1
Introdução

"To my taste the main characteristic of intelligent thinking is that one is willing and able to study in depth an aspect of one's subject matter in isolation, for the sake of its own consistency, all the time knowing that one is occupying oneself with only one of the aspects." (Dijkstra, 1976)

Com os avanços das aplicações de software baseadas em rede, a introdução de componentes autônomos nos sistemas de software tornou-se um requisito comum em muitas aplicações [91, 93, 128, 164, 248]. Essa necessidade estimulou a revitalização da tecnologia de agentes como um complemento ao paradigma de objetos em uma grande variedade de domínios de aplicação, incluindo o comércio eletrônico [113], ambientes de desenvolvimento de software [89] e assistentes pessoais [166, 215]. Exatamente como os objetos, os agentes também oferecem serviços aos clientes, mas são reconhecidamente diferentes deles, do ponto de vista da engenharia de software [8, 56, 128, 217, 248, 262].

Diferente de objetos, agentes são entidades autônomas que tomam a iniciativa de representar usuários de software [33, 183]. Um desafio importante é construir agentes com base no paradigma de objetos [8, 91, 183, 217]. A arquitetura de agentes inclui concerns básicos, como os serviços de agente disponíveis para os clientes. No entanto, ela também compreende um conjunto de concerns adicionais que aumentam a complexidade do software. Os engenheiros de software de sistemas multiagentes (SMAs) têm de lidar com muitos concerns adicionais, além dos básicos de cada agente de software. A arquitetura interna de um único agente incorpora muitos concerns, inclusive interação, adaptação, autonomia, conhecimento, colaboração, papéis, aprendizagem e mobilidade. Como conseqüência, os engenheiros de SMAs dedicam-se a estes problemas: como fazer um agente interagir apropriadamente, como lidar com seu comportamento adaptativo, com a estruturação de seu comportamento autônomo e a incorporação dos mecanismos de aprendizagem à sua estrutura.

Como resultado, a *separação de concerns* é o centro do desenvolvimento de SMAs [8, 137, 189, 190, 240]. A reutilização e a manutenção de SMAs dependem muito da capacidade de os métodos e as abstrações da engenharia de software de oferecem suporte à separação explícita dos concerns específicos a agentes [189, 190]. Os métodos e as abstrações devem permitir a modularização de cada concern do agente de forma que a segregação alcançada limite significativamente o impacto de uma mudança e aumente as chances de reusabilidade em outros desenvolvimentos de SMAs. Essa separação precisa ocorrer em todas as fases de desenvolvimento, em especial da fase arquitetural até a fase de implementação. A separação arquitetural será perdida se as abstrações da implementação não conseguirem preservá-la e vice-versa.

1.1 O problema

Apesar da separação de concerns ser crítica para os engenheiros de SMAs, normalmente é muito difícil alcançá-la em sistemas realísticos. Os sistemas multiagentes incorporam tipos heterogêneos de agentes [7, 8, 137, 182]. A arquitetura interna dos diferentes tipos de agentes varia muito, uma vez que incorporam propriedades diferentes [8, 112, 137, 182]. Cada propriedade interage com a funcionalidade básica do agente e, normalmente, com outras propriedades do mesmo. Esses concerns são compostos de formas significativamente diferentes que dependem dos tipos de agentes [7, 8, 173]. Como conseqüência, precisamos de uma abordagem arquitetural flexível o suficiente para oferecer suporte à construção de arquiteturas de agentes heterogêneos.

Além disso, a transição de uma especificação arquitetural para um projeto detalhado orientado a objetos não ocorre de forma direta. O uso de objetos para o projeto dos agentes introduz naturalmente falhas devido às diferenças conceituais entre objetos e agentes [7, 8]. Primeiro, a noção de herança [233] não oferece suporte à construção flexível de tipos de agentes heterogêneos. Isso normalmente resulta em grandes árvores de herança com replicação de código e uma explosão no número de classes [7, 8].

