7
Estudos de Caso

Os estudos experimentais [17] são uma maneira eficaz de fornecer comprovações empíricas que podem melhorar a compreensão das técnicas e métodos da engenharia de software. No entanto, há poucos experimentos relacionados à orientação a aspectos na literatura. Este trabalho de pesquisa envolveu cinco estudos de casos a fim de avaliar os componentes da abordagem orientada a aspectos: o método arquitetural (Capítulo 4), a linguagem de padrões (Capítulo 5) e o framework de avaliação (Capítulo 6). Foi realizada uma avaliação sistemática para avaliar a abordagem proposta e para demonstrar sua utilidade, benefícios e obstáculos em termos dos critérios quantitativos e qualitativos.

Nome do sistema	**Domínio**	**Tipos de agente**	**Concerns de agência**[6]							**LOC**
			In	Ad	Au	Co	Pa	Mo	Ap	
TSA	Simulação de tráfego	Reativo	■	■	■					1859
Portalware v.2 (estudo qualitativo)	Ambiente de desenvolvimento	Reativo	■	■	■	■	■			1654
Portalware v.2 (estudo quantitativo)		Cognitivo Reativo	■	■	■	■	■	■	■	3063
Expert Committee	Gerenciamento de conferências	Híbrido	■	■	■		■	■	■	6564

Tabela 4. Características dos sistemas multiagentes

[6] In = Interação, Ad = Adaptação, Au = Autonomia, Co = Colaboração, Pa = Papéis, Mo = Mobilidade, Ap = Aprendizagem.

Os estudos de casos baseiam-se em protótipos de domínios de diferentes aplicações: simulação de tráfego, ambiente de desenvolvimento de portal e gerenciamento de conferências (Tabela 4). Esses estudos envolveram três pequenos sistemas representativos, compostos de agentes reativos, agentes cognitivos e agentes híbridos (Seção 3.2.3). Baseiam-se em aplicações em larga escala existentes, mas escalonadas sem reduzir a complexidade inerente relacionada aos tópicos da pesquisa. A Tabela 4 mostra que cada sistema incorporou diferentes propriedades de agência. Duas linguagens orientadas a aspectos foram usadas para a implementação dos estudos de caso: AspectJ [140] e Hyper/J [235] (Seção 2.2). A Figura 80 apresenta um esquema da evolução histórica dos estudos de caso.

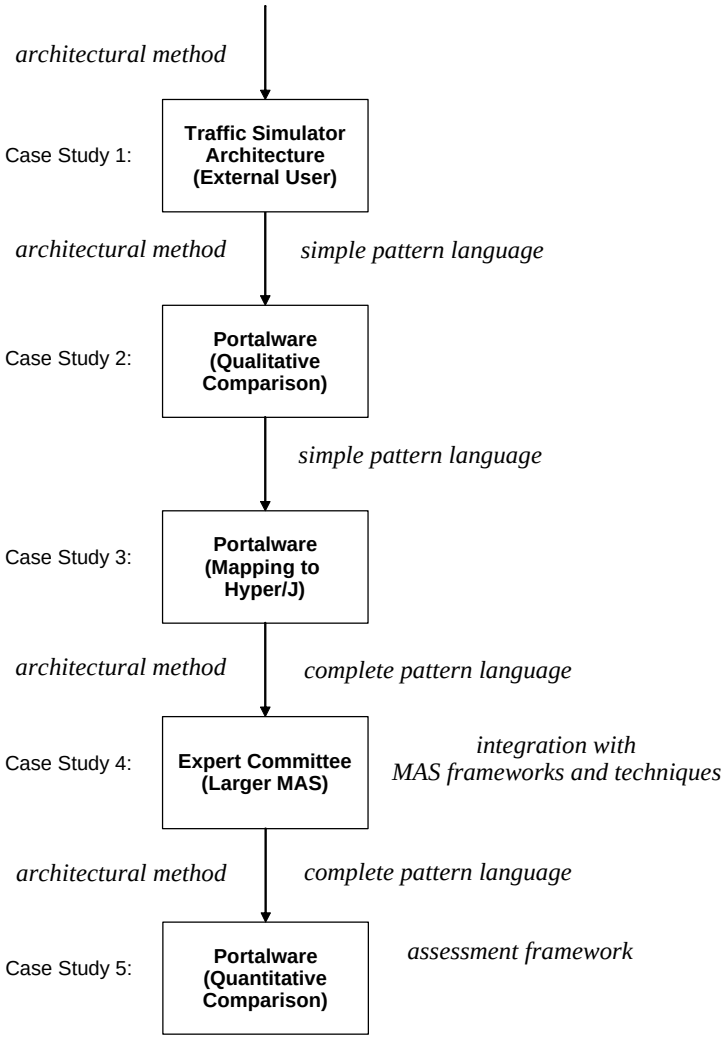

Figura 80. A evolução dos estudos de caso.

A finalidade do primeiro estudo era avaliar o método arquitetural proposto (Capítulo 4). Foi um estudo de caso simples que forneceu o feedback inicial para a abordagem orientada a aspectos no contexto de desenvolvimento fora de nosso ambiente de pesquisa. O segundo estudo de caso foi um estudo de caso comparativo para melhor avaliar o método arquitetural e a linguagem de padrões. Preocupava-se com a comparação qualitativa da abordagem orientada a aspectos com uma abordagem baseada em padrões [82]. Foi um trabalho pioneiro na literatura ao comparar vários padrões de projeto com soluções orientadas a aspectos [82]. Possui até mesmo algumas contribuições importantes anteriores, como a conhecida comparação de Hannemann e Kiczales que envolve os padrões GoF [115].

O terceiro estudo de caso consistiu em um mapeamento da implementação de AspectJ do Portalware [89] em uma implementação de Hyper/J com o objetivo de verificar a generalidade da arquitetura de agentes orientada a aspectos (Capítulo 4) e a linguagem de padrões (Capítulo 5). Foram as primeiras comparações entre as implementações de AspectJ e Hyper/J [42]. O quarto estudo de caso envolveu o sistema Expert Committee (Seção 5.1). É um sistema multiagentes mais complexo em termos das linhas de código (Tabela 4). Também incorporou vários concerns do agente e usou diferentes técnicas e frameworks de implementação de SMAs. É composto de agentes de software para o gerenciamento do envio de trabalhos e dos processos de seleção (Seção 5.1). O desenvolvimento desse sistema aplicou as mais sofisticadas formas de padrões de projeto orientado a aspectos.

Os resultados animadores obtidos nos estudos de caso anteriores motivaram a realização de um experimento mais rigoroso, usando a metodologia GQM de Basili [16] e o framework de avaliação proposto. Nesse sentido, o objetivo do experimento é duplo: (i) fornecer uma primeira avaliação do framework de avaliação e (ii) realizar uma investigação quantitativa da abordagem orientada a aspectos a fim de confirmar ou refutar as descobertas dos estudos qualitativos anteriores. No entendimento do autor, é o primeiro estudo empírico que envolveu uma comparação quantitativa entre aspectos e objetos. O Capítulo 8 apresenta em detalhes esse estudo experimental e seus resultados.

As mais importantes contribuições do capítulo também foram publicadas em trabalhos apresentados em workshops [42, 81, 88, 90] e em um periódico [82]. Os trabalhos apresentados nos workshops descrevem os primeiros estudos e o experimento de mapeamento; já o do periódico descreve o estudo comparativo com base em critérios qualitativos. O restante deste capítulo está organizado da seguinte forma. A Seção 7.1 fornece uma breve descrição das características e resultados do primeiro estudo de caso. A Seção 7.2 apresenta o estudo comparativo qualitativo; a Seção 7.2.1 apresenta a abordagem baseada em padrões para o desenvolvimento de aplicações com base em agentes e a