parent feature A is selected.	

Tabela 2. Elementos do Diagrama de Características

3.2
Concerns fundamentais

Esta seção apresenta os concerns essenciais para criar um sistema multiagentes. Os concerns fundamentais incluem agentes, objetos, ambientes (Seção 3.2.1), eventos (Seção 3.2.2) e tipos de agentes (Seção 3.2.3).

3.2.1
Agentes, objetos e ambientes

Um *sistema multiagentes* (SMA) é composto por um conjunto de entidades. Essas entidades incorporam diferentes tipos de *agentes* e *objetos* que são inseridos em *ambientes* [127]. Um agente é visto como uma extensão de um objeto [91, 112, 183, 217]. Objetos e agentes fornecem serviços a seus clientes. No entanto, os objetos são entidades não-autônomas que representam elementos de sistemas passivos. Um

agente é uma entidade interativa, adaptativa e autônoma que age no ambiente e manipula objetos [33, 17, 183, 223]. Somente sistemas interativos, adaptativos e autônomos são agentes [119, 127, 183, 202].

	CONCERN DE AGÊNCIA	DEFINIÇÃO
Concerns fundamentais	**Ambiente**	A entidade que contém os objetos e os agentes de sistema
	Objeto	Uma entidade passiva que oferece serviços a outras entidades
	Tipo de objeto	Uma categoria específica de objeto no sistema
	Agente	Uma entidade interativa, adaptativa e autônoma que oferece serviços a outras entidades
	Tipo de agente	Uma categoria específica de agente no sistema
	Evento	Um evento é tudo que acontece para alterar o ambiente ou tudo aquilo com que o agente deve ser preocupar
Concerns de agência	**Conhecimento intrínseco**	A funcionalidade básica do agente (ou seja os serviços básicos) disponibilizada para outras entidades do ambiente
	Interação	Um agente se comunica com outras entidades por meio de sensores e efetores
	Adaptação	Um agente adapta/modifica seu conhecimento e comportamento de acordo com os eventos observados
	Autonomia	Um agente é capaz de agir sem intervenção externa direta; ele possui sua própria thread de controle, aceita ou recusa uma solicitação de serviço e age proativamente
Concerns adicionais	**Aprendizagem**	Um agente pode aprender com base em experiências anteriores enquanto interage com seu ambiente e colaborando com outros agentes
	Mobilidade	Um agente é capaz de se transportar de um ambiente em uma rede para outro
	Colaboração	Um agente pode cooperar com outros agentes a fim de alcançar seus objetivos e os objetivos do sistema
	Função	Agrega uma parte do conhecimento do agente (conhecimento extrínseco), que é necessária em contextos colaborativos específicos

Tabela 3. Uma visão geral de concerns do agente

Um agente é composto de conhecimento e de um conjunto de propriedades, chamadas *propriedades de agente* ou *propriedades de agência* (Figura 7). As propriedades de agente são características comportamentais que podem ser

incorporadas por um agente. São classificadas como *propriedades de agência* ou *propriedades adicionais* (Figura 7). Autonomia, interação e adaptação são consideradas propriedades de agência de agentes de software (Seção 3.3), enquanto a colaboração, os papéis, a aprendizagem e a mobilidade não são condições necessárias, tampouco suficientes, para a agência [183]. A Tabela 3 resume as definições para as propriedades adicionais e da agência.

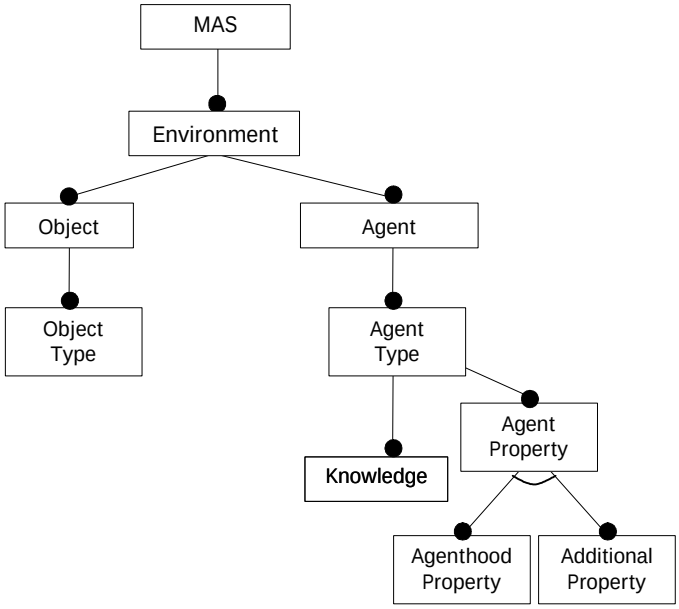


Figura 7. Ambiente, objetos e agentes.

3.2.2 Eventos

Os agentes sentem (ou percebem) *eventos* gerados por entidades do sistema. Um evento é tudo que acontece para alterar o ambiente ou tudo aquilo com que o agente deve ser preocupar. Um evento é *interno* ou *externo* a um agente (Figura 8). Alguns exemplos de eventos externos incluem a chegada de uma nova *mensagem* de outros agentes e as alterações nos objetos do ambiente. Os eventos internos incluem a execução de alguma ação de agente interno, a alteração de algum conhecimento, a definição de um novo objetivo de agente, a finalização de um plano de agente etc.

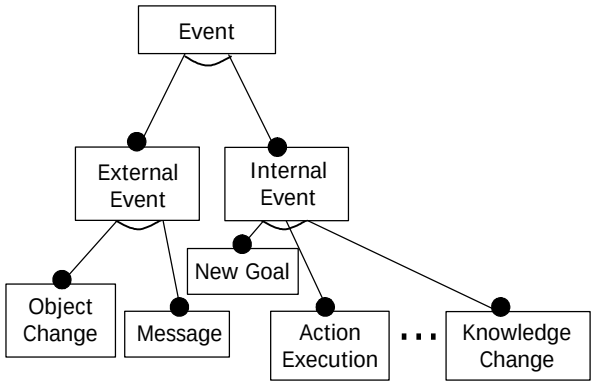


Figura 8. Eventos.

Os eventos são gerados por objetos e agentes e pelo ambiente propriamente dito. Eles disparam a execução de ações de agentes, que têm de descobrir a significância do evento. Por exemplo, eles disparam ações de interação (por exemplo, envio e recebimento de mensagens), ações autônomas (por exemplo definição de um novo objetivo) e ações de aprendizagem (por exemplo, a execução de um serviço).

3.2.3 Tipos de agente

Um agente de software normalmente não é encontrado sozinho em um sistema. Um SMA, em geral, possui vários tipos de agentes de software [27, 182]. As arquiteturas internas dos tipos de agente podem ser diferenciadas entre si uma vez que cada tipo de agente normalmente incorpora serviços distintos e diferentes propriedades de agência. Cada tipo de agente possui seu próprio conhecimento intrínseco e implementa os concerns de agência (Seção 3.3); no entanto, cada um incorpora um conjunto arbitrário de concerns adicionais (Seção 3.4). Os relacionamentos entre os concerns dependem do tipo de agente (Seção 3.5).

Como resultado, os tipos de agentes são classificados de acordo com as diferentes características que as distinguem entre si. Há diferentes maneiras de classificar tipos de agente. Eles são derivados, por exemplo, de um conjunto de serviços fornecido a seu ambiente e a outros tipos de agente (Figura 9). Os tipos de agente predominantes encontrados em quase todos os sistemas baseados em agentes são agentes de informação, agentes de usuário e agentes de interface.

Os *agentes de informação* estão firmemente acoplados a fontes de informações de forma a encontrar informações em resposta a buscas e monitorar ativamente as fontes em relação às condições especificadas [34, 182]. Eles ajudam a automatizar o processo de decisão em relação à utilidade da informação disponível. Os *agentes de usuário* representam entidades físicas (em geral, humanas) e atuam como assistentes pessoais para concluir as tarefas do usuário [24, 182]. Entre outras responsabilidades, eles armazenam as preferências e as informações do usuário. Os *agentes de interface* gerenciam as interfaces gráficas e adquirem, modelam e adaptam seu estado e comportamento de acordo com as preferências dos usuários usando a interface [164].

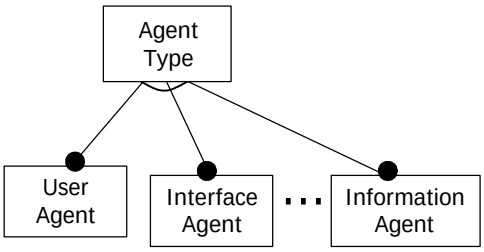


Figura 9. Tipos de agente: classificação baseada em serviços de agente.

Os tipos de agente também podem ser classificados com base em sua capacidade cognitiva [69] (Figura 10). Um *agente cognitivo* [69] reage no ambiente usando o conhecimento com base no histórico do agente e incorpora estratégias de adaptação sofisticadas (Seção 3.3.3) ou técnicas de aprendizagem automáticas (Seção 3.4.3). Um *agente reativo* [69] é um agente que reage no ambiente sem qualquer conhecimento prévio de sua história. Se dois eventos do mesmo tipo forem recebidos pelo agente reativo em momentos diferentes, será executado um comportamento idêntico. Os agentes reativos só têm autonomia de execução; eles não têm autonomia de decisão ou autonomia proativa (Seção 3.3.4). Um *agente híbrido* é um agente reativo e cognitivo [69].

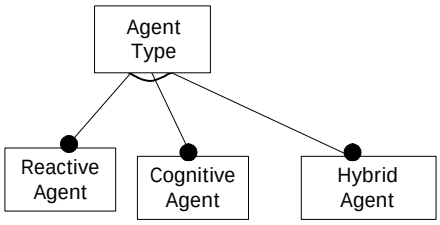


Figura 10. Classificação baseada em capacidades cognitivas.

Essas classificações de tipos de agente não são ortogonais. Por exemplo, um agente cognitivo pode ser um agente de informação ou um agente de interface. Além disso, um agente de usuário pode ser um agente reativo ou um agente cognitivo. Os tipos de agente listados anteriormente são formas predominantes encontradas na maioria dos sistemas baseados em agentes. Uma análise mais detalhada de uma aplicação pode identificar outras formas de agente, como agentes do facilitador, agentes intermediários, agentes do gerenciador etc. Para obter uma discussão mais completa sobre as diferentes classificações de tipos de agente, consulte [182, 183].

3.3
Concerns de agência

Os concerns de agência são características incorporadas por cada arquitetura de agente independente do tipo de agente. Apesar de não haver nenhuma definição de agência amplamente aceita, parece haver um consenso de que as propriedades necessárias de agência são interação, adaptação e autonomia. Esses conceitos são normalmente citados como as três propriedades características da agência [33, 100, 119, 127, 183, 202]. Essas propriedades de agente e o conhecimento intrínseco do agente incorporam os concerns de agência (Figura 11). Dessa forma, uma arquitetura de agente básica consiste na funcionalidade básica do agente (ou conhecimento intrínseco) e nos mecanismos para a interação com o ambiente a sua volta (Seção 3.3.2), adaptando seu conhecimento e comportamento de acordo com essas interações (Seção 3.3.3) e agindo de forma autônoma para alcançar os objetivos (Seção 3.3.4).

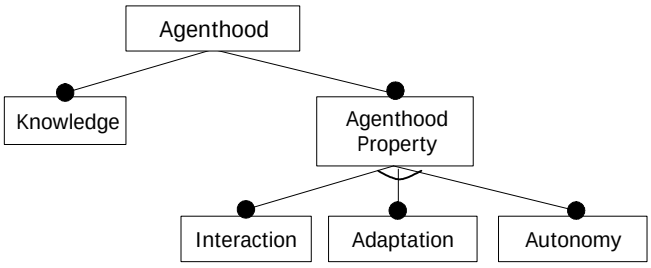


Figura 11. Concerns de agência.

3.3.1 Conhecimento

Um concern fundamental para os engenheiros de SMAs é a definição de *conhecimento* de seus agentes. O conhecimento é o fato ou a condição que percebe algo [22]. O conhecimento do agente consiste em: (i) informações sobre o agente propriamente dito, o ambiente do agente e outras entidades do ambiente (objetos e agentes) e (ii) o conjunto de serviços fornecidos pelo agente. O conhecimento do agente possui duas partes: conhecimento intrínseco e conhecimento extrínseco [136] (Figura 12). O *conhecimento intrínseco* é aquele relacionado à funcionalidade básica do agente. Ele incorpora as informações básicas e os serviços básicos fornecidos pelo agente [136]. O *conhecimento extrínseco* é aquele relacionado aos diferentes papéis exercidos pelo agente em diversos contextos colaborativos [136] (Seção 3.4.2). Ele incorpora as informações e os serviços necessários para que o agente exerça uma função específica.

O processo de tomar o conhecimento e colocá-lo em uma forma reconhecida pelo computador a fim de usá-lo para resolver problemas é chamado de *representação de conhecimento*. Há diferentes maneiras de representar o conhecimento de agente, e as abordagens mais populares são [22]: (i) representação procedimental, (ii) representação declarativa, (iii) representação relacional, (iv) representação hierárquica, (v) lógica de predicados, (vi) frames, (v) redes semânticas, (vi) representação da incerteza (rede bayesiana, rede de crença, rede causal ou rede probabilística) e (vii) KIF (Knowledge Interchange Format) [104].

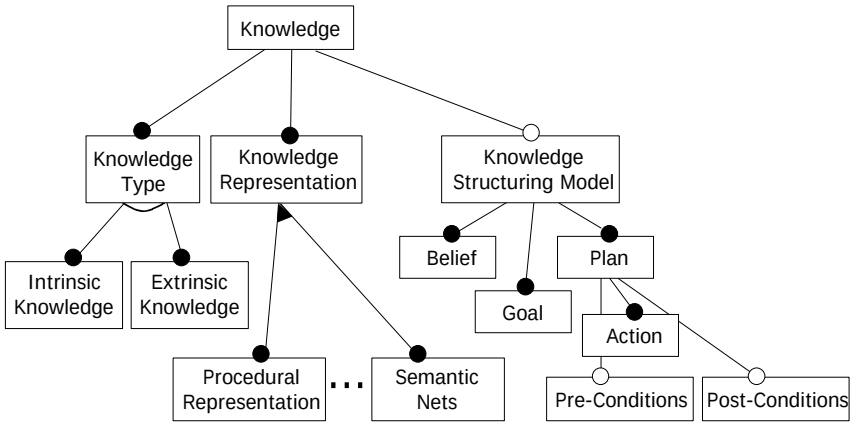


Figura 12. Concern de conhecimento.

A representação do conhecimento atraiu a atenção de muitos pesquisadores na área da inteligência artificial. A discussão sobre as vantagens e desvantagens do uso de um determinado tipo de representação de conhecimento está além do escopo deste trabalho. Há muitas referências interessantes sobre essas discussões [22, 172, 207]. Este trabalho se dedica à representação do conhecimento com base em abstrações orientadas a objetos, apesar de suas deficiências [22, 202, 207]. O paradigma de objetos pode ser usado para representar o conhecimento procedimental, as regras se-então, as redes semânticas, hierarquias de conceitos e frames. Esses diversos tipos de representações de conhecimento podem codificar o conhecimento sobre vários domínios de problema [22].

O conhecimento do agente pode ser estruturado de acordo com modelos específicos [197, 198, 217] (Figura 12). Há diferentes modelos propostos para a estruturação de conhecimento, mas os *elementos do conhecimento* são normalmente expressos por “componentes mentais”. Em geral, os elementos do conhecimento usados são crenças, objetivos, ações e planos [137, 197, 198, 217]. Este trabalho se concentra nesse modelo de estruturação de conhecimento, porque muitos projetos consideram o modelo BID (belief-desire-intention) [197, 198] como a linha de base para a estruturação do conhecimento do agente [37, 137, 217].

As *crenças* do agente são elementos do conhecimento que descrevem informações sobre o agente propriamente dito, o ambiente e seus parceiros. Um *objetivo* pode ser alcançado por meio de diferentes planos. Um *plano* descreve uma

estratégia para alcançar um objetivo interno do agente, e a seleção dos planos baseia-se em crenças de agentes. Dessa forma, o comportamento de agentes é orientado pela execução de seus planos que selecionam *ações* a fim de alcançar os objetivos definidos. As ações e os planos são usados para implementar os serviços de agente. Os planos são associados a precondições e pós-condições [137]. As *precondições* listam as crenças que devem ser mantidas a fim de executar o plano, enquanto as *pós-condições* descrevem os efeitos da execução de um plano bem-sucedido usando crenças de um agente [137]. Apesar de os planos não fazerem parte da definição original do modelo BDI [197, 198], a abstração de plano é adotada por muitas arquiteturas e frameworks de SMAs (por exemplo [8, 37, 38, 39, 137]).

Há diferentes tipos de planos, e eles são específicos a aplicações [137]. Os sistemas multiagentes complexos incorporam diferentes tipos de planos: planos de reação, planos de decisão e planos proativos. Um *plano de decisão* implementa ações de tomada de decisão de um agente e é sempre executado antes de planos proativos ou reativos. Um *plano reativo* implementa ações reativas em resposta a um evento externo. Um plano reativo pode ser executado sem a execução prévia de um plano de decisão. Um *plano proativo* implementa ações proativas e é disparado em resposta a uma configuração interna específica do estado do agente.

3.3.2 Interação

O concern de interação é a propriedade de agente que implementa a comunicação com o ambiente externo. O *comportamento de interação* consiste em receber e enviar mensagens para outros agentes. O protocolo também pode implementar um comportamento sensorial. Esse comportamento consiste em observar os eventos nas entidades do ambiente.

Esse protocolo de interação tem suporte de um conjunto de *sensores* que detectam eventos externos. Os sensores detectam a chegada de novas mensagens de outros agentes e as alterações nos objetos do ambiente. Quando um evento externo é detectado, ele é traduzido para um formato de mensagem interna e armazenado em uma caixa de mensagens de agente. O concern de interação inclui *efetores* que

enviam mensagens ou geram eventos no ambiente. Quando um agente está executando ações e planos, ele precisa enviar mensagens a outros agentes. Uma mensagem é enviada a partir de uma ação simples ou de um plano associado ao agente, propriamente dito, ou com papéis do agente (Seção 3.4.2). Antes de enviar a mensagem, ela é armazenada em uma caixa de saída de agente e traduzida em um formato de mensagem específico de forma que o agente receptor possa interpretá-la.

A fim de receber e enviar mensagens para/de outros agentes, os sensores são associados a diferentes *infra-estruturas de comunicação*. Os sensores também estão associados a entidades de ambiente, que não estão cientes de que um agente está observando seus comportamentos. Alguns exemplos de entidades de ambiente incluem componentes de interface, componentes de banco de dados, objetos de negócio etc. As mensagens de agente estão estruturadas de acordo com a *linguagem de comunicação de agentes* (A*gent communication language*, ACL) [72, 75]. Agentes diferentes às vezes usam ACLs distintas. Por isso, as mensagens são traduzidas em um estilo de mensagem interna usado pelo agente.

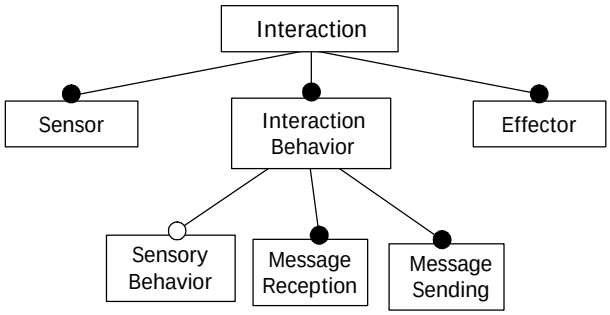


Figura 13. Concern de interação.

3.3.3 Adaptação

O concern de adaptação é a propriedade de agente que modifica o comportamento e o conhecimento do agente de acordo com os eventos internos e externos [22, 183, 207, 243]. O *protocolo de adaptação* consiste em observar os eventos internos ou ambientais relevantes, juntando as informações necessárias, selecionando e chamando os *adaptadores* associados [243]. Há dois tipos de

adaptação: *adaptação de conhecimento* e *adaptação de comportamento* (Figura 14). Eles seguem o mesmo protocolo básico. No entanto, a adaptação de conhecimento resulta na modificação de alguma parte do conhecimento de agente. A adaptação de comportamento resulta no cancelamento do plano ou na seleção de novos planos que devem ser executados em seguida.

A adaptação de agente ocorre em diversas circunstâncias: devido a eventos externos – por exemplo, recebimentos de mensagens – ou devido a eventos internos, alterações de crenças, definição de novos objetivos etc. Dessa forma, o agente deve ser capaz de observar o ambiente externo (concern de interação) e seu próprio comportamento e estrutura interna a fim de se adaptar. Os adaptadores são implementações de diferentes estratégias de adaptações, que podem ser simples ou sofisticadas. Os adaptadores sofisticados incluem técnicas de raciocínio (adaptação de conhecimento) [22, 172, 207] e planejadores (adaptação de comportamento) [207, 247].

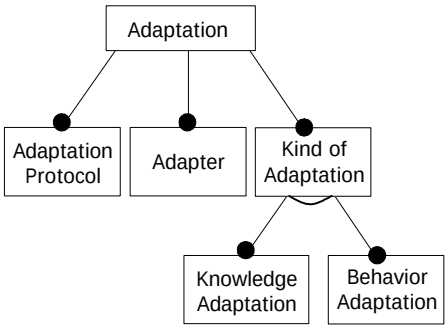


Figura 14. Concern de adaptação.

3.3.4 Autonomia

A autonomia normalmente significa que um agente tem controle sobre suas ações e pode agir de forma independente de outros. Em outras palavras, o concern de autonomia realiza o gerenciamento de objetivos de agente [119, 173, 180]. Para ser autônomo, o agente deve instanciar e alcançar seus objetivos. Os eventos motivam a instanciação de objetivos. Para alcançar os objetivos, os agentes têm suas próprias threads de controle (autonomia de execução) [2, 119], tomam decisões em relação a instanciações de objetivos (autonomia de decisão) [119, 180] e realiza ações sem uma

intervenção externa direta (autonomia proativa) [65, 119, 128, 183]. Estas são as três *dimensões de autonomia de agente* normalmente encontradas na literatura [65, 247] (Figura 15). A *autonomia de execução* é o controle das threads de agente. Ela oferece suporte à instanciação de threads e seu acoplamento aos agentes. Há diferentes estratégias para a instanciação de threads de agente. Por exemplo, uma única thread é instanciada quando o agente é criado, uma thread é instanciada para cada novo objetivo do agente, uma thread é criada para cada mensagem externa recebida etc.

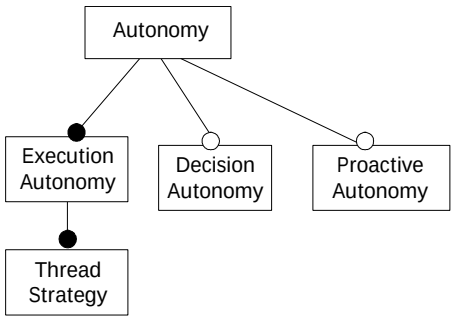


Figura 15. Concern de autonomia.

A *autonomia de decisão* consiste em tomar decisões sobre a instanciação de objetivos proativos e reativos. A capacidade de tomar decisão determina o curso de ação que um agente toma em um determinado momento. Esse curso de ação pode ser caracterizado como reativo ou proativo [58]. O comportamento de decisão é implementado por ações de decisão simples ou planos de decisão, associados aos objetivos de decisão. Essas ações ou planos são realizados quando ocorre um evento e eles decidem se um objetivo deve ser instanciado ou não. Os eventos externos podem causar a instanciação de objetivos reativos, enquanto os eventos internos podem originar a criação de objetivos proativos.

A *autonomia proativa*, ou proatividade, é a execução de ações ou planos sem uma solicitação explícita de outros agentes ou entidades de outros ambientes [61, 207]. Esses planos e ações proativos estão associados aos objetivos proativos. A conquista de objetivos proativos depende do grau de autonomia. Esse grau aumenta ou diminui de acordo com os sucessos ou falhas das ações proativas do agente no passado. Por exemplo, quando um agente propõe proativamente a seu usuário o

cancelamento de um compromisso, o usuário pode concordar ou não com a sugestão. A concordância do usuário aumenta, enquanto a oposição do usuário diminui o grau de autonomia do agente.

As dimensões da autonomia são essenciais para distinguir agentes de objetos. Os objetos são controlados pela parte externa, em oposição a agentes que têm um comportamento autônomo que não pode ser controlado externamente. Os agentes podem decidir dizer "não" a uma solicitação de serviço, iniciar ações conforme desejar e incorporar suas próprias threads de controle. Os agentes reativos (Seção 3.2.3) incorporam a autonomia de execução e podem incluir também a autonomia de decisão. No entanto, os agentes cognitivos (Seção 3.2.3) também incluem autonomia proativa. Os atores ou os objetos ativos [2] são agentes simples que são dotados apenas de autonomia de execução.

3.4
Concerns adicionais

Além dos concerns do agente básicos, o desenvolvedor de agente pode ter de enfrentar *concerns adicionais*. O agente pode ter de colaborar com outros agentes (Seção 3.4.1), exercer diferentes papéis nas colaborações (Seção 3.4.2), ganhar novo conhecimento para melhorar o desempenho (Seção 3.4.3) e mover de um ambiente para outro (Seção 3.4.4).

3.4.1
Colaboração

A colaboração é vista como uma forma de interação mais sofisticada, uma vez que incorpora comunicação e coordenação [247]. A interação só está relacionada à comunicação, ou seja, ao envio e recebimento de mensagens. Durante a colaboração interagente, as mensagens são recebidas de agentes participantes e enviadas para eles. Contudo, a propriedade de colaboração também define como colaborar, ela trata do *protocolo de colaboração* (Figura 16). Esse protocolo define a forma como um agente de software pode interagir com outros agentes em uma organização

multiagentes. Um protocolo de colaboração simples consiste em uma sincronização com o agente que está esperando uma resposta.

Um agente pode colaborar com outros agentes a fim de usar os serviços do parceiro por meio de uma linguagem de comunicação [72, 75]. Os agentes colaborativos exercem papéis diferentes na tentativa de alcançar os objetivos e trabalham juntos com outros agentes em vários contextos. Os papéis são dependentes da aplicação e são específicos a cada contexto. A fim de cooperar, um plano é instanciado, e cada agente exerce um papel diferente.

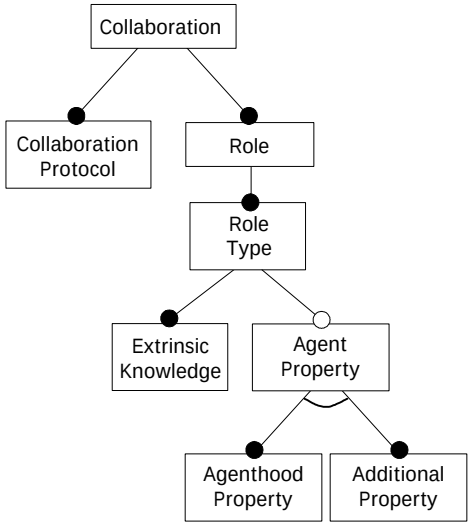


Figura 16. Concerns de papel e colaboração.

3.4.2 Papéis

Os papéis foram usados durante a conceitualização [223, 244], análise [249, 253, 263], projeto [136, 253, 261] e implementação [77, 136, 181] de sistemas multiagentes. Um agente exerce ou realiza diferentes papéis em um sistema multiagentes. Um tipo de agente possui dois tipos de papéis: um *papel intrínseco*, que implementa seu conhecimento intrínseco ou funcionalidade básica, e *papéis extrínsecos ou colaborativos*. Para simplificar, usamos papéis intercambiáveis e colaborativos neste texto. Os tipos de agente estipulam seu conhecimento intrínseco (papel intrínseco), enquanto a noção de um papel se concentra nos recursos ou no conhecimento extrínseco de um agente dentro de colaborações específicas. Um tipo

de papel captura como um agente interage com outros agentes em colaborações específicas. Neste trabalho, a definição de papéis é similar à definição encontrada no contexto da orientação a objetos [77].

Cada papel possui o conhecimento e o comportamento necessários para realizar as colaborações com outros agentes (Figura 16). Como um agente, um papel possui crenças, objetivos, ações e planos. Pode ter comportamentos específicos para interagir com outros agentes, algoritmos de decisão específicos e estratégias de adaptação específicas. Também pode ter comportamentos especializados para propriedades adicionais. Como conseqüência, o comportamento e a estrutura do papel são semelhantes ao comportamento e à estrutura do agente.

Entretanto, os papéis possuem alguns princípios [137, 146, 223] que os distinguem dos agentes:

- *Dependência*: um papel não pode existir sem um tipo de agente. De acordo com [146], as ações e os planos do papel podem ser definidos em termos das ações e dos planos dos tipos de agente, mas não vice-versa.
- *Dinamicidade*: um papel pode ser adicionado ou removido durante o ciclo de vida de um agente. Isso ocorre no nível da instância; instâncias diferentes do mesmo tipo de agente podem ter papéis adicionados ou removidos durante o ciclo de vida.
- *Identidade*: o papel e o agente possuem a mesma identidade. O tipo de agente e seus papéis são vistos e podem ser manipulados como uma entidade.
- *Herança*: um papel para um tipo de agente também é um papel para qualquer subtipo de agente, e um superpapel é um papel para um tipo de agente se seu subpapel for um papel para o tipo de agente.
- *Multiplicidade*: diversas instâncias de um papel podem existir para uma dada instância de agente em um determinado momento. Uma instância de agente pode exercer vários papéis ao mesmo tempo, incluindo várias instâncias do mesmo papel.
- *Visibilidade:* o acesso ao agente é restringido por um papel. A visibilidade de um agente pode ser restringida às ações de um papel. Isso pode incluir as

ações intrínsecas do tipo de agente, mas excluirá as ações extrínseca dos demais papéis.

- *Abstratividade*: os papéis podem ser organizados em hierarquias.
- *Agregação*: os papéis podem ser compostos por outros papéis.

3.4.3 Aprendizagem

A propriedade de aprendizagem envolve o comportamento do agente responsável pelo refinamento ou pelo ganho de conhecimento baseado na experiência do agente. Os agentes cognitivos aprendem com base em sua experiência como resultado de suas ações, seus erros, as interações sucessivas com o ambiente externo e as colaborações com outros agentes [39, 172, 207]. A adaptação de conhecimento (Seção 3.3.3) também pode disparar o processo de aprendizagem do agente. Um agente observa os eventos a fim de reunir informações necessárias para os propósitos da aprendizagem.

Em um SMA único, os agentes podem empregar diferentes técnicas para implementar o concern de aprendizagem. Há diversas técnicas de aprendizagem, mas o *protocolo de aprendizagem* geral (Figura 17) é o seguinte [22, 172, 207]: (i) um evento é detectado como relevante, (ii) o evento é capturado e as informações são reunidas a partir do contexto em que ele ocorreu, (iii) o algoritmo de aprendizagem processa as informações reunidas, (iv) as informações são armazenadas e levam a novas conclusões, (v) se uma nova conclusão é alcançada, o agente de conhecimento é alterado. A propriedade de aprendizagem influencia a autonomia do agente (Seção 3.3.4). Ao aprender um novo conhecimento, ele pode afetar as decisões do agente e a ativação de objetivos proativos. O concern de aprendizagem é visto como um tipo especial de adaptação [22, 172, 207]. A motivação para a separação da adaptação e dos concerns de aprendizagem é que encontramos vários exemplos na literatura em que esses concerns são tratados separadamente [37, 215].

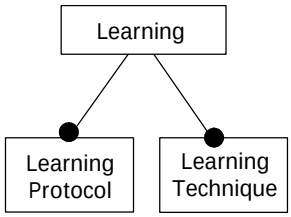


Figura 17. Concern de aprendizagem.

3.4.4 Mobilidade

O concern de mobilidade incorpora o comportamento para implementar a noção de que um agente se desloca em direção a ambientes diferentes. Vários tipos de agentes e papéis podem ter a propriedade de mobilidade. Durante a execução de seus planos, um agente móvel pode ter de se mover de um ambiente em uma rede para outro a fim de alcançar seus objetivos. Muitas facetas da estratégia de mobilidade devem ser consideradas [240], incluindo a especificação de quais elementos do agente (papéis ou tipos de agente) são móveis, as descrições das circunstâncias em que o agente precisa se mover, a partida para ambientes remotos, o retorno ao ambiente inicial e o controle de seu itinerário.

Os sistemas de agentes móveis existentes seguem duas formas de mobilidade (Figura 18), dependendo dos constituintes do agente que podem ser migrados [78]. Por um lado, os sistemas de agentes móveis que oferecem suporte à migração do estado de execução (mobilidade forte) e, por outro lado, aqueles que não oferecem esse tipo de suporte (mobilidade fraca). Neste trabalho, nós nos concentramos na mobilidade fraca na qual somente o código do programa e os dados da instância são movidos. Muitos frameworks de mobilidade oferecem suporte à mobilidade fraca [78].

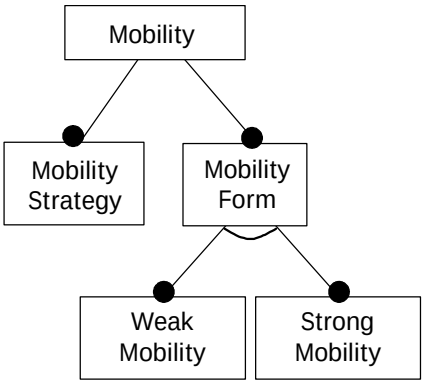
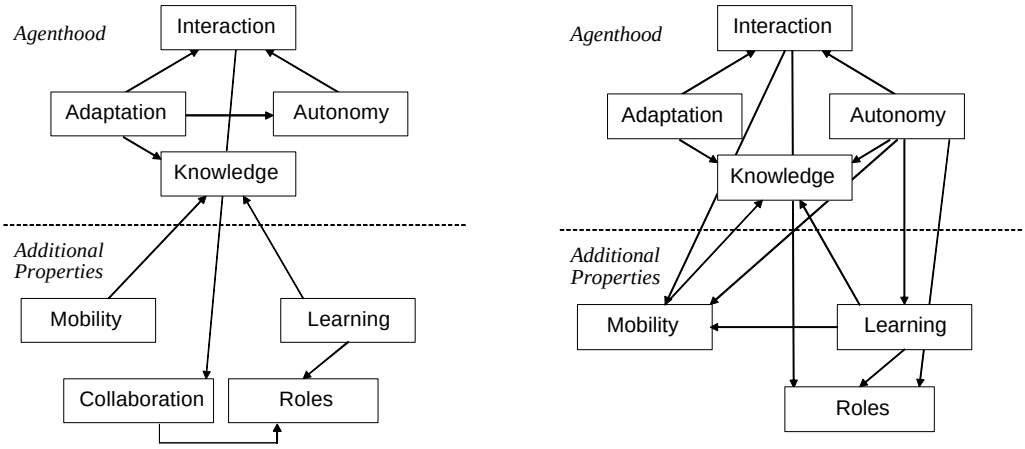


Figura 18. Concern de mobilidade.

3.5 Relacionamentos entre concerns do agente

Apesar de termos apresentado cada concern separadamente, os concerns do agente não são ortogonais – em geral, eles interagem entre si [7, 8, 112, 189, 190]. A Figura 19a ilustra os relacionamentos entre os concerns em relação aos agentes reativos em um determinado sistema multiagentes (Seção 4.1). Por exemplo, a adaptação depende da autonomia porque é necessário adaptar o conhecimento de agente quando a propriedade de autonomia decide aceitar uma mensagem recebida (Figura 19a). Além disso, algumas propriedades de agente são sobrepostas, como a interação e a colaboração. A colaboração é vista como uma forma de interação mais sofisticada, porque incorpora comunicação e coordenação (Seção 3.4.1).

Além disso, os relacionamentos entre concerns do agente dependem da complexidade do agente e do tipo do mesmo [7, 8, 112, 189, 190] (Seção 3.2.3). A Figura 19b ilustra os relacionamentos entre os concerns em relação aos agentes cognitivos em um sistema multiagentes diferente (Seção 5.1). Nesse sistema, o concern de adaptação não está relacionado ao concern de autonomia. O concern de interação está relacionado não só à adaptação e à autonomia, mas também a papéis e à mobilidade.



(a) agentes reativos **(a)** agentes cognitivos

Figura 19. Relacionamentos entre concerns do agente.

3.6
Soluções orientadas a objetos: pontos fortes e pontos fracos

Esta seção analisa a interação entre agentes e objetos. Ela investiga até que ponto as abstrações orientadas a objetos oferecem suporte à separação explícita dos concerns do agente e à construção de agentes de software com reusabilidade e manutenibilidade. Gasser [100] e Shoham [217] reconhecem as características potenciais do paradigma orientado a objetos para a construção de agentes. No entanto, há várias formas de associar propriedades de agente e objetos. Foram propostas algumas tentativas de lidar com a complexidade do agente usando abstrações orientadas a objetos. Essas propostas podem ser encontradas na literatura [8, 137, 240].

Um sistema multiagentes pode ser considerado um sistema orientado a objetos no qual os objetos tenham associado outras propriedades (Seção 3.2.1). Se os objetos pudessem adquirir essas propriedades de forma flexível, os agentes poderiam ser construídos explorando ferramentas e técnicas orientadas a objetos. Há várias formas de construir agentes a partir de objetos: definindo interfaces e comportamentos comuns em superclasses abstratas, envolvendo objetos com comportamento do agente usando técnicas de composição etc. No entanto, essas formas apresentam problemas

para transformar os objetos existentes em agentes e para desenvolver agentes a partir de um esboço, em especial, se forem necessárias várias propriedades de agente.

Esta seção analisa essas alternativas problemáticas. Analisamos os problemas encontrados ao tentar construir sistemas multiagentes com uma boa estrutura orientada a objetos para modularizar vários concerns de agente. A maioria desses problemas foi detectada: (i) durante a realização de uma extensa pesquisa na literatura [7, 8, 33, 112, 137, 217, 240], (ii) durante a investigação detalhada de alguns sistemas multiagentes [51, 93, 215, 257] e (iii) durante o desenvolvimento e a implementação de sistemas multiagentes [51, 89, 205, 212, 218] (Capítulo 7). Esta seção discute e ilustra as questões relacionadas aos problemas encontrados ao lidar com concerns do agente com base no paradigma de objetos, a saber: (i) agentes vistos como objetos, (ii) explosão de classes, (iii) replicação de código, (iv) esquizofrenia de agentes e (v) crosscutting concerns do agente.

3.6.1 Agentes vistos como objetos

Apesar de haver muitas motivações para o desenvolvimento de agentes de software baseados no paradigma de objetos, essa tarefa não é trivial devido às diferenças entre objetos e agentes (Seção 3.2.1). Não está claro como organizar a estrutura interna de agentes. Uma etapa inicial óbvia é usar uma única classe abstrata para representar o comportamento comum dos tipos de agente do sistema. De fato, os projetistas de frameworks orientados a objetos [20, 120, 152, 181] usam uma única classe abstrata para capturar todas as propriedades internas de um agente. Os desenvolvedores de agente criam subclasses para projetar tipos de agente diferentes. Cada instância dessas subclasses é um “objeto de agente”, ou seja, uma instância de agente. Um exemplo de “objeto de agente” é um agente de usuário: ele pode ser desenvolvido usando uma classe UserAgent que possui métodos definidos para a implementação dos serviços básicos do agente e para a interação, adaptação, aprendizagem, deslocamento etc.

Embora seja desejável que um agente apareça como uma classe única, esse esquema resulta em uma implementação e um projeto do agente sem separação de

concerns. O projeto do agente é bastante complexo e difícil de entender, manter e reutilizar na prática [7, 8, 112]. Ele não oferece suporte direto para o tratamento e a reutilização de propriedades de agente e da funcionalidade básica do agente separadamente. Além disso, nem sempre é fácil desenvolver agentes de software adequados, uma vez que os desenvolvedores de sistemas multiagentes devem levar em consideração muitas propriedades de agente. O ideal é que os desenvolvedores de sistema de agentes apliquem técnicas de estruturação especiais e que usem formas disciplinadas de associar as propriedades de um agente com suas funcionalidades básicas. Uma outra solução é o uso do mecanismo de herança, discutido na próxima seção.

3.6.2
Explosão de classes e replicação de código

Ao usar herança para o desenvolvimento de agentes, o comportamento comum de qualquer tipo de agente pode ser agrupado em uma classe chamada Agent. Podemos produzir diferentes tipos de agentes usando a especialização dessa classe. No entanto, se a herança for a única técnica de projeto usada, o projeto do agente apresenta vários problemas [7, 8, 112]. O principal deles é a presença de código replicado ao longo da árvore de herança [7, 8]. Ademais, muitas outras classes devem ser criadas para oferecer suporte à composição de vários concerns do agente.

A razão é que esses agentes específicos combinam diferentes concerns do agente de várias formas [7, 8, 112]. Por exemplo, as estratégias de interação, os algoritmos de tomada de decisão, as estratégias de aprendizagem e os protocolos de colaboração podem ser combinados de maneiras diferentes para cada tipo de agente. Esses comportamentos podem ter de ser replicados em diferentes pontos da hierarquia (herança simples) ou as hierarquias complexas precisam ser gerenciadas (herança múltipla) [7, 8]. Os relacionamentos estáticos entre as classes expressos por meio de herança dão origem a esses problemas. Além disso, o código entrelaçado causa anomalias na herança devido ao forte acoplamento dos concerns. Fica impossível redefinir uma implementação de método ou o concern especial misturado em uma subclasse sem redefini-los.

3.6.3
Esquizofrenia de agente

O uso de *delegação* no projeto orientado a objetos oferece suporte a uma maior flexibilidade para separar e compor concerns [7, 8]. Diferente da composição baseada apenas em herança, a delegação oferece suporte à separação de concerns usando árvores de herança. Cada árvore de herança deve tratar de um determinado concern do agente. Por exemplo, a Figura 20 apresenta como a delegação pode ser usada para separar o conhecimento básico dos tipos de agente das estratégias de aprendizagem usando o padrão Estratégia [79]. A composição é alcançada inserindo chamadas a métodos das classes Learning em métodos dos tipos de agente.

Essa solução melhora a modularização dos concerns do agente uma vez que são, em parte, separados. No entanto, ela tem algumas desvantagens. Primeiro, a separação explícita de concerns não é alcançada. Por exemplo, as chamadas aos mecanismos de aprendizagem estão entrelaçadas com as funcionalidades básicas do agente. Segundo, precisamos também criar algumas classes adicionais para tratar da composição dos concerns (por exemplo, a classe Learning na Figura 20). Além disso, o problema de separar o agente que deve ser um único objeto refere-se à "esquizofrenia de agente" [55, 136]. A Seção 4.2 discutirá os problemas relacionados ao uso de um padrão de projeto baseado em delegação, o padrão Mediador [79], para a separação de concerns do agente no contexto de um exemplo de SMA. O Capítulo 5 ilustrará os problemas enfrentados usando outros padrões de projeto com base em delegação para lidar com concerns do agente.

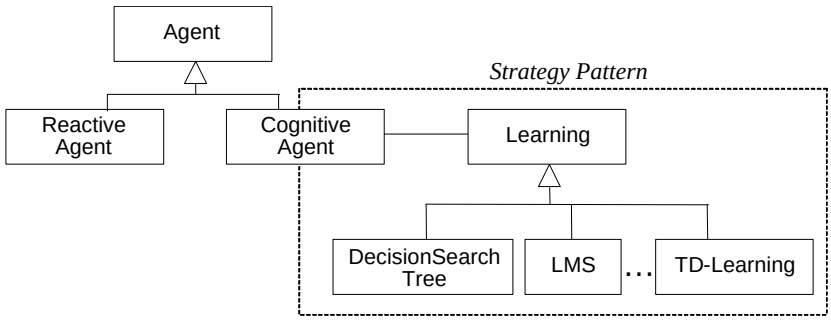


Figura 20. Uso da delegação para a composição de concerns do agente.

3.6.4
Crosscutting concerns do agente

Uma arquitetura de agente incorpora muitas propriedades, inclusive interação, adaptação, autonomia, colaboração, aprendizagem e mobilidade. No entanto, sua implementação normalmente afeta muitas partes do código e classes do agente. Muitas implementações das arquiteturas de agente levam a códigos entrelaçados, misturando concerns como autonomia, interação, colaboração, mobilidade e a funcionalidade principal dos tipos de agente. Às vezes é difícil incluir os concerns do agente em aplicações orientadas a objetos novas ou já existentes.

Considere o problema da introdução de uma estratégia de aprendizagem em uma aplicação orientada a objetos. Por sua natureza, a aprendizagem passa por muitas classes e métodos da aplicação. Além disso, a estratégia de aprendizagem deve ser aplicada de forma uniforme em qualquer adição à medida que a aplicação se desenvolve. A estratégia de aprendizagem aplicada também pode se desenvolver. Capturar os concerns do agente como uma estratégia de aprendizagem de forma disciplinada é difícil e propenso a erros em uma linguagem de programação orientada a objetos.

Contudo, as abstrações orientadas a objetos são úteis para representar vários concerns do agente, como elementos de conhecimento e tipos de agente. Entretanto, embora os mecanismos de modularidade hierárquica de um paradigma orientado a objetos sejam extremamente úteis e os padrões de projeto (Seção 2.1.2) ofereçam uma grande variedade de formas flexíveis de combinar concerns com base em classes e objetos, eles são inerentemente incapazes de modularizar todos os concerns do agente; em especial, porque alguns afetam as classes. A Seção 4.1 apresentará um exemplo que ilustra os problemas associados a crosscuting concerns do agente. O Capítulo 5 mostrará por que os padrões de projeto conhecidos não são capazes de separar concerns do agente, como aprendizagem, adaptação e interação.

3.7
Arquiteturas de agente e suas limitações

Os desenvolvedores de software normalmente adiam os concerns do agente para as etapas de implementação e projeto detalhado. No entanto, alguns pesquisadores tentaram formular arquiteturas de agente que orientem a separação de diferentes concerns do agente a fim de oferecer suporte a uma melhor manutenção e reutilização no nível da arquitetura. A idéia é superar ou, pelo menos, minimizar os problemas apresentados nas seções anteriores. As arquiteturas de agente, como JAFIMA [137], Brainstorm [8] e SkeletonAgent [37, 38], propõem muitas e diversas formas de estruturar concerns do agente. Cada arquitetura de agente incorpora diferentes padrões arquiteturais [35]. Esta seção oferece uma comparação e uma avaliação das arquiteturas investigadas, além de apresentar uma identificação das limitações primárias de sua aplicação na prática para desenvolver sistemas multiagentes com usabilidade e manutenibilidade. A avaliação usa o framework TAO para discutir as arquiteturas dos concerns do agente e para distinguir uma arquitetura de outra.

A investigação limita-se a arquiteturas de software cujo objetivo é oferecer suporte à separação de vários concerns do agente a partir da funcionalidade básica dos agentes. Não incluímos aqui as soluções arquiteturais dedicadas exclusivamente ao suporte à implementação de um ou dois concerns específicos. Esse é o caso dos frameworks de mobilidade [132], como Aglets [152] e Ajanta [133], frameworks de adaptação e aprendizagem, como ABLE [1], e linguagens ou arquiteturas de coordenação [167], como as TSpaces [155], T-Rex [219, 220], TuCSoN [187], LGI [171] e MARS [36]. Tampouco incluímos algumas plataformas de agente, como JADE [20], Retsina [232], Zeus [181], InfoSleuth [179], JAFMAS [40], Plangent [185] e FIPA-OS [196]. O objetivo desses frameworks é fornecer uma infra-estrutura e mecanismos associados. Eles apenas oferecem APIs para construir agentes de software, mas não objetivam permitir a separação de concerns do agente.

Por exemplo, JADE é um framework orientado a objetos para desenvolver aplicações de agente em conformidade com as especificações FIPA [74] para SMAs interoperáveis. O propósito é simplificar o desenvolvimento enquanto garante a

conformidade com os padrões por meio de um amplo conjunto de agentes e serviços de sistema. JADE é considerado um middleware de agente que implementa uma plataforma de agente e um framework de desenvolvimento. Ele oferece um API padrão no qual os engenheiros de SMAs podem confiar. Suas interfaces oferecem recursos para gerenciar basicamente três entidades: a plataforma de agente, agentes individuais e organizações de agente. No entanto, o JADE e outras implementações de agente em conformidade com a especificação FIPA lidam com a funcionalidade do sistema que não oferece suporte às partes internas do agente e que são independentes das aplicações, como o transporte de mensagens, a codificação e a interpretação, ou o ciclo de vida do agente. Apesar de os frameworks orientados a objetos serem essenciais ao desenvolvimento de SMAs, os desenvolvedores desse tipo de sistema se vêem sem um método de desenvolvimento para estruturar os concerns do agente com base em abstrações orientadas a objetos. Além disso, eles têm muitas restrições arquiteturais o que pode tornar seu uso impraticável. Por isso, os frameworks de implementação possuem diferentes objetivos e não estão incluídos na comparação apresentada aqui.

As próximas seções analisam as arquiteturas de agente existentes que enfatizam a separação de concerns. Elas são classificadas em três categorias: (i) *arquiteturas em camadas* (Seção 3.7.1), (ii) *arquiteturas reflexivas* (Seção 3.7.2) e (iii) *arquiteturas baseadas em mediadores* (Seção 3.7.3).

3.7.1 Arquiteturas em camadas

Kendall et al. [137] propõem o padrão arquitetural *Agente em camadas* que usa o padrão arquitetural *Camadas* [35] (Seção 2.1.1) a fim de separar diferentes concerns do agente usando a abstração em camadas. O padrão possui sete camadas (Figura 21), mobilidade, tradução, colaboração, autonomia, raciocínio ou adaptação, camada de conhecimento e camada sensorial. Cada camada deve encapsular um determinado concern do agente. O concern de interação é modularizado em duas camadas: a camada de tradução e a camada sensorial. Essa abordagem arquitetural oferece suporte a quase todos os concerns do agente e trata de tipos de agentes

cognitivos e reativos [137]. A arquitetura em camadas não cuida explicitamente dos concerns de papel e aprendizagem. No entanto, Kendall e alguns colegas [137] argumentam que ela pode ser adicionada como uma outra camada.

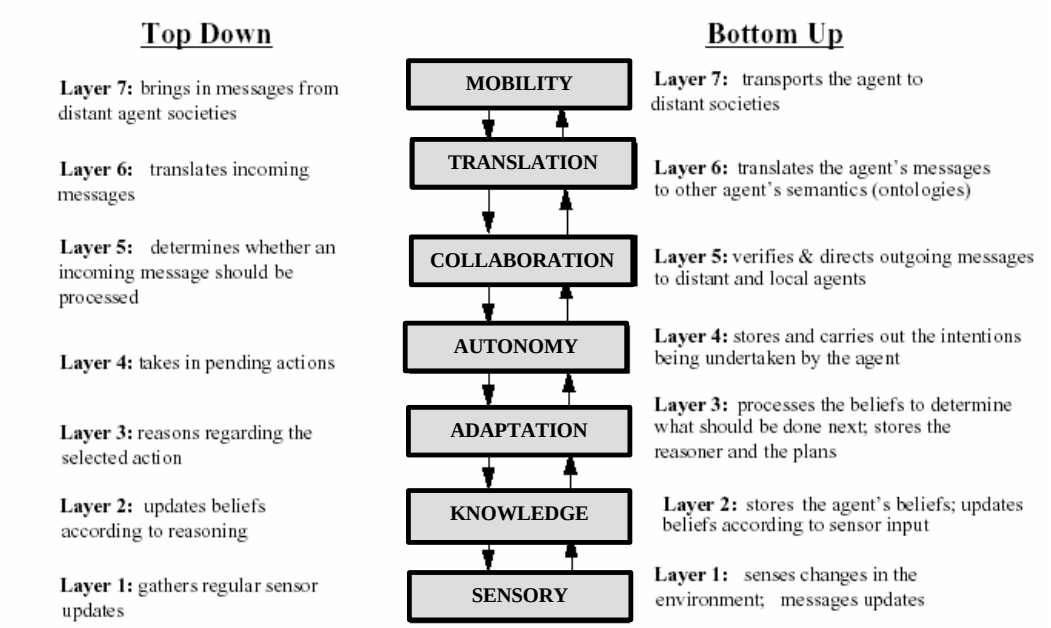


Figura 21. O padrão arquitetural de agentes em camadas [137].

A arquitetura em camadas modulariza os concerns de agência usando a abstração em camadas. Ela estabelece um estilo de composição no qual todas as interações apresentam um fluxo de informações de duas vias e somente as camadas adjacentes se comunicam entre si (Figura 21). Contudo, esse estilo de composição é muito restritivo uma vez que as propriedades de agente interagem entre si de várias formas (Seção 3.5). A arquitetura em camadas só deve ser usada quando os concerns estão relacionados entre si de duas formas, e as camadas podem ser adicionadas e removidas com facilidade.

O problema é que, à medida que a complexidade do agente aumenta, muitos concerns do agente, como adaptação, afetam diferentes camadas do agente (Figura 22). Além disso, a evolução dessa abordagem de projeto é incômoda, porque a remoção de qualquer uma dessas camadas não é algo trivial; requer a reconfiguração das camadas adjacentes. O trabalho citado não apresenta orientações para o

desenvolvimento de comportamento de agente a fim de acomodar novos concerns do agente, como aprendizagem, ou de remover os existentes.

Ademais, essa arquitetura de agente em camada ajuda a separar os concerns do agente somente no nível arquitetural. Quando os componentes arquiteturais – as camadas – são decompostos usando os padrões de projeto, conforme proposto pela abordagem de Kendall [137], a separação arquitetural dos concerns do agente não é preservada no nível do projeto detalhado. Seus padrões de projeto levam a um código menos entrelaçado, como quando o código de conhecimento se entrelaça com o código de adaptação, mas não o evita completamente (Figura 23).

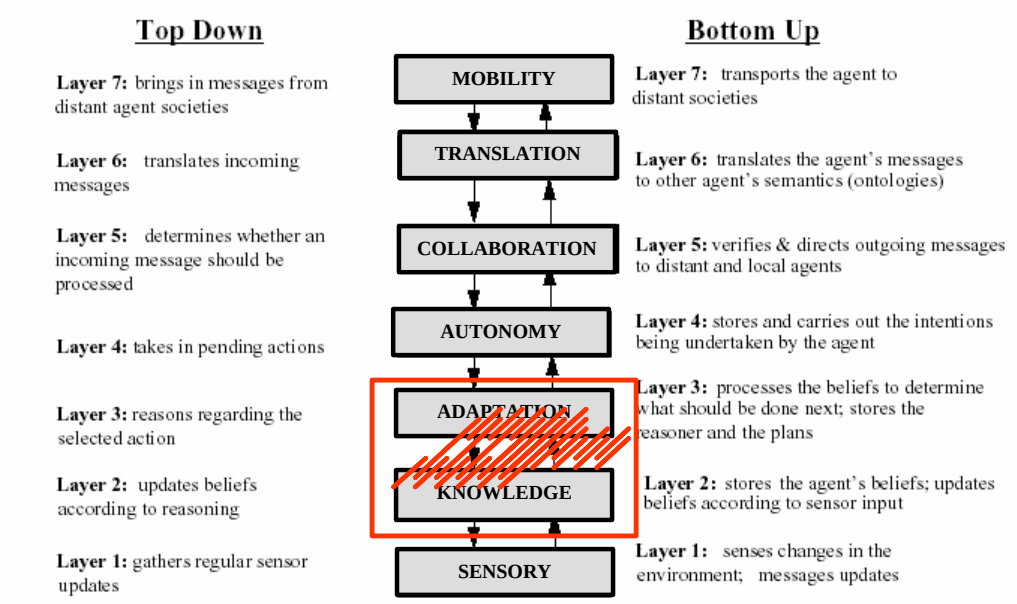


Figura 22. Adaptação na abordagem de agentes em camadas: comportamento crosscutting.

Por exemplo, ela propõe o uso do padrão Composição [79] e o padrão Observador [79] para estruturar respectivamente as camadas de Conhecimento e Adaptação (Figura 23). Essas crenças do agente podem ser primitivas ou compostas. As crenças compostas são formadas por várias crenças primitivas (padrão Composição). Assim, quando qualquer crença primitiva é alterada, os componentes de adaptação (adaptadores) precisam ser notificados para poderem se adaptar às crenças compostas associadas (padrão Observador). A Figura 23 mostra como as subclasses Adapter são os observadores das crenças compostas e primitivas. Quando uma crença é alterada, é necessário um ciclo de interpretação na camada de adaptação

a fim de encontrar qual parte do conhecimento do agente (uma crença, por exemplo) deve ser adaptada.

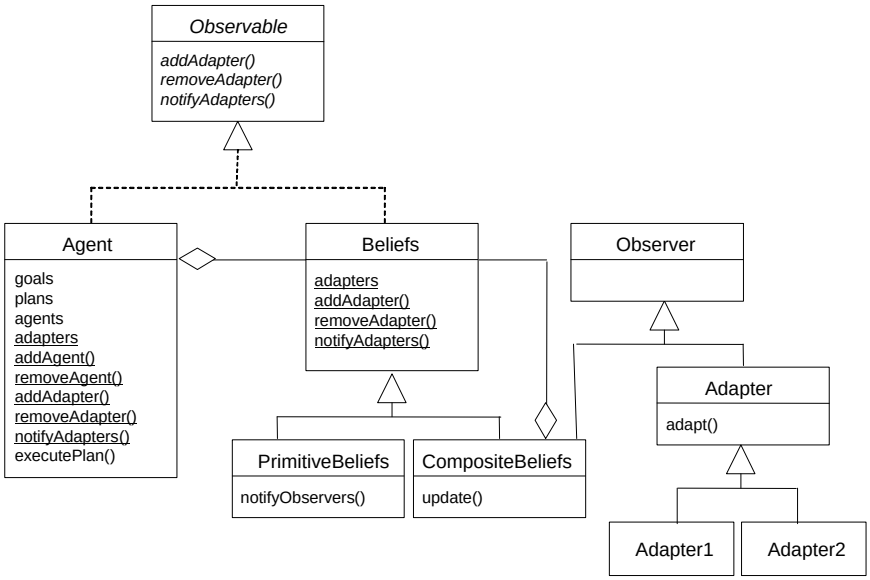


Figura 23. Estruturação baseada em padrões das camadas de adaptação e conhecimento.

O problema é que o código para notificar os componentes de adaptação (adaptadores) sobre os eventos relacionados ao conhecimento não pode ser desenlaçado e não ser espalhado usando a solução de projeto proposta. As referências explícitas para os adaptadores tampouco podem ser desentrelaçadas. Dessa forma, o concern de adaptação é parte da camada de adaptação e da camada de crenças (Figura 23). Cada subclasse de crença é instrumentada com métodos notifyAdapters(…) que são replicados para notificar as classes sobre alterações nas crenças correspondentes. Isso porque a padrão Observador é usado para implementar esse comportamento. Como conseqüência, é difícil entender, manter e reutilizar o código e o projeto do agente. A Figura 23 descreve os métodos que afetam o concern de adaptação.

3.7.2
Arquiteturas reflexivas

Amandi [7, 8] propõe uma abordagem arquitetural baseada na reflexão computacional [225], chamada Brainstorm. Essa abordagem usa o padrão arquitetural *Reflexão* [35] a fim de separar diferentes concerns do agente. A partir do ponto de vista da orientação a objetos [165], a reflexão computacional introduz duas novas

abstrações na engenharia de software: *metaobjetos* e um *protocolo metaobjeto (PMO)*. As arquiteturas de software reflexivas são organizadas em dois níveis: o *nível base* que contém os objetos e o *metanível* que é composto por metaobjetos. O PMO implementa a interface entre o nível base e o metanível. Ele é responsável pelo redirecionamento do fluxo de controle do nível base ao metanível em determinados pontos de execução de objetos do nível base. Alguns middlewares e linguagens de programação oferecem suporte à implementação de arquiteturas reflexivas, incluindo Guaraná [186] e MetaXa [107].

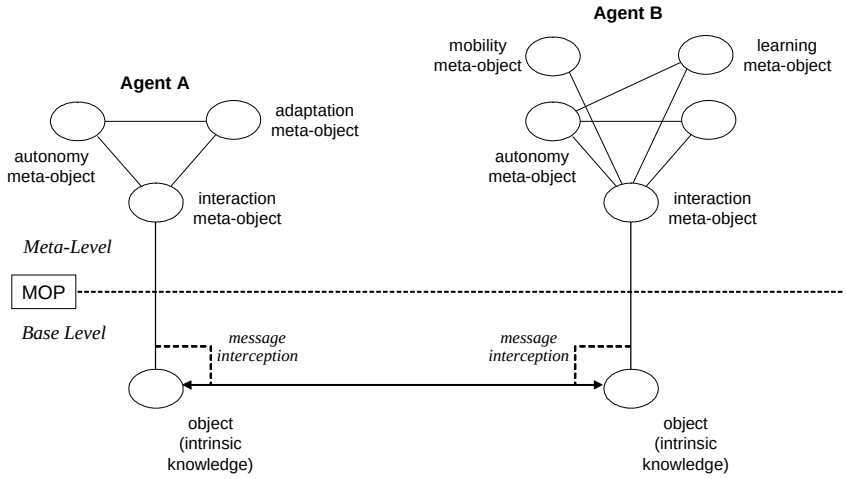


Figura 24. Agentes de brainstorm [8].

O Brainstorm explora os metaobjetos como abstrações para oferecer suporte à modularização de concerns do agente. Os metaobjetos interferem no comportamento passivo dos objetos, tornado-os agentes autônomos. Cada concern do agente é modularizado em metaobjetos específicos e associado a objetos passivos, que implementam o conhecimento intrínseco de um agente (Figura 24). Um objeto e um conjunto de metaobjetos definem um agente. Cada metaobjeto incorpora uma propriedade de agente específica a objetos simples. O brainstorm não trata de funções e concerns de mobilidade. A Figura 24 apresenta dois agentes baseados em uma arquitetura reflexiva. O Agente A é um agente reativo e possui três metaobjetos associados a ele. O Agente B é um agente cognitivo e possui cinco metaobjetos.

Os metaobjetos (no metanível) são normalmente conectados aos objetos definidos no nível base. A conexão causal estabelece-se quando um objeto recebe uma mensagem: ela é interceptada pelo metanível, que decide o que fazer. Por

exemplo, um metaobjeto de autonomia pode decidir negar de forma transparente uma determinada solicitação de outro agente. O metanível intercepta todos os eventos do nível base e modifica o comportamento dos objetos do nível base. Ao deter mensagens enviadas a objetos, os metaobjetos têm a oportunidade de realizar trabalhos em nome das propriedades de agente. Por exemplo, eles podem capturar as informações para alcançar o processo de aprendizagem (Agente B). Isso permite que a funcionalidade básica (objetos no nível base) seja escrita sem os concerns de agência, que, por sua vez, podem ser programados nos metaobjetos. Portanto, um sistema multiagentes pode ser considerado um sistema orientado a objetos que possui um conjunto de metaobjetos associados a ele para representar as propriedades de agente típicas.

Briot e Guessoum [112] propõem uma arquitetura reflexiva similar para o desenvolvimento de agentes de software. Esse tipo de arquitetura oferece dois benefícios: (i) melhora a separação de concerns do agente uma vez que são localizados pelos metaobjetos e (ii) oferece o suporte à transformação transparente de objetos em agentes. No entanto, essa arquitetura tem algumas desvantagens. Primeiro, a composição de vários metaobjetos não é trivial. Os metaobjetos são objetos e sua composição encontra-se na herança e nos mecanismos de delegação, o que leva aos mesmos problemas apresentados nas seções anteriores. Segundo, as abordagens baseadas em reflexão precisam de protocolos metaobjetos e arquivos de configuração para conectar objetos e metaobjetos [163].

Terceiro, um metaobjeto pode algumas vezes ser associado a um objeto. É restritivo porque várias propriedades de agente (metaobjetos) podem afetar diretamente o conhecimento intrínseco do agente (objeto). Quarto, algumas delas são naturalmente interativas e podem interferir uma na outra (Seção 3.5). Por exemplo, muitas estratégias de aprendizagem requerem uma associação com os componentes de adaptação, interação e autonomia. Quando a composição de muitas propriedades de agente não tem suporte adequado, há grandes possibilidades de ocorrer crosscutting concerns (Seção 3.6.4). Como conseqüência, é difícil compor diferentes propriedades do agente para produzir tipos heterogêneos de agentes. Uma abordagem

alternativa para lidar com a interferência durante a composição em um SMA é usar soluções baseadas em mediadores, descritas na próxima seção.

3.7.3 Arquiteturas baseadas em mediadores

O uso de mediadores é uma abordagem arquitetural para tratar da composição de concerns do agente que interagem de diversas formas. Os padrões de composição, como o padrão Mediador [79], o padrão Composição [79] e o padrão Cadeia de Responsabilidade [79], são soluções orientadas a mediadores. Eles fornecem meios que permitem a integração de propriedades de agente usando um *mediador*.

O padrão Mediador, por exemplo, define um componente mediador que encapsula como um conjunto de componentes, os *colegas*, interagem entre si. Essa solução promove um acoplamento leve ao evitar que os componentes façam referência explícita entre si e permite que os desenvolvedores de agente variem sua interação de forma independente. O framework Esqueleto do agente [37] realiza uma arquitetura baseada em mediadores ao implementar o padrão Composição. A Seção 4.1 explica as vantagens e desvantagens das abordagens baseadas em mediadores em termos de um exemplo. Os Capítulos 7 e 8 apresentarão uma comparação da abordagem orientada a aspectos, apresentada nos Capítulos 4 e 5, e da abordagem baseada em mediadores usando critérios qualitativos e quantitativos.

3.8 Resumo

Para poder aproveitar todos os benefícios da separação de concerns, os concerns separados devem ser correspondentes aos concerns necessários para lidar com um domínio específico. O desenvolvimento de SMAs envolve concerns de tipos extremamente diferentes. Este capítulo foi dividido em duas partes. A primeira parte apresenta o TAO (o framework conceitual *Taming Agents and Objects*), cujo objetivo é fornecer as bases para a compreensão dos concerns do agente a partir de um ponto de vista da engenharia de software. O TAO não é uma avaliação de conceitos existentes usados por métodos, linguagens e metodologias para SMAs, mas a

definição de concerns essenciais no desenvolvimento desses sistemas. A vantagem de ter um framework conceitual é oferecer suporte para desenvolver novos métodos de desenvolvimento e arquiteturas de agente (ou até mesmo linguagens de modelagem) com base em conceitos essenciais definidos e relacionados no framework. Cada conceito é visto como um concern para métodos de desenvolvimento e arquiteturas de agente que serão aplicados em diferentes fases do desenvolvimento de SMAs. O TAO usa modelos de características [130] para descrever os concerns do agente.

Ele classifica os concerns de agência em três categorias: (i) concerns fundamentais, (ii) concerns de agência e (iii) concerns adicionais. O framework enfoca quatorze concerns. Ressaltamos que essa lista não é exaustiva; compilamos um conjunto de concerns de engenharia de software que foram freqüentemente encontrados em diferentes sistemas multiagentes. O TAO pode ser adaptado para diferentes domínios uma vez que sua ontologia básica pode ser estendida para acomodar novos concerns do agente necessários em diferentes domínios de aplicações de agente. Ele permite que as equipes de pesquisa comparem e discutam as arquiteturas de agente e os métodos de desenvolvimento com base na terminologia unificada dos fundamentos propostos. O TAO define outros conceitos de SMAs [223], como *organizações* de agente, mas esses conceitos não foram apresentados uma vez que o enfoque deste trabalho está na arquitetura interna de agentes.

A segunda parte deste capítulo defende que o paradigma OO dificulta o gerenciamento de concerns do agente. Apresentamos os pontos fortes e fracos do uso de abstrações orientadas a objetos no domínio de agente. De fato, a modularização de propriedades de agente baseadas no modelo de objeto tradicional é difícil de ser conquistada e possui limitações significativas. Discutimos alguns problemas: (i) explosão de classes, (ii) replicação de código, (iii) esquizofrenia de agentes e (iv) crosscutting concerns do agente. Passamos então para a análise das abordagens arquiteturais que tentam minimizar os problemas mencionados anteriormente. As arquiteturas de software investigadas usam estilos arquiteturais específicos (Camadas, Reflexão e Mediador) para estruturar concerns do agente, mas não são capazes de tratar da separação de crosscutting concerns do agente. Conseqüentemente, isso leva a agentes de software que são difíceis de serem reutilizados e mantidos.