4
Arquiteturas de agente: um método orientado a aspectos

"Inside every large program there is a small program trying to get out" (C. Hoare)

As arquiteturas de agente são inerentemente complexas e têm de lidar com muitos concerns do agente para atender aos requisitos do sistema [8, 91, 189]. O projeto de vários concerns do agente com as abordagens arquiteturais existentes normalmente aumenta, em vez de diminuir, sua complexidade e, como conseqüência, dificulta a tarefa de construir o SMA de alta qualidade (Seção 3.7). A baixa qualidade percebida em sistemas multiagentes existentes é normalmente atribuída a um projeto arquitetural pobre relacionado a concerns do agente [91, 190, 240]. Além disso, os desenvolvedores de software normalmente adiam os concerns do agente para as etapas de implementação. As propriedades de agência são, em geral, introduzidas em um sistema orientado a objetos de forma *ad hoc* [91, 189], levando a arquiteturas de agente difíceis de entender, manter e reutilizar (Seção 3.6).

Os concerns do agente devem ser tratados de forma estruturada e disciplinados durante a etapa de desenvolvimento arquitetural a fim de manter sob controle a complexidade causada pelo tratamento desses concerns. Se uma arquitetura que inclui suporte à separação de concerns do agente for escolhida desde o início, a manutenção e a reutilização ao longo do desenvolvimento de um SMA podem ser alcançadas. Há a necessidade de uma abordagem arquitetural que incentive o tratamento separado de cada propriedade de agente. Além disso, ela deve oferecer suporte a maneiras flexíveis de construir diferentes tipos de agentes de software. As abordagens arquiteturais existentes (Seção 3.7) não atendem a esses requisitos. Este capítulo trata de duas questões básicas:

(i) Como oferecer suporte à separação de concerns do agente na etapa arquitetural?

(ii) quais são as orientações arquiteturais para estruturar uma arquitetura de agente orientada a aspectos?

Nesse contexto, o capítulo apresenta um método orientado a aspectos que oferece suporte à separação de concerns do agente na etapa arquitetural. O método oferece diretrizes para o desenvolvimento de arquiteturas de agente orientadas a aspectos. As diretrizes devem ser seguidas nas etapas de implementação e projeto detalhado. Dessa forma, o método proposto oferece, logo no início da etapa de projeto arquitetural, o contexto no qual podem ser tomadas decisões mais detalhadas sobre concerns do agente e técnicas associadas nas etapas posteriores do desenvolvimento do SMA. O método identifica uma separação adequada de concerns do agente em termos de classes e aspectos. As diretrizes arquiteturais descrevem as soluções que são independentes de linguagens de programação ou frameworks de implementação do SMA.

As contribuições deste capítulo foram descritas em duas publicações. A primeira versão do método arquitetural foi publicada em um trabalho apresentado em um simpósio [80] e a versão completa em um periódico [95]. O restante deste capítulo está organizado da seguinte forma. A Seção 4.1 introduz um exemplo usado para ilustrar o método arquitetural. A Seção 4.2 apresenta os requisitos para concerns do agente que não são tratados por uma arquitetura baseada em mediadores e as abstrações orientadas a objetos com base em exemplos. A Seção 4.3 apresenta o método baseado em aspectos para o desenvolvimento de arquiteturas de agente. A Seção 4.4 introduz as diretrizes arquiteturais e as aplica no sistema Portalware. A Seção 4.5 descreve algumas questões de implementação. A Seção 4.6 discute algumas vantagens e desvantagens da aplicação da abordagem proposta e discute os trabalhos relacionados. A Seção 4.7 apresenta alguns destaques conclusivos.

4.1 Agentes no Portalware: um estudo de caso

Esta seção introduz um SMA a fim de ilustrar a aplicação de agentes e a praticabilidade do método orientado a aspectos apresentado na Seção 4.3. Esse sistema é derivado de um estudo de caso realizado no Grupo SoC+Agents [227] na PUC-Rio no Brasil (doravante Portalware [89]). O Portalware é um ambiente baseado na Web que oferece suporte ao desenvolvimento e gerenciamento de portais de

comércio eletrônico. Esse ambiente introduz uma abordagem sistemática para a produção e a manutenção de portais por meio da distribuição de funções de usuário. O Portalware também oferece suporte a recursos de comunicação para a coordenação e a organização de atividades de desenvolvedores de portais.

Os agentes reativos foram introduzidos ao Portalware a fim de auxiliar seus usuários nas atividades que consomem tempo e nas tarefas de usuários repetitivas. Escolhemos esse sistema como estudo de caso, porque os concerns do agente tratados neste projeto são aqueles típicos de muitas aplicações existentes. Esse SMA incorpora vários concerns do agente, incluindo não apenas concerns fundamentais (Seção 3.2) e concerns de agência (Seção 3.3). Também incorpora concerns adicionais (Seção 3.4), como colaboração (Seção 3.4.1) e papéis (Seção 3.4.2). Além disso, cada tipo de agente é associado a vários papéis.

São atribuídos muitos papéis a cada usuário do Portalware, sendo estes os principais: (i) *fornecedor de conteúdo*, (ii) *revisor* e (iii) *editor*. O fornecedor de conteúdo (CS) é responsável por fornecer informações ao portal (por exemplo, notícias). Um revisor analisa cada thread de conteúdo e pode alterá-las caso deseje ou seja necessário. O editor é responsável pela seleção entre as threads de conteúdo disponíveis para publicação e por tornar agentes individuais revisores ou fornecedores de conteúdo. Pode ser atribuído um ou mais papéis a cada usuário do Portalware.

O Portalware incorpora três tipos de agente: (i) *agentes de interface*, (ii) *agentes de informação* e (iii) *agentes de usuário*. Cada tipo de agente oferece serviços diferentes, mas todos implementam concerns de agência (Seção 3.3). Além disso, possuem também outras propriedades de agente. A Figura 25 apresenta os tipos de agente no ambiente Portalware e as diferentes propriedades associadas a eles. Os agentes de interface monitoram a interface gráfica a fim de interagir com os usuários do Portalware. Eles aprendem atalhos ao capturar as preferências e ao receber instruções explícitas do usuário. Por exemplo, os agentes de interface operam enquanto um fornecedor de conteúdo (CS) está criando threads, usando palavras-chave inseridas no documento e o modelo adquirido a partir das preferências do

usuário. Os agentes de interface não possuem recursos colaborativos, uma vez que não cooperam com outros agentes de software.

Os agentes de usuário representam os usuários do Portalware e são implementados para reduzir a necessidade de conversa cruzada entre os usuário. Como os editores, revisores e os FCs precisam se comunicar entre si para manter o portal da Web, os agentes de usuário exercem esses papéis e possuem protocolos de colaboração para cooperar entre si. O agente que está exercendo o papel de editor possui a responsabilidade de entrar em contato com os revisores prospectivos e os FCs e negociar com eles o uso de seus serviços.

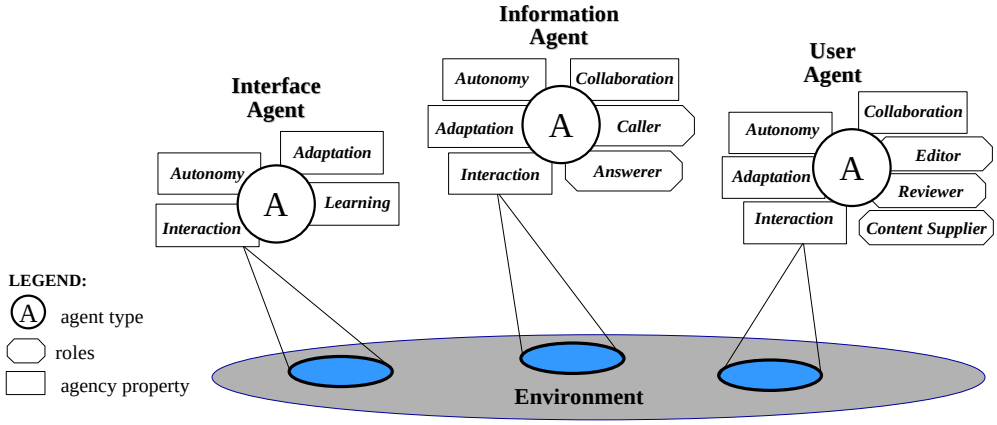

Figura 25. Agentes no Portalware.

Os usuários do Portalware normalmente precisam buscar informações armazenadas em diferentes bancos de dados. Cada agente de informação é acoplado a um banco de dados e contém planos para a busca de informação. O plano de busca determina o serviço de busca do agente. O agente de informação exerce um papel de chamador quando não consegue achar as informações no banco de dados acoplado. Ele exerce esse papel para chamar os outros agentes de informação e pedir essas informações. De forma similar, um papel de respondedor é ativado nos agentes de informação contatados de forma que possam receber a solicitação e enviar o resultado da busca. Observe que os agentes de informação podem exercer os papéis de respondedor e chamador.

4.2
Uso de uma arquitetura de agente baseada em mediadores

A modelagem orientada a agentes do Portalware é baseada na linguagem de modelagem de Elammari [66] e no framework conceitual TAO [223]. O projeto detalhado e arquitetural dos agentes de software enfatizou a separação de concerns. O projeto arquitetural dos agentes do Portalware seguiu um estilo orientado a mediadores (Seção 3.7.3) a fim de melhorar a modularização de concerns do agente. Como conseqüência, o padrão de projeto Mediador [79] exerce um papel central na arquitetura do Portalware. Cada concern do agente é modularizado em classes separadas (os *colegas*[4]); a classe Property (o *mediador*[4]) implementa a composição dos concerns do agente. O padrão Mediador também é útil para facilitar a associação de diferentes propriedades de agente com vários tipos de agente do Portalware.

O Portalware foi desenvolvido e implementado com abstrações da engenharia de software orientada a objetos. As notações UML [24] e a linguagem Java foram usadas para gerar projetos e implementações orientados a objetos, respectivamente. A Figura 26 representa um pedaço do projeto orientado a objetos do Portalware, que segue o estilo arquitetural baseado em mediadores. A Figura 27 apresenta algumas linhas de código Java. O projeto detalhado orientado a objetos enfatiza o uso dos padrões de projeto [79] a fim de produzir um sistema modificável e extensivo. Por exemplo, o padrão Objeto do Papel [19] foi usado para representar os papéis do agente, e o padrão Estratégia [79] foi usado para implementar diferentes planos de agente. O projeto do Portalware também incorporou padrões de projeto adaptados ao domínio do agente [137, 221].

Entretanto, como os mercados de portais mudam muito rapidamente, as deficiências nas técnicas de engenharia de software usadas logo ficaram aparentes para os desenvolvedores de sistemas. Para sobreviver, o Portalware deve permanecer extensível e modificável, e seu projeto e implementação devem ser capazes de responder com rapidez às mudanças. O projeto orientado a objetos do Portalware apresentou muitas limitações que resultaram no desenvolvimento de outros SMAs [205, 212, 218, 227]. Essas limitações foram a origem de muitas alterações intrusivas

e dificuldades de reusabilidade à medida que o sistema evoluiu. A primeira limitação é que as abstrações orientadas a objetos não são capazes de modularizar concerns do agente, como a colaboração.

A Figura 26 apresenta o problema do espalhamento (Seção 2.2.1) relacionado ao concern de colaboração no projeto do Portalware. Esse concern afeta várias classes de sistema, dificultando o entendimento do projeto e as alterações no código do sistema (Seção 3.6.4). Qualquer alteração relacionada ao concern pode envolver muitos módulos que implementam o concern. Ainda pior, não foi possível encapsular um concern do agente em um único padrão de projeto, conforme demonstrado na Figura 26, de forma que não pôde ser reutilizado como um bloco de projeto único. O problema do crosscutting não se limitou aos concerns de colaboração e papel; problemas semelhantes envolveram propriedades de agência – interação, adaptação e autonomia – e outras propriedades adicionais – mobilidade e aprendizagem.

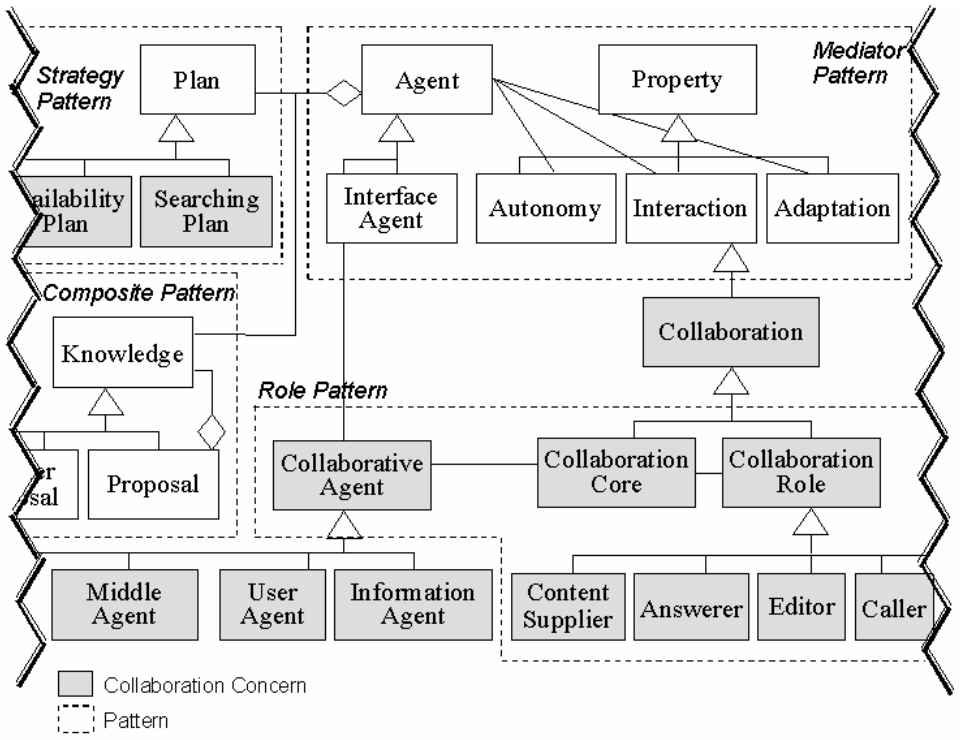

Figura 26. Um subconjunto do projeto orientado a objetos do SMA do Portalware.

[4] *Colegas* e *Mediador* são termos específicos do padrão *Mediador* [79].

Outro problema está relacionado ao fato de que não é fácil entender os concerns do agente acoplados a um tipo de agente específico. Por exemplo, não está claro no projeto que um agente de usuário está associado aos papéis de CS, revisor e editor (Figura 26). O uso do padrão Papel ajuda a modularizar os papéis, mas aumenta o número de classes no sistema (Seção 3.6.3), que são criadas para controlar a composição de instâncias de papel e de agente. Isso, por sua vez, resultou em vários desalinhamentos entre os artefatos de software gerados nas fases de desenvolvimento de nível superior e aqueles descritos no projeto orientado a objetos. Como um agente pretende participar de algumas colaborações heterogêneas, os papéis devem ser introduzidos na funcionalidade básica do agente de forma que sejam estruturados e não intrusivos.


```
public class PAgent {

    private String agentName;
   ...
   ...
    protected Interaction theInteraction;
    protected Autonomy    theAutonomy;
    protected Adaptation  theAdaptation;

//  Constructors

    public PAgent(String aName, Vector pl) {
        init();
        agentName = aName;
        theInteraction = new Interaction(this);
        theAutonomy = new Autonomy(this);
        theAdaptation = new Adaptation(this);
        planList = pl;
        System.out.println(" Name == " +
agentName);
   }
   ...
   ...
   ...
/* Interface  for  Interaction */
   public void receiveMsg(Message msg)
   {
       theAutonomy.makeDecision(msg);
       theAdaptation.adaptBeliefs(msg);
   }

   public void outcomingMsg(Message msg)
   {
       theInteraction.outcomingMsg(msg);
   }
}
```

Figura 27. Código Java do Portalware.

O problema não se limita ao projeto do sistema. O tratamento dos concerns de SMAs foi ainda mais difícil a partir de um ponto de vista de implementação, uma vez que muitos concerns do agente entram na funcionalidade básica do sistema. Há propriedades de agente, como a adaptação e a autonomia, que deviam ser modularizadas, mas sua implementação foi difusa durante os métodos que representavam os serviços de agente básicos. Em outras palavras, o uso de abstrações orientadas a objetos levou ao problema do entrelaçamento (Seção 2.2.1). A Figura 27 ilustra esse problema para a classe Agent. A falta de suporte no tratamento da natureza sobreposta e interativa das propriedades de agência (Seção 3.5) também limitou a compreensão, a manutenibilidade e a reusabilidade dos SMAs. Observe, por exemplo, que o corpo do método receiveMsg(…) (comportamento interativo do agente) é combinado com chamadas explícitas ao: (i) método makeDecision(…) que implementa o comportamento autônomo do agente e (ii) método adaptBeliefs(…) que implementa o comportamento adaptativo do agente. Assim, os desenvolvedores do Portalware enfrentaram a falta de suporte ao isolamento explícito e à composição de concerns do agente com base em abstrações orientadas a objetos disponíveis na linguagem Java.

Resumindo, o uso da arquitetura baseada em mediadores levou aos problemas a seguir: (i) o encapsulamento funcional da funcionalidade básica do agente foi perdido, (ii) concerns do agente ficam entrelaçados e espalhados entre si no projeto baseado em mediadores e (iii) a construção de tipos de agente heterogêneos é difícil. A complexidade inerente na construção de SMAs dificulta a promoção da separação de concerns. Como conclusão, a escolha de uma arquitetura de software adequada é fundamental para oferecer suporte à separação dos concerns do agente e à construção de arquiteturas de agente com reusabilidade e manutenibilidade. A próxima seção apresenta o método arquitetural orientado a aspectos proposto para superar essas limitações.

4.3
O método arquitetural orientado a aspectos

4.3.1
O propósito

O propósito do método proposto é auxiliar os engenheiros de software na modularização de crosscutting concerns do agente na etapa arquitetural. Ele pretende explorar os benefícios das abstrações orientadas a aspectos para estruturar as arquiteturas de agente, enquanto leva à construção de SMAs, fáceis de entender, manter e usar. Ele resolve os problemas normalmente detectados no desenvolvimento de SMAs baseados em abordagens arquiteturais existentes (Seção 3.6).

O método arquitetural baseia-se na premissa de que os agentes são objetos com propriedades adicionadas (Seção 3.2.1), e a noção de aspectos arquiteturais é usada para enriquecer os objetos com propriedades de agente de forma transparente. Os aspectos arquiteturais são usados para modularizar os crosscutting concerns do agente.

A abordagem arquitetural proposta não deve ser confundida com um framework orientado a objetos que oferece suporte a serviços e mecanismos no nível do sistema (Seção 3.7). De fato, a abordagem é independente dos frameworks de implementação de SMAs, como JADE [20], ZEUS [181] e Retsina [232], e linguagens de comunicação específicas, como ACL [75] e KQML [72]. A percepção de tal independência é importante para colocar o escopo do método arquitetural em perspectiva, uma vez que não enfatiza os detalhes dos frameworks de implementação ou das linguagens de comunicação.

4.3.2
Arquiteturas de agente orientadas a aspectos

O método proposto orienta a definição de arquiteturas de agente orientadas a aspectos. A arquitetura de software oferece os componentes para o tratamento de cada crosscutting concern do agente como um aspecto individual. Os componentes de uma arquitetura de agente orientada a aspectos representam os concerns de agência e os adicionais definidos no framework TAO (Tabela 3 – Seção 3.1.2). A arquitetura

orientada a aspectos é composta por dois tipos de componentes arquiteturais: (i) o *componente Kernel* que modulariza o conhecimento intrínseco do agente e (ii) *componentes aspectuais* que separam os crosscutting concerns do agente uns dos outros e do componente Kernel.

O componente Kernel é o componente agente que implementa os serviços fornecidos para os clientes do agente. Esse componente é responsável pela modularização dos elementos do conhecimento intrínseco do agente, como ações, planos, objetivos e crenças. Ele estabelece uma interface que torna disponíveis os serviços implementados pelo agente. O componente Kernel deve ser implementado como um conjunto de classes no nível do projeto detalhado. Esse componente pode representar um objeto existente, que precisa ser transformado em um agente.

Os componentes aspectuais, ou *aspectos arquiteturais*, representam aspectos (Seção 2.2.2) no nível arquitetural. Eles são usados para melhorar a separação dos crosscutting concerns em arquiteturas de agente. Além disso, modularizam as propriedades de agência – interação, adaptação e autonomia – e as propriedades adicionais – mobilidade, colaboração, papéis e aprendizagem. Essas propriedades de agente são isoladas do kernel do agente e entre si a fim de facilitar sua composição e a construção de diferentes tipos de agentes e de arquiteturas de agente heterogêneas. Um componente aspectual é refinado como um conjunto de aspectos e classes auxiliares, que também fazem parte do crosscutting concern no nível do projeto detalhado.

A Figura 28 apresenta um exemplo de arquitetura de agente orientada a aspectos, para agentes de informação (Seção 4.1). Ela apresenta os componentes arquiteturais e seus relacionamentos. Cada componente arquitetural possui uma ou mais interfaces. A modelagem da arquitetura baseia-se na linguagem de modelagem aSideML [41]. Essa linguagem oferece dois modos distintos para apresentar um aspecto: (i) visão completa ou visão do projeto detalhado e (ii) visão arquitetural ou condensada. A visão completa de um aspecto fornece uma descrição detalhada dos seus elementos. A Seção 2.2.2 descreveu a notação para a apresentação do projeto detalhado dos aspectos.

A Figura 28 descreve a visão arquitetural dos aspectos. Os aspectos arquiteturais também são representados como losangos. A visão condensada de um aspecto omite todas as informações sobre os elementos internos, exceto os nomes das interfaces crosscutting. Cada interface crosscutting é exibida como um pequeno círculo com o nome colocado ao lado (Figura 28). O círculo está ligado ao losango por uma linha sólida que representa o aspecto arquitetural.

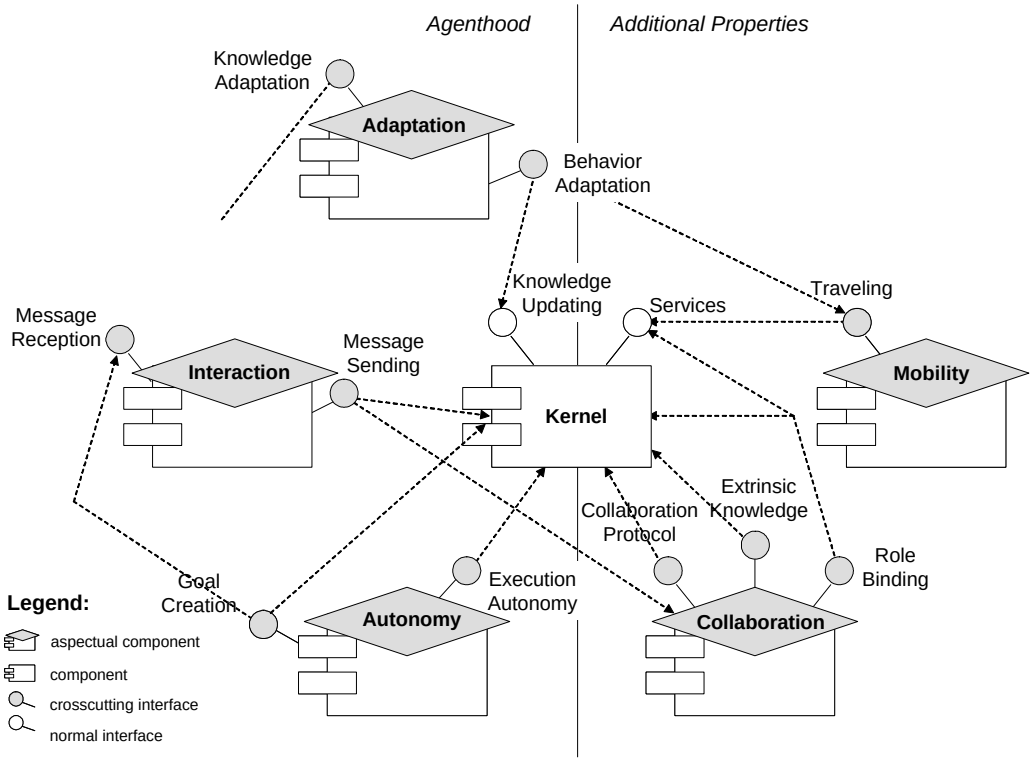

Figura 28. A arquitetura orientada a aspectos dos agentes de informação.

Interfaces dos componentes

A Figura 28 ilustra os componentes da arquitetura e suas interfaces. Cada componente aspectual está relacionado a mais de um componente arquitetural, representando sua natureza crosscutting. Os relacionamentos estão associados às interfaces do componente. Cada componente arquitetural possui uma ou mais interfaces. As interfaces estão divididas em dois grupos: (i) interfaces normais e (ii) interfaces crosscutting (Seção 2.2.2). Uma interface crosscutting interface é diferente

de uma interface normal. A última só fornece serviço a outros componentes. Já as interfaces crosscutting fornecem serviços ao sistema, mas também especificam quando e como um aspecto arquitetural afeta outros componentes arquiteturais. As interfaces normais são coloridas de branco e as crosscutting de cinza (Figura 28).

Um componente aspectual age de acordo com um conjunto de interfaces crosscutting. Ele pode perceber mais de uma *interface crosscutting* porque pode afetar vários componentes de diferentes formas. A interface de um aspecto arquitetural pode afetar o componente Kernel e outros aspectos arquiteturais. Uma interface de aspecto afeta os elementos internos de um componente ou os elementos de outras interfaces. O primeiro caso significa que o aspecto arquitetural afeta a estrutura interna ou o comportamento dinâmico do componente. O segundo caso significa que o aspecto afeta diretamente um aspecto arquitetural.

Propriedades arquiteturais

As arquiteturas de agente orientadas a aspectos exibem diferentes propriedades porque são compostas por componentes e aspectos. A arquitetura proposta tem as três propriedades do paradigma orientado a aspectos (Seção 2.2.2): (i) dicotomia baseada em aspectos, (ii) transparência e (iii) quantificação. A *dicotomia baseada em aspectos* significa a adoção de uma distinção arquitetural clara entre o componente Kernel do agente – a base – e os componentes aspectuais. Os aspectos arquiteturais modularizam os crosscutting concerns, e o componente Kernel modulariza os non-crosscutting concerns, ou seja, a funcionalidade básica do agente. Os componentes aspectuais são explicitamente especificados separados entre si e do componente Kernel.

A arquitetura de agente proposta também segue a propriedade de *transparência*: o componente Kernel não percebe os componentes aspectuais que o afetam. A idéia é que os elementos internos da funcionalidade básica do agente não precisam ser especificamente preparados para receber as melhorias proporcionadas pelos aspectos arquiteturais. Assim, a propriedade de transparência dos aspectos

arquiteturais oferece suporte à introdução transparente de propriedades de agente dentro do kernel.

Finalmente, a arquitetura proposta também tem a propriedade de *quantificação*, que é a capacidade de descrever componentes separados que afetam a estrutura e a dinâmica de um número arbitrário de componentes nas interfaces crosscutting. Essa propriedade permite a modularização dos crosscutting concerns do agente. A quantificação proporciona a capacidade de descrever algo como: “nos componentes arquiteturais P, sempre que surgir a condição C, faça uma ação A que é parte de um aspecto específico”.

4.4 As diretrizes

As diretrizes baseiam-se na premissa de que um agente é um objeto com mais recursos. A noção de aspectos arquiteturais é usada para enriquecer os objetos com propriedades de agente de forma transparente. Os objetos são, então, considerados a abstração básica para a construção do kernel do agente. No entanto, como um único agente é uma abstração muito mais rica do que um objeto, as diretrizes enfatizam a definição dos aspectos arquiteturais para a representação de diferentes propriedades de agente. As diretrizes ajudam na definição dos aspectos arquiteturais e sua associação com o kernel do agente. Elas consideram as propriedades de agente, adicionais à funcionalidade básica, de forma gradual. Primeiro o método orienta a organização dos concerns que têm mais suporte das abstrações orientadas a objetos e, em seguida, os demais concerns.

O método orientado a aspectos é dividido em algumas etapas, cada uma composta por um conjunto de diretrizes. As diretrizes auxiliam os engenheiros de software no desenvolvimento de arquiteturas de agente orientadas a aspectos (Seção 4.3.2). Elas são agrupadas em termos dos diferentes concerns do agente, a saber: (1) kernel do agente, (2) propriedades de agência, (3) tipos de agente, (4) propriedades adicionais e (5) papéis. A Figura 29 descreve as diretrizes arquiteturais. A primeira etapa é necessária somente para o desenvolvimento de arquiteturas de agente a partir do esboço. Os arquitetos de software, que desejarem transformar um objeto existente

em um agente, deverão ignorar essa etapa. A idéia é que o objeto existente represente o kernel do agente.

O método descreve também alguma orientação sobre como compor todos os componentes arquiteturais (etapa 6) e como desenvolvê-los e reutilizá-los ao longo do ciclo de vida do software (etapa 7). As diretrizes arquiteturais são aplicadas nos exemplos (Seção 4.1) a fim de mostrar o uso do método arquitetural em prática. As subseções a seguir mostram as etapas do método orientado a aspectos. Para simplificar, os termos "aspecto" e "aspecto arquitetural" serão usados de forma alternada, daqui em diante.

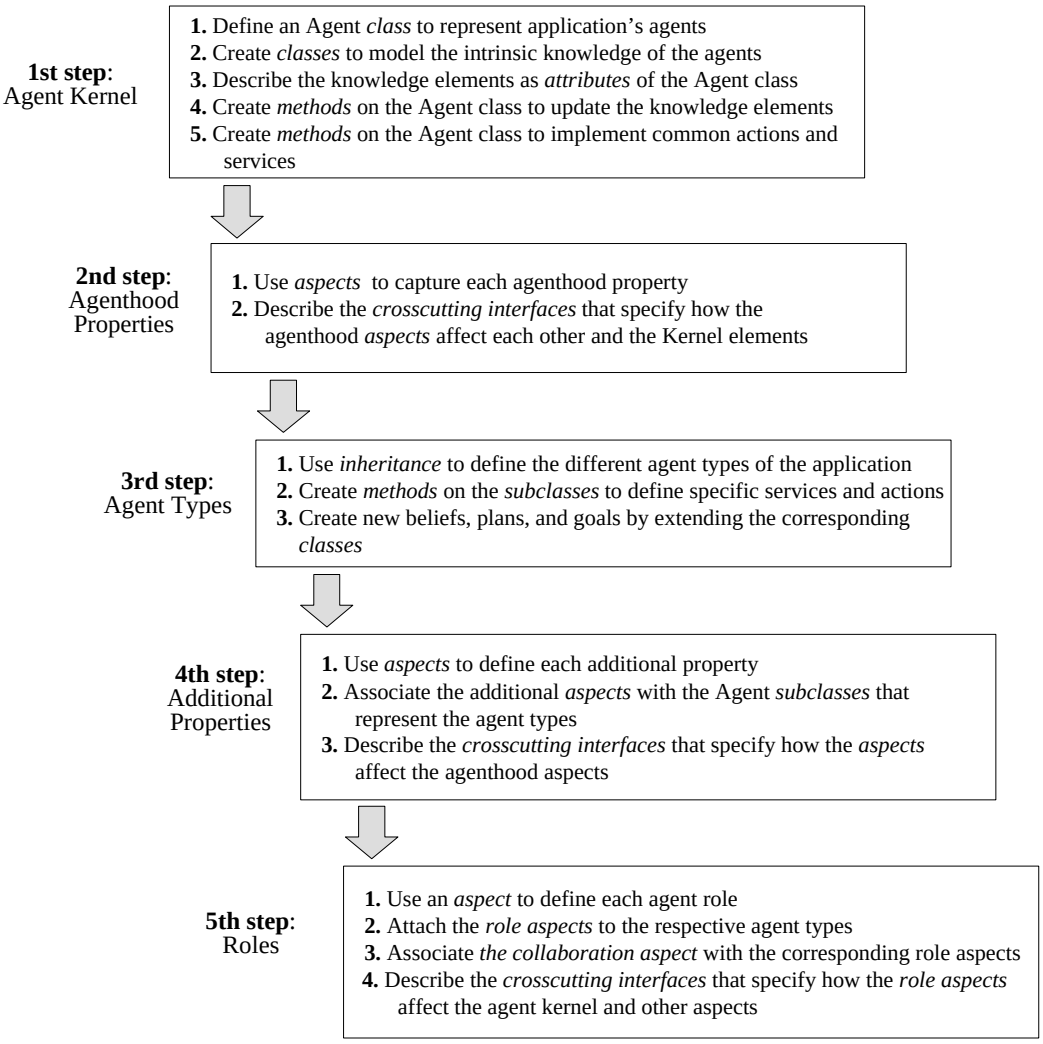

Figura 29. O método arquitetural orientado a aspectos para o desenvolvimento de SMAs.

4.4.1
Etapa 1: definição do kernel do agente

Use as classes para representar os elementos do componente Kernel de uma arquitetura de agente. O componente Kernel deve ser refinado como um conjunto de classes, representando os agentes propriamente ditos e seus elementos de conhecimento. A classe Agent especifica o comportamento comum aos tipos de agente do sistema e deve ser estendida para criar os tipos de agente da aplicação (*Etapa 3* - Seção 4.4.3). Os métodos da classe Agent são usados para atualizar os elementos do conhecimento intrínseco do agente e implementar as ações que são comuns a todos os tipos de agente em uma aplicação.

Os serviços específicos aos tipos de agente são implementados pelos métodos nas subclasses Agent, em que o método representa uma determinada ação. Os atributos das subclasses Agent podem ser usados para representar crenças de agente simples. As classes também são usadas para representar os elementos do conhecimento do agente: crenças, planos e objetivos (Seção 3.3.1). Os projetistas de aplicações devem subdividir as classes Belief, Goal e Plan para definir os tipos de agente de acordo com os requisitos da aplicação (*Etapa 3* - Seção 4.4.3).

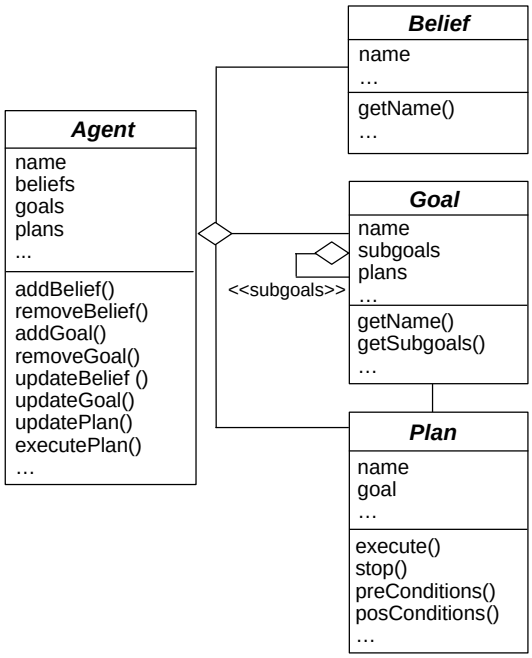

Figura 30. Kernel do agente.

A Figura 30 mostra as principais classes que representam o kernel de agentes do Portalware. Um agente do Portalware possui, entre outros atributos, um conjunto de crenças, objetivos e planos. Esses atributos são um conjunto de objetos Belief, Goal e Plan, e a classe Agent oferece operações-padrão para a busca, inserção, atualização e exclusão de um elemento. Cada agente tem um nome, que é uma crença simples que identifica o agente no sistema. Os serviços dos agentes são implementados pelos métodos nas subclasses Agent, em que cada método representa uma determinada ação.

4.4.2 Etapa 2: definição das propriedades de agência

Cada propriedade de agência deve ser modelada como um aspecto arquitetural devido à sua natureza crosscutting (Seção 4.2). Esses aspectos do agente são chamados de *aspectos de agência.* Cada aspecto de agência é responsável pelo fornecimento do comportamento apropriado para uma propriedade básica do agente. Eles definem as propriedades essenciais do agente em relação à agência: interação, adaptação e autonomia. Esses aspectos afetam o componente Kernel e a si mesmos. Assim, as interfaces crosscutting devem ser definidas para especificar como cada agência afeta os outros componentes.

A Figura 31 descreve os aspectos de agência para a arquitetura de agente no sistema Portalware. Nela, o primeiro compartimento de interfaces crosscutting representa inter-type declarations, e o segundo, pointcuts e os advices acoplados. Muitos métodos e parâmetros de advice foram omitidos para fins de simplificação. A Figura 31 também mostra as classes e os métodos afetados pelos aspectos de agência. As interfaces de interação dos aspectos afetam o componente Kernel. As interfaces especificam as inter-type declarations a fim de adicionar operações específicas à interação à classe Agent. O componente Interação introduz no componente Kernel as operações para receber (interface MessageReception) e enviar mensagens (interface MessageSending). A interface MessageSending contém o pointcut outgoingMsg para definir quando as mensagens precisam ser enviadas a partir de ações e planos de

agente. A interface MessageReception define o pointcut incomingMsg para interceptar o recebimento de mensagens. O aspecto Interation também pode perceber uma interface sensorial que define pointcuts para detectar eventos nos objetos do ambiente (Seção 3.3.2). Essa interface não está representada na Figura 31 uma vez que os agentes do Portalware não precisam desses recursos.

Como os agentes são entidades autônomas, o próprio agente precisa começar suas threads de controle e decidir quando os objetivos precisam ser instanciados (Seção 3.3.4). O componente Autonomia está de acordo com as duas interfaces crosscutting: uma interface GoalCreation e uma interface ExecutionAutonomy. A interface GoalCreation especifica como o componente Autonomia afeta outros componentes para instanciar os objetivos do agente. A interface GoalCreation afeta o componente Interação porque pode ser necessário criar objetivos quando são recebidas mensagens do ambiente externo. Ela também afeta o componente Kernel porque os objetivos podem ter de ser criados sempre que alguma parte do conhecimento do agente for alterada. A interface ExecutionAutonomy descreve como a autonomia de execução afeta o componente Kernel para instanciar as threads do agente. No sistema Portalware, as threads são instanciadas quando um agente é criado. É capturado pelo pointcut newAgent (Figura 31).

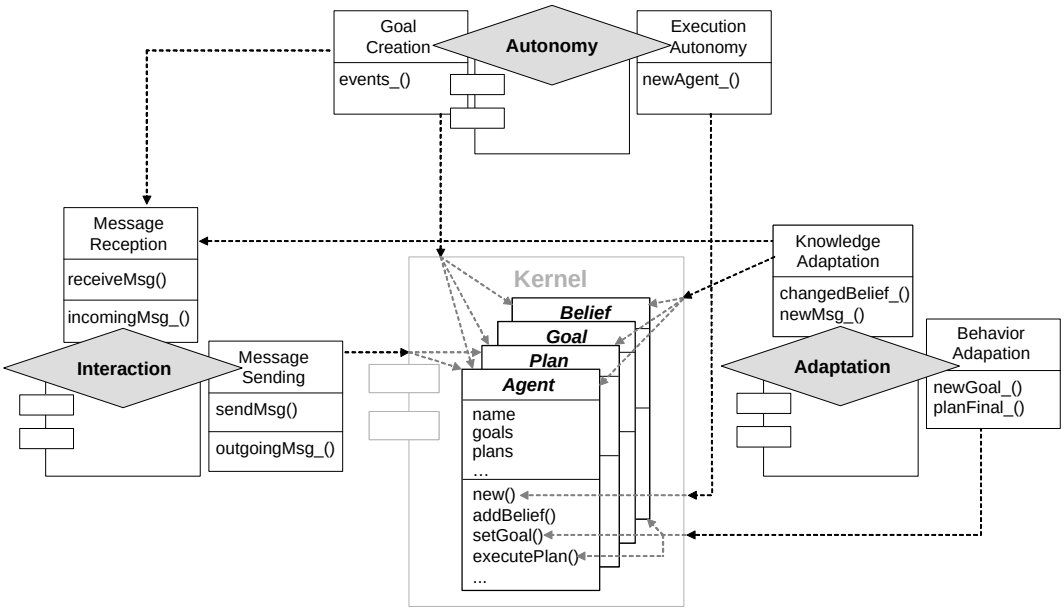

Figura 31. Os aspectos de agência da arquitetura de agente.

O concern de adaptação envolve dois tipos de adaptação: adaptação de conhecimento e adaptação de comportamento (Seção 3.3.3). As interfaces KnowledgeAdaptation e BehaviorAdaptation especificam como o componente Adaptação afeta os outros componentes arquiteturais. O componente Adaptação afeta o componente Interação e o componente Kernel. Eles estão conectados pela interface KnowledgeAdaptation porque, no momento do recebimento de mensagens externas, pode ser necessária a adaptação de conhecimento. A conexão ocorre porque a adaptação de conhecimento é necessária sempre que ocorrem eventos internos, como alteração de crenças. Além disso, o componente Adaptação afeta o componente Kernel pela interface BehaviorAdaptation, uma vez que a seleção de um novo plano se faz necessária sempre que um novo objetivo é definido, e a execução do plano pode ter de ser cancelada devido a alterações em determinadas crenças.

4.4.3
Etapa 3: definição de tipos de agente

Use herança para criar tipos de agente. Os diferentes tipos de agentes são organizados hierarquicamente como subclasses que herdam da classe raiz Agent. Os métodos dessas subclasses implementam as ações de cada tipo de agente. Uma subclasse Agent é parte do componente Kernel da arquitetura de agente. A Figura 32 ilustra as subclasses que representam os tipos de agentes do estudo de caso (Seção 4.1): (i) a classe InterfaceAgent, (ii) a classe InformationAgent e (iii) a classe UserAgent. Por exemplo, o método search(Keyword) da classe InformationAgent implementa o serviço de agentes de informação buscando informações de acordo com a palavra-chave especificada.

4.4.4
Etapa 4: definição das propriedades adicionais

Use aspectos para modularizar as demais propriedades de um tipo de agente. Os aspectos devem ser usados para modularizar as propriedades de mobilidade, aprendizagem e colaboração. Esses aspectos são chamados de *aspectos adicionais*.

Cada tipo de agente normalmente oferece serviços específicos à aplicação e possui diferentes propriedades de agente (Seção 3.2.3). Esses aspectos adicionais específicos a cada tipo de agente são associados às subclasses correspondentes (Figura 33). Assim, o uso de aspectos oferece suporte à definição de arquiteturas heterogêneas para os tipos de agente. Observe que os diferentes tipos de agentes de software herdam os aspectos do agente acoplados à superclasse Agent (Figura 32). Como conseqüência, os três tipos de agente reutilizam os aspectos de agência e somente definem seus aspectos e serviços específicos.

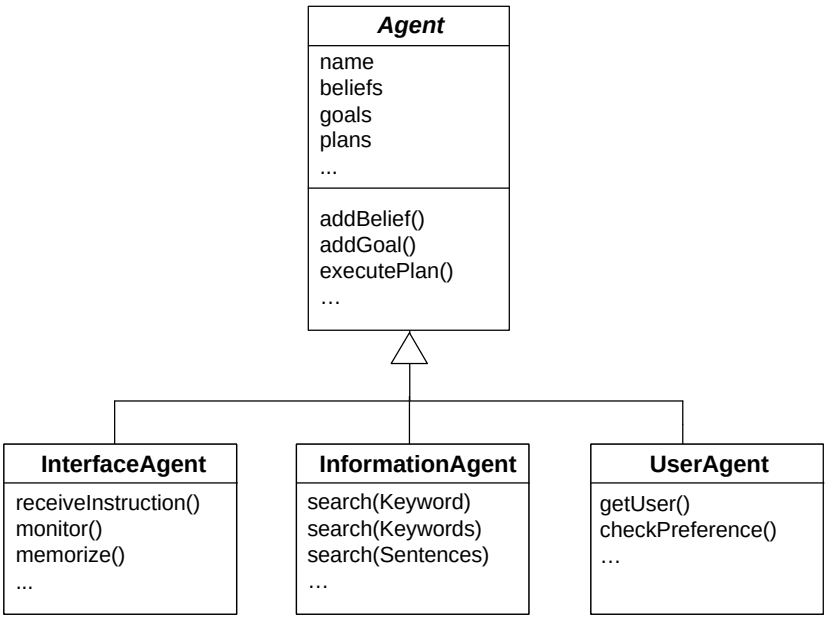

Figura 32. Tipos de agente.

A Figura 33 mostra que o aspecto Collaboration está associado às classes InformationAgent e UserAgent, enquanto o aspecto Learning está acoplado à classe InterfaceAgent. Esse aspecto implementa o protocolo de colaboração, que consiste na sincronização de agentes que participam de colaborações. O CollaborationProtocol bloqueia o envio de mensagens do agente pelo pointcut de bloqueio e desbloqueia quando recebe a resposta pelo pointcut de desbloqueio. O aspecto Learning intercepta os métodos na classe InterfaceAgent a fim de reunir as informações necessárias para o processo de aprendizagem. A interface InformationGathering especifica a lista de métodos que serão interceptados nos pointcuts de evento.

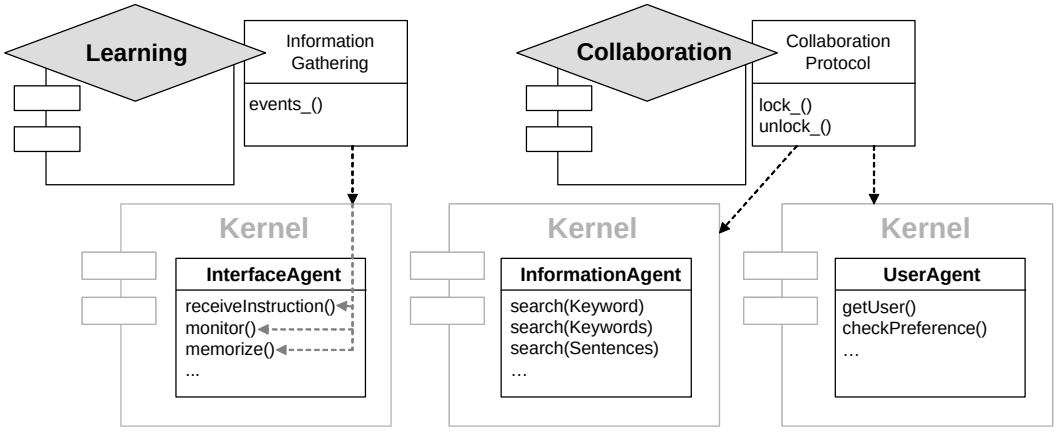

Figura 33. Outros aspectos do agente.

4.4.5
Etapa 5: definição dos papéis do agente

O concern de colaboração envolve os protocolos de colaboração e os papéis do agente, que fazem parte de colaborações (Seção 3.4.1). Assim, a definição do componente Colaboração requer duas etapas. A definição dos aspectos de colaboração que especificam os protocolos de colaboração (*etapa 4*) e a definição de outros componentes para representar os papéis. Os aspectos são usados para implementar os papéis de um agente. Esses aspectos são chamados de *aspectos de papéis*. São uma parte de um agente que define sua atividade dentro de um conjunto de colaborações. Como resultado, ele desacopla os serviços básicos do agente da semântica do papel, que, por sua vez, melhora a compreensão, a reusabilidade e a evolução dos SMAs.

Os aspectos de papéis podem ser associados a várias classes do agente, uma vez que um tipo de agente pode precisar exercer muitos papéis. Os arquitetos de SMAs precisam especificar as interfaces a fim de descrever quando esses aspectos afetam o componente Kernel para conectar o papel ao agente. A Figura 34 ilustra esses aspectos para os agentes de informação do Portalware (Seção 4.1). Um agente de informação precisa exercer os papéis de chamador e respondedor a fim de cooperar com outros agentes de informação em diferentes contextos. Assim, os aspectos de papéis Caller e Answerer são acoplados à classe InformationAgent.

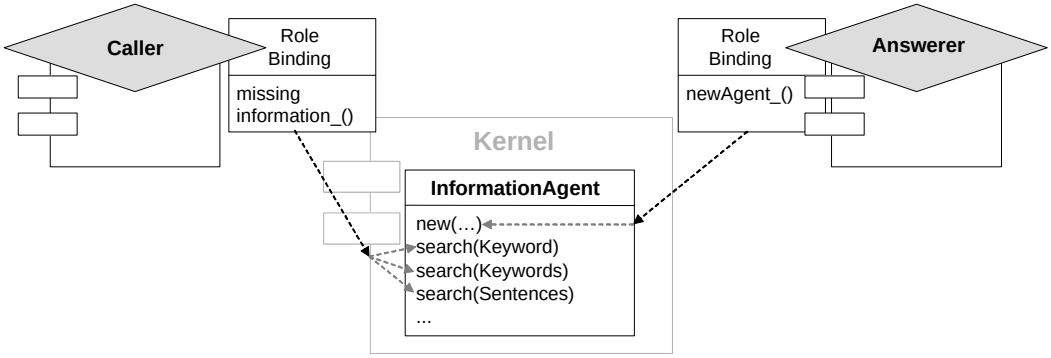

Figura 34. Os aspectos de papéis de agentes de informação.

O aspecto Caller fornece a um agente a capacidade de enviar a solicitação de busca a um agente que esteja respondendo e de receber o resultado da busca. De forma similar, o aspecto Answerer introduz a capacidade de receber a solicitação de busca e de enviar o resultado da busca. Os dois realizam uma interface RoleBinding que especifica quando o papel está ligado às instâncias do agente. O pointcut missingInformation está associado a execuções de métodos de busca e é responsável pelo envio da solicitação de busca quando o próprio agente não é capaz de encontrar as informações necessárias. Esse pointcut está associado ao after advice, que verifica os resultados dos métodos de busca de forma que o papel de chamador seja ativado sempre que o resultado do método seja nulo. Observe que esses papéis são introduzidos de forma transparente e não-intrusiva.

4.4.6
Etapa 6: composição do aspecto

As propriedades de agente não são ortogonais; elas podem ser sobrepostas e interagir entre si (Seção 3.5). Não há necessidade de capturar as características de sobreposição e interatividade dos aspectos do agente na etapa arquitetural. O uso de aspectos arquiteturais oferece suporte à especificação de padrões de relacionamento que incorporam a não-ortogonalidade de propriedades de agente. As etapas anteriores orientaram como os arquitetos de SMAs compõem: aspectos de agência com o kernel do agente, outros aspectos com tipos de agente e aspectos de função com tipos de agente.

No entanto, como a complexidade do agente aumenta, é necessário tratar de outras composições entre concerns do agente: (i) um aspecto de agência pode afetar um outro aspecto e (ii) um outro aspecto ou um aspecto de agência pode afetar aspectos de papéis. Para especificar essas composições de aspectos, os arquitetos de SMAs precisam definir novas interfaces crosscuting ou simplesmente novos pointcuts. Por exemplo, o aspecto Collaboration também afeta os aspectos de Answerer e Caller no sistema Portalware. Dessa forma, os métodos desses dois últimos aspectos são especificados em novos pointcuts da interface CollaborationProtocol. Além disso, a interface InformationGathering do aspecto Learning afeta os aspectos de papéis para que o agente possa aprender em contextos colaborativos.

A Figura 28 ilustra outras composições interaspecto na arquitetura de agentes de informação. Por exemplo, a interface MessageSending também afeta o componente Collaboration uma vez que as mensagens têm de ser enviadas a partir de métodos nos aspectos de papéis. A interface BehaviorAdaptation afeta a interface Traveling porque é necessário adaptar os planos do agente quanto ele está se deslocando para um ambiente remoto.

A Figura 35 mostra o diagrama de interações para um cenário de chamadas básico entre dois agentes de informação no Portalware. O diagrama ilustra como o componente Kernel e os componentes aspectuais trabalham juntos para implementar o comportamento do agente. Um agente de informação recebe uma mensagem para buscar informações de acordo com uma determinada palavra-chave. O aspecto Interaction recebe essa mensagem por um sensor e atualiza a Caixa de Entrada com a nova mensagem (1). O aspecto Autonomy direciona o fluxo de controle normal para seu advice quando a mensagem é recebida (2). Esse aspecto realiza o processo de tomada de decisão ao chamar o plano de decisão (3). Assim, para cada mensagem recebida pelo agente, pode ser determinado, com base nos objetivos e crenças e no estado da conversa, se a mensagem deve ser processada ou não. Depois que o aspecto Autonomy toma essa decisão, ele instancia um objetivo do agente, porque o agente deve conseguir alcançar um novo objetivo de busca (4). Quando o objetivo é definido, o aspecto Adaptation busca um plano de agente apropriado para alcançar

esse objetivo (5) e adapta o comportamento dando prosseguimento ao plano de busca (6). O aspecto Adaptation seleciona um plano na configuração de conhecimento.

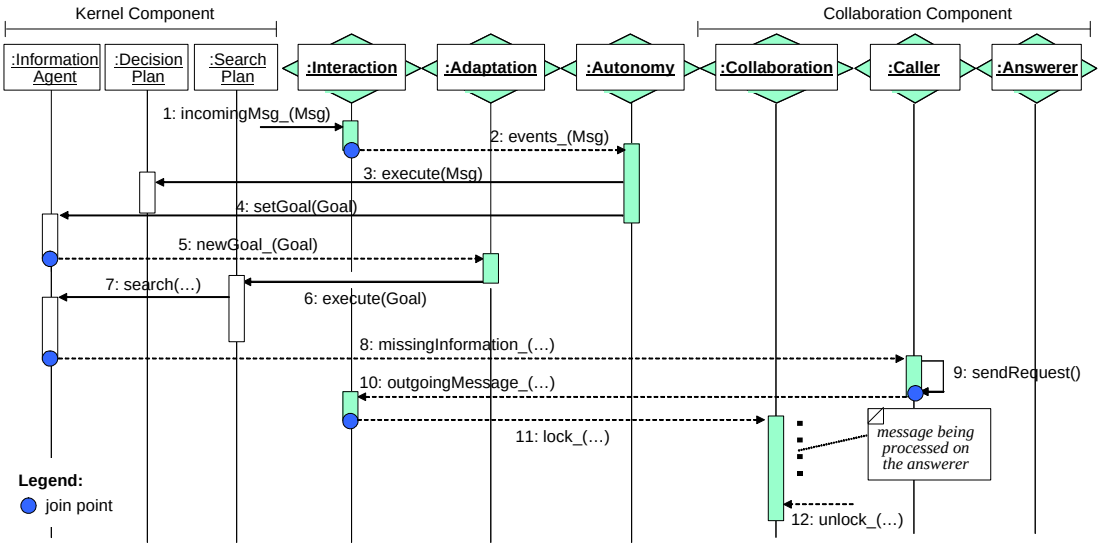

Figura 35. Um diagrama de interação para os agentes de informação do Portalware.

Durante a execução do plano, o serviço de busca do agente é fornecido pela chamada do método search() (7). Como o agente de informação não consegue encontrar a palavra-chave especificada, o aspecto Caller é ativado a fim de exercer o papel de chamador (8). Ele chama o outro agente de informação e pede as informações necessárias (9). O aspecto Interaction monta a mensagem e a envia ao agente de informação do respondedor (10). Depois que a mensagem é enviada, o aspecto Collaboration realiza o protocolo de coordenação de forma que o agente chamador aguarde o resultado da busca (11). De forma similar, o agente que recebe a mensagem exerce o papel de respondedor para receber a solicitação e enviar o resultado da busca. É importante ressaltar que os advices e os métodos de aspectos dependentes são realizados sempre depois dos aspectos dos quais dependem. O papel de chamador é desbloqueado assim que recebe a resposta do outro agente de informação (12).

4.4.7
Etapa 7: evolução de agente

As arquiteturas de agente podem se desenvolver para incorporar novas propriedades de agente e atender aos requisitos de novas aplicações. Os arquitetos de SMAs podem adicionar pointcuts ou interfaces crosscutting a fim de adicionar novas propriedades de agente. Eles não alteram o componente Kernel ou outros componentes arquiteturais. Por outro lado, podem remover os pointcuts ou as interfaces crosscutting a fim de remover as propriedades de agente. A inclusão ou remoção de objetivos, crenças, ações, planos e serviços estão localizadas no componente Kernel.

No contexto do Portalware, vamos supor que os agentes de informação não precisem mais cooperar entre si para encontrar as informações. Em vez disso, os agentes de informação são agora necessários para se transportarem de um ambiente na rede para outro a fim de alcançar o objetivo da busca. Como conseqüência, eles não precisam ter os papéis de chamador e recebedor, mas precisam ser móveis.

Em uma arquitetura orientada a aspectos, essa modificação é normalmente transparente, pois os aspectos do agente podem ser adicionados ou removidos de classes de forma *plug-and-play*. Os aspectos Caller e Answerer são desacoplados da classe InformationAgent sem precisar de qualquer adaptação invasiva para outros componentes de agente (Figura 36). O comportamento restante do agente é mantido inalterado. No entanto, é necessário associar o aspecto Mobility à classe InformationAgent. O aspecto Mobility introduz a essa classe a capacidade de migrar a rede e reunir as informações em nome do proprietário. Esse processo de associação inclui a definição de uma interface Traveling, que especifica alguns pointcuts. Por exemplo, declara o pointcut em movimento que captura o fim das execuções dos métodos de busca (search(*)) como um pointcut. No tempo de execução, quando a execução de um método search() é concluída, o fluxo de controle do programa é desviado do aspecto Mobility. Ele pega os resultados dos métodos de busca para verificar se o agente de informação conseguiu encontrar as informações. Dessa forma, se o resultado do método for nulo, esse aspecto é responsável pela migração do agente até outro *host* a fim de iniciar uma nova busca pelas informações.

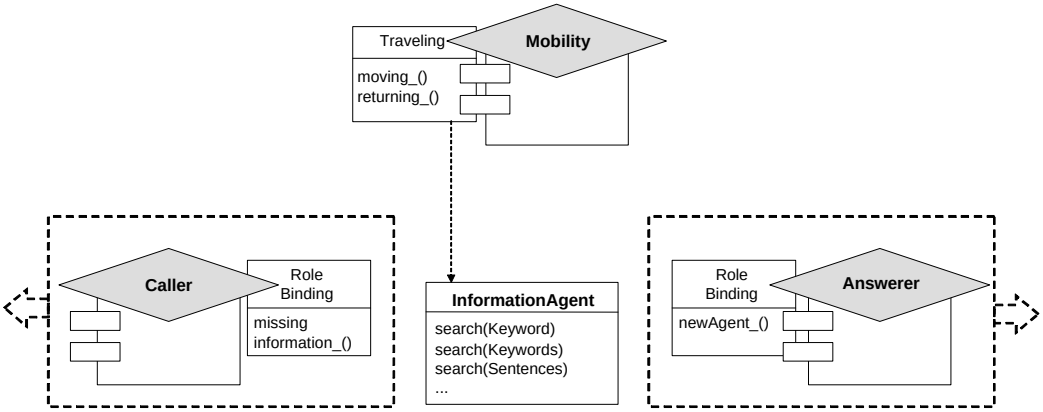

Figura 36. Introdução do aspecto Mobility a agentes de informação.

4.5
Questões de implementação

Uma arquitetura de agente orientada a aspectos foi implementada para o estudo de caso (Seção 4.1) usando AspectJ [140]. O Apêndice I discute questões importantes para a implementação de uma arquitetura de agente orientada a aspectos. AspectJ foi usado no desenvolvimento do Portalware porque é mais simples e mais integrado ao ambiente Java do que a linguagem Hyper/J [235] (Seção 7.3). Pode ser encontrada uma comparação mais detalhada entre essas linguagens [42, 266] na Seção 7.3 e no Apêndice II, que apresenta suas similaridades e diferenças.

A implementação dos agentes do Portalware consiste em 91 classes e 10 aspectos do agente. Alguns aspectos interagem entre si ao interceptar os mesmos join points (Seção 4.4.6). Em alguns casos, a ordem de execução é importante. Por exemplo, a interface KnowledgeAdaptation (aspecto Adaptation) e a interface GoalCreation (aspecto Autonomy) interceptam a recepção de mensagens, ou seja a interface IncomingMessage (aspecto Interaction). Nesse caso, há uma restrição na ordem: o aspecto Adaptation precisa ser executado antes do aspecto Autonomy. O construto dominates da linguagem AspectJ (versão 0.8) foi usado a fim de implementar essa relação de dependência entre esses aspectos. Esse construto indica que a ordem dos aspectos relacionados é importante e oferece suporte à descrição de quaisquer restrições de ordem. O construto dominates não tem suporte na versão atual

do AspectJ [12]; os projetistas de linguagem o substituíram por um construto similar, o declare precedence [12].

A implementação das arquiteturas de agente também usou TSpaces/IBM [155], uma arquitetura blackboard para a comunicação de rede com recursos de banco de dados. TSpaces oferece serviços de comunicação de grupo, serviços de banco de dados e serviços de notificação de eventos. É implementado na linguagem de programação Java e, assim, possui a ubiqüidade de rede por meio de independência de plataforma. As mensagens foram implementadas como tuplas e, finalmente, os agentes se comunicaram colocando as tuplas no blackboard. O único problema relacionado ao uso do TSpaces e AspectJ juntos é que os aspectos Interaction afetam as classes do TSpaces; então, os arquivos-fonte do TSpaces foram necessários durante o processo de combinação (Seção 2.2.2). A versão atual do AspectJ não requer os arquivos-fonte uma vez que oferece suporte ao processo de combinação de bytecode.

No sistema Portalware, as diferentes instâncias da classe InformationAgent devem ter características variantes. As instâncias InformationAgent podem ser colaborativas ou não-colaborativas; estáticas ou móveis, e podem aprender ou não. Portanto, é recomendável construir agentes de informação personalizados para acoplar dinamicamente os aspectos a diferentes instâncias de agentes de informação. A arquitetura proposta oferece suporte a esse recurso, uma vez que os aspectos arquiteturais podem ser acoplados a instâncias individuais. No entanto, AspectJ não oferece suporte a esse recurso porque o processo de combinação é estático. Isso foi possível na versão 0.7 da linguagem, ao usar o construto addObject(); entretanto, ele foi removido da linguagem. Atualmente, há muitos combinadores (*weavers*) dinâmicos (Seção 2.2.2) que oferecem suporte ao acoplamento de aspectos a objetos. *Combinador* é o mecanismo responsável pela composição de classes e aspectos.

Apesar de termos estabelecidos uma arquitetura interna para os agentes, cujas bases estão, de certa forma, no formalismo BDI [197, 198], diferentes estruturas também podem ser usadas, como aquelas definidas em frameworks de implementação específicos [20, 181, 232]. O propósito deste capítulo é ressaltar um método de projeto mais genérico em vez se enfatizar as abordagens de princípios específicos

[22]. Como conseqüência, os frameworks de implementação existentes podem ser usados em conjunto com arquiteturas de agente orientadas a aspectos a fim de produzir diferentes arquiteturas de agente.

4.6 Discussão e trabalhos relacionados

Ao lidar com vários concerns adicionais e de agência, como adaptação e aprendizagem, na fase da arquitetura, a definição foi reconhecida como um problema sério que não recebeu a devida atenção [91, 189, 190]. De fato, os trabalhos relacionados nessa área têm sido escassos, não tendo havido tentativas em que os concerns do agente fossem considerados dentro da etapa arquitetural [91]. As pesquisas na engenharia de software orientada a agentes concentraram-se em metodologias de alto nível e linguagens de modelagem [56, 91, 122, 131]. A Seção 3.7 apresenta as abordagens arquiteturais existentes para a separação de concerns do agente.

Os arquitetos de software desejam separar concerns da aplicação em componentes separados, mas os estilos arquiteturais existentes (incluindo o estilo baseado em mediadores) não são capazes de realizar essa separação em sistemas multiagentes. A arquitetura de agente proposta é diferente de uma arquitetura baseada em mediadores porque a composição de concerns do agente não está centralizada em um único componente, o mediador (Seção 3.7.3). Cada aspecto arquitetural especifica como isso afeta os demais componentes arquiteturais. A arquitetura proposta não é reflexiva (Seção 3.7.2) porque os aspectos arquiteturais, diferente dos metaobjetos, não se limitam ao acoplamento a um único objeto. Além disso, os aspectos arquiteturais podem alterar a interface de outros componentes com a introdução de novos campos e métodos (Seção 2.2.2). Esse recurso de crosscutting estático também distingue a arquitetura orientada a aspectos das arquiteturas orientadas por eventos [35] e as arquiteturas blackboard [35].

Finalmente, a arquitetura de agente orientada a aspectos também é diferente das arquiteturas de agente em camadas (Seção 3.7.1) definida na abordagem de Kendall [137]. Os aspectos arquiteturais não são estruturados em camadas; cada

componente arquitetural pode ser associado a mais de dois componentes. No entendimento do autor, o trabalho de Kendall é a única proposta que oferece suporte a um método arquitetural para estruturar concerns do agente. Sua abordagem baseia-se no padrão arquitetural Agente em Camadas [137], que separa os concerns do agente usando camadas (Seção 3.7.1). Kendall propõe um método para estruturar cada concern do agente baseado na abstração de camadas. Contudo, há poucas provas relativas à reusabilidade e à manutenibilidade de arquiteturas de agente em camadas em termos de avaliações quantitativas. O Capítulo 8 oferece uma avaliação quantitativa da arquitetura de agente orientada a agentes proposta neste capítulo. Além disso, a Seção 3.7.1 apresentou algumas limitações da abordagem de Kendall, como a esquizofrenia de agente. A arquitetura orientada a aspectos proposta minimiza esse problema uma vez que o agente é representado por uma única instância de Agente no sistema (Seção 4.4.1).

O método proposto descreve um conjunto de decisões arquiteturais, que contribuem para melhorar a manutenibilidade de SMAs. A segregação alcançada limita significativamente o impacto de uma alteração porque os componentes arquiteturais modularizam os crosscutting concerns do agente. Diferente do uso de arquiteturas de agente baseadas em mediadores (Seção 4.2), o uso de arquiteturas orientadas a aspectos oferece suporte ao encapsulamento funcional da funcionalidade básica do agente porque o componente Kernel não está entrelaçado com as propriedades de agente. As interfaces crosscutting permitem a adição das propriedades adicionais e agência à funcionalidade básica de forma não-intrusiva. Como conseqüência, as arquiteturas dos objetos existentes podem ser transformadas em arquiteturas de agente sem qualquer alteração nos métodos.

A arquitetura orientada a aspectos proposta neste capítulo também aumenta as chances de reusabilidade dos componentes do agente. As aplicações que adotam essa arquitetura podem reutilizar e refinar os componentes arquiteturais de forma mais modular uma vez que os crosscutting concerns do agente (Seção 4.2) são encapsulados em aspectos. Os concerns do agente não estão espalhados e entrelaçados entre si como na solução baseada em mediadores. A melhor separação de concerns facilita também a construção de arquiteturas de agentes heterogêneos,

conforme ilustrado nas Seções 4.4.6 e 4.4.7. Cada componente arquitetural é transparente na forma como é modificado por aspectos do agente. Não há referência no componente Kernel a aspectos do agente. Como conseqüência, é mais fácil adicionar ou remover aspectos do agente da arquitetura de agente.

4.7 Resumo

Este capítulo apresentou um método arquitetural cujo objetivo é ser simples o suficiente para ser usado no desenvolvimento de arquiteturas de agente com manutenibilidade e reusabilidade em diferentes tipos de aplicações orientadas a agentes. Os requisitos de qualidade de um sistema de software são amplamente permitidos ou restringidos por sua arquitetura. Se uma arquitetura apropriada que oferece suporte à separação de concerns do agente for escolhida desde o início da fase de projeto, são grandes as chances de obter uma implementação e um projeto adequado ao longo de todo o desenvolvimento do software. O objetivo do método é a geração de arquiteturas orientadas a aspectos, que são descrições de alto nível da estrutura interna do agente em termos dos componentes arquiteturais e seus relacionamentos. A arquitetura orientada a aspectos proposta oferece suporte: (i) à modularização de crosscutting concerns do agente, (ii) à integração de concerns do agente e (iii) à independência de linguagens de programação ou frameworks de implementação de SMAs. Como o método orientado a aspectos é independente da linguagem de programação ou do framework de implementação, muitos desenvolvedores de aplicação poderão empregá-lo.

O uso da arquitetura orientada a aspectos pode minimizar a complexidade causada pela natureza crosscutting das propriedades de agente (Seção 3.6.4). Ela propõe o uso de aspectos para fornecer uma clara separação de concerns entre a funcionalidade básica do agente e as propriedades do mesmo. A separação explícita permite a definição de uma ordem no tratamento dos concerns do agente. O método arquitetural proposto tem o potencial de levar as propriedades de agente aos sistemas de software complexos de vários domínios, como o comércio eletrônico e as telecomunicações, devido a sua simplicidade e facilidade de implementação. Além do

mais, as arquiteturas orientadas a aspectos permitem que as propriedades de agentes sejam incorporadas a um sistema orientado a objetos quando os desenvolvedores desejam transformar objetos em agentes. Os protótipos da arquitetura de agente orientada a aspectos proposta foram desenvolvidos usando a linguagem de programação AspectJ (consulte o Capítulo 7 e o Apêndice I).

O método arquitetural oferece regras de alto nível para estruturar a arquitetura de agente e a ordem em que os concerns do agente serão tratados. Os aspectos arquiteturais precisam ser redefinidos para produzir o projeto detalhado e a implementação dos agentes de software. Nesse contexto, surgem algumas perguntas, como: (i) como especificar a estrutura interna dos componentes aspectuais, (ii) como especificar os advices e (iii) como definir várias interfaces crosscutting para o mesmo componente aspectual. O próximo capítulo apresenta os padrões de projeto propostos a fim de refinar os componentes gerais de uma arquitetura orientada a aspectos. Os padrões seguem as diretrizes arquiteturais apresentadas neste capítulo.