É difícil promover a separação dos concerns do agente somente com base em métodos e abstrações orientadas a objetos. O projeto e a implementação dos concerns do agente tendem a afetar muitas classes do código e do projeto do sistema, incluindo aquelas que representam a funcionalidade básica do agente [189, 190]. O uso da orientação a objetos leva a programas que são um entrelaçamento confuso de linhas de códigos de diferentes concerns [189] (Figura 1).

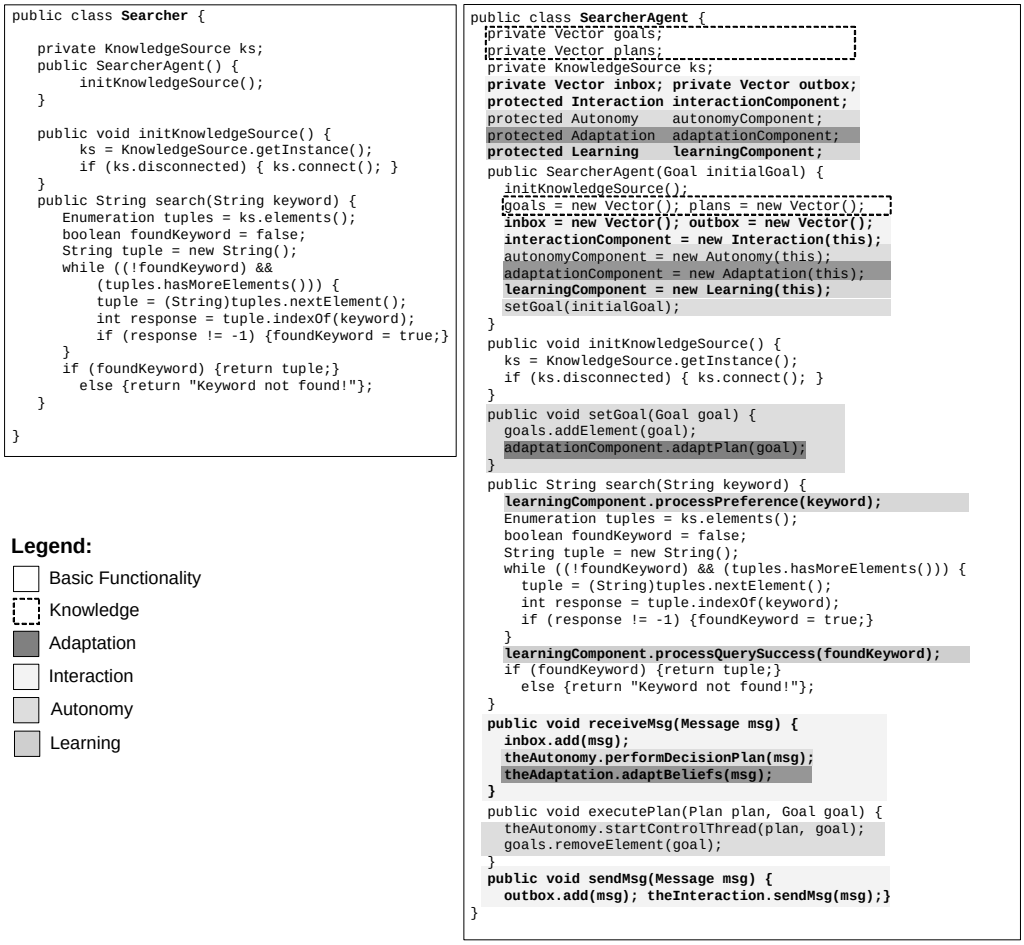

Figura 1. Entrelaçamento de códigos em aplicações orientadas a agentes.

A Figura 1 captura o problema do entrelaçamento de códigos em sistemas multiagentes. À esquerda, há o código para uma classe como uma entidade não-agente. À direita, há uma classe equivalente com o código para a implementação de concerns específicos a agentes. A primeira observação é que o código à direita é muito maior: um fato nada surpreendente, uma vez que um agente é inerentemente mais complexo do que um objeto simples. Contudo, depois de uma análise cuidadosa do código à direita, concluímos que esse código: (i) perdeu o encapsulamento

funcional da versão não-agente, (ii) é uma mistura confusa de linhas de código para concerns diferentes e (iii) está repleto de informações redundantes.

O que acontece, então, é que muitos concerns do agente são perdidos através das classes do sistema, diminuindo a manutenibilidade e a reusabilidade do sistema [190]. Esses problemas também dificultam a transformação de classes existentes em agentes autônomos. O uso de abstrações orientadas a objetos requer alterações invasivas à funcionalidade básica das classes. A implementação de propriedades específicas a agentes prejudica a modularidade do comportamento não-agente. A falta de modularização introduz problemas indesejáveis, como maior acoplamento e menos coesão nos componentes do sistema. Apesar desses problemas, os desenvolvedores de SMAs confiam mais em técnicas de projetos orientados a objetos e em linguagens de programação orientadas a objetos, como Java.

1.2 Limitações de trabalhos relacionados

A engenharia de software baseada em agentes já foi estudada a partir de diferentes perspectivas, inclusive linguagens e metodologias orientadas a agentes, para fases de desenvolvimento de nível superior [66, 184, 249], modelagem conceitual [157, 223, 244] e frameworks de implementação [20, 28, 52, 137]. Os frameworks de implementação oferecem APIs orientados a objetos para o desenvolvimento de SMAs, sem fornecer orientações para a modularização dos concerns do agente. Além disso, muitos métodos propostos são de alto nível e não indicam como dominar a complexidade desses concerns com base em abstrações orientadas a objetos. Como conseqüência, os SMAs foram projetados e implementados de forma *ad hoc*.

Embora a separação de concerns seja amplamente reconhecida como crucial para o desenvolvimento de SMAs com manutenibilidade e reusabilidade [8, 112, 137, 240], há poucas abordagens que oferecem suporte à separação das propriedades dos agentes. Alguns pesquisadores reconheceram esses problemas em alguns concerns de um único agente, como papéis [136] e mobilidade [240], e propuseram técnicas para lidar com elas. Entretanto, essas técnicas só servem para separar esses concerns

individuais. Além disso, a maioria delas se concentra apenas na etapa de implementação e não fornece uma separação explícita dos concerns no nível do código.

Até agora, foram publicados poucos trabalhos sobre a definição de um método de desenvolvimento para estruturar os concerns do agente em sistemas de software nas etapas preliminares do projeto. Foram propostas algumas abordagens arquiteturais para separar e integrar concerns do agente. Essas propostas se baseiam nos padrões tradicionais de arquitetura, como o padrão Reflexão [8, 112], o padrão Camadas [137] e o padrão Mediador [37]. Essas soluções usam abstrações arquiteturais, por exemplo *camadas* e *metaobjetos*, como componentes para isolar concerns do agente. Todavia, essas soluções impõem conexões rígidas nos componentes arquiteturais, o que impossibilita a construção de tipos de agentes heterogêneos.

Além disso, as linguagens de programação orientadas a objetos não oferecem suporte direto a essas abstrações. Elas não estão alinhadas com os mecanismos de composição e decomposição do paradigma de objetos. Então, a transição de modelos arquiteturais para programação orientada a objetos é incômoda e leva muito tempo. Apesar de os concerns poderem ser de alguma forma separados no nível arquitetural, trazê-los para o código levanta o problema do entrelaçamento de código (Seção 1.1). A separação arquitetural desaparece no projeto e na implementação de agentes. Kendall et al [137] e outros pesquisadores [79, 146, 240] propõem os padrões de projetos orientados a objetos para SMAs, que parecem ser a melhor solução para minimizar os problemas mencionados anteriormente. No entanto, eles fornecem uma separação de concerns apenas limitada e não conseguem resolver os problemas de explosão de classe e replicação de código [115].

1.3 A solução proposta

Este trabalho propõe uma abordagem orientada a aspectos para oferecer suporte à modularização e ao tratamento separado de concerns específicos a SMAs. Os *Aspectos* são usados como abstrações unificadoras para capturar os concerns do

agente que sejam difíceis de modularizar com abstrações arquiteturais existentes e com a engenharia de software orientada a objetos. A abordagem oferece suporte à separação e à composição desses crosscutting concerns, que podem ser pensados individualmente como estando em isolamento ao longo do ciclo de vida de sistemas baseados em agentes. A separação é preservada desde o projeto de arquitetura até a implementação, resultando em artefatos de software desembaraçados e em uma melhor separação de concerns. A abordagem proposta é independente de frameworks de implementação de SMAs, como JADE [20] e ZEUS [181].

O desenvolvimento de softwares orientados a aspectos [139] é um paradigma em evolução para modularizar concerns cujos objetos não são capazes de capturar explicitamente. Ele incentiva descrições modulares de softwares complexos ao fornecer suporte à separação clara da funcionalidade do sistema de seus crosscutting concerns. Contudo, abordagens orientadas a aspectos [21, 139, 160, 234] limitaram-se ao contexto de implementação e projeto orientado a objetos. A abordagem proposta usa abstrações baseadas em aspectos para melhorar a separação de concerns específicos a SMAs ao longo da definição, do projeto e da implementação arquitetural.

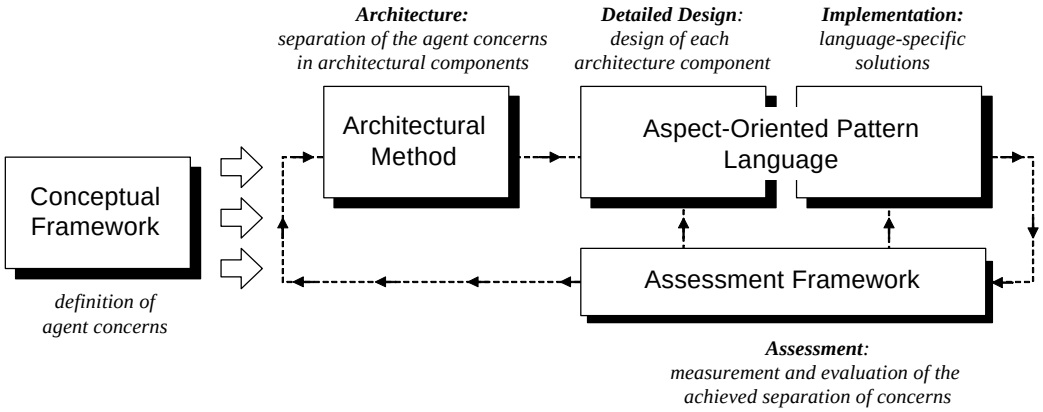

Figura 2. A abordagem orientada a aspectos: uma visão geral.

A Figura 2 apresenta a abordagem orientada a aspectos para oferecer suporte à construção de agentes de software com manutenibilidade e reusabilidade. Essa abordagem oferece suporte à separação de concerns do agente em quatro etapas de desenvolvimento: (i) a etapa de projeto arquitetural, (ii) o refinamento da arquitetura

ou a etapa do projeto detalhado, (iii) a etapa de implementação e (iv) a etapa de avaliação. Cada etapa tem o suporte de um componente da abordagem proposta. O método arquitetural oferece suporte à definição de arquiteturas de agente, e a linguagem padrão auxilia o projeto detalhado e a implementação. O framework de avaliação oferece suporte à avaliação quantitativa da implementação e do projeto de SMAs. A abordagem compreende também um *framework conceitual*, chamado TAO[1] [223]. A função principal do TAO é fornecer as bases para a engenharia de software baseada em agentes. Ele desenvolve conceitos que definem os concerns típicos dos SMAs e especifica seu relacionamento. O TAO oferece a terminologia relacionada a agentes a outros elementos da abordagem.

O *método arquitetural* [35, 80] oferece suporte à especificação de arquiteturas de agentes orientadas a aspectos com separação explícita de concerns. A idéia é incentivar o tratamento individual das propriedades dos agentes e fornecer um esquema disciplinado para sua composição em uma etapa preliminar do projeto. O método apresenta orientações para a separação de concerns que devem ser seguidas durante o projeto e a implementação. Ele define os componentes arquiteturais para lidar com diferentes concerns do agente, determinando quais devem ser modelados como aspectos e quais devem ser modelados como classes. As arquiteturas de agentes obtidas são flexíveis o suficiente para oferecer suporte a diferentes composições das propriedades dos agentes para tipos de agentes heterogêneos. O método prescreve não só como desenvolver arquiteturas de agentes desde o esboço, mas também como transformar a arquitetura dos objetos existentes em arquiteturas de agentes, de forma não intrusiva.

O método de arquitetura proposto oferece logo no início do ciclo de vida o contexto no qual podem ser tomadas decisões do projeto mais detalhadas, relacionadas a concerns do agente, em vez de serem tomadas durante as etapas de implementação e projeto detalhado. Nesse contexto, a *linguagem de padrões* promove a separação de concerns no nível do projeto detalhado, oferecendo suporte ao refinamento das arquiteturas de agentes orientadas a aspectos. Os padrões do projeto fornecem aos engenheiros de software soluções orientadas a aspectos para a

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

construção, de forma disciplinada, de SMAs com reusabilidade e manutenibilidade. Essas soluções concentram-se no uso de aspectos para modularizar os concerns do agente que são naturalmente combinados na implementação e no projeto orientado a objetos. As classes só lidam com a funcionalidade básica dos agentes. Cada padrão cuida de um concern do agente específico e determina como refinar o componente arquitetural correspondente definido anteriormente. A linguagem de padrões também descreve como conectar os padrões do projeto a fim de desenvolver uma solução geral. Os padrões podem ser mapeados diretamente para elementos da implementação. Os padrões do projeto propostos foram implementados em AspectJ [140], uma extensão orientada a aspectos contínua à linguagem de programação Java.

O *framework de avaliação* [210] oferece suporte à avaliação quantitativa da implementação e do projeto orientado a aspectos. Ele é composto por um modelo de qualidade e um conjunto de métricas. O modelo de qualidade oferece definições operacionais para as qualidades que devem ser medidas com as métricas. As métricas são mecanismos que ajudam a entender a reusabilidade e a manutenibilidade dos artefatos de software orientados a aspectos. O modelo de qualidade e as métricas são encontrados em atributos de engenharia de software bem conhecidos, como acoplamento, coesão e separação de concerns.

O framework de avaliação tem como objetivo três contextos de desenvolvimento: (i) *contexto de comparação* – ou seja, a comparação entre produtos existentes orientados a aspectos e orientados a objetos, (ii) *contexto de desenvolvimento iterativo* – ao desenvolver o sistema incrementalmente, pode ser interessante usar métricas a fim de analisar o impacto de soluções alternativas orientadas a objetos e orientadas a aspectos para os recursos que serão introduzidos e (iii) *contexto de reengenharia orientada a aspectos* – um sistema pode exibir crosscutting concerns à medida que o sistema se desenvolve ao longo do tempo; as métricas podem ajudar os engenheiros de software a encontrar os crosscutting concerns e as oportunidades para refatorações localizadas.

Cada componente da abordagem proposta é visto como uma contribuição original. As contribuições representam o primeiro esforço no suporte à separação dos concerns do agente desde a etapa arquitetural até as etapas de implementação e

avaliação. A originalidade de cada componente na abordagem será discutida no final do capítulo que o descreve. Essas contribuições foram publicadas em dois periódicos [82, 95], três capítulos de livro [98, 221, 223], três trabalhos apresentados em simpósios [80, 92, 210] e diversos trabalhos apresentados em workshops internacionais [42, 81, 84, 89, 90, 147, 219, 220, 259].

1.4 Avaliação empírica

Foi realizada uma avaliação sistemática para avaliar a abordagem orientada a aspectos proposta e para demonstrar sua utilidade, benefícios e obstáculos em relação a critérios quantitativos e qualitativos relevantes. Três sistemas representativos de diferentes domínios de aplicação e com diferentes graus de complexidade foram selecionados como objetos da avaliação. Os sistemas incorporam vários tipos de agentes e diferentes concerns do agente combinados de formas distintas. A avaliação foi conduzida com base em dois conjuntos de estudos de caso: estudos qualitativos e um estudo quantitativo. O primeiro conjunto de estudos de caso foi realizado para avaliar os benefícios potenciais e os possíveis obstáculos do uso de uma abordagem orientada a aspectos. Foi investigada a adeqüabilidade da abordagem para oferecer suporte à separação dos concerns dos SMAs usando como base critérios qualitativos [82]. Além disso, a abordagem foi qualitativamente comparada à abordagem orientada a objetos com base em padrões de projeto [82].

O estudo quantitativo foi planejado como um experimento mais controlado [98] usando o método GQM de Basili [16] e o framework de avaliação proposto [210]. Esse estudo foi realizado a fim de confirmar ou refutar os resultados dos estudos qualitativos. Foram considerados muitos procedimentos para minimizar os problemas normalmente encontrados nos estudos empíricos [17]. O uso das soluções propostas resultou em: (i) menos linhas de códigos, (ii) menos componentes de implementação e projeto (iii) menor acoplamento entre os componentes e (iv) componentes de sistema menos complexos internamente.

1.5 Descrição da tese

Este texto está organizado da seguinte forma. O Capítulo 2 apresenta as definições e as técnicas associadas à separação de concerns. O Capítulo 3 introduz o framework conceitual que descreve e define os concerns do agente. Esse framework é usado para descrever e caracterizar métodos e arquiteturas orientadas a agentes e, em especial, foi usado também para caracterizar e direcionar a abordagem orientada a aspectos proposta para o desenvolvimento de sistemas multiagentes. Esse capítulo também ilustra os problemas e motiva a necessidade de aspectos para cuidar dos crosscutting concerns do agente, desde a etapa de projeto preliminar.

O Capítulo 4 apresenta o método arquitetural proposto. Ele fornece orientações para construir arquiteturas de agente orientadas a aspectos que melhoram a modularização de concerns do agente. O Capítulo 5 descreve a linguagem de padrões de projeto proposta. Cada padrão é apresentado em detalhes. O Capítulo 6 apresenta o framework de avaliação e descreve o modelo de qualidade e o conjunto de métricas.

O Capítulo 7 apresenta quatro estudos de caso qualitativos. O Capítulo 8 apresenta o estudo de caso quantitativo, que usou o framework de avaliação. O Capítulo 9 apresenta as conclusões que relacionam as contribuições e premissas desta tese. Esse capítulo também discute o trabalho que está sendo realizado no momento e trabalhos futuros.

O Apêndice A apresenta a implementação dos padrões de projeto orientados a aspectos propostos. O Apêndice B apresenta as regras para transformar programas AspectJ™ em programas Hyper/J™; AspectJ e Hyper/J são as linguagens orientadas a aspectos mais conhecidas. Dessa forma, a abordagem orientada a aspectos proposta é demonstrada como sendo independente dos dois modelos de implementação específicos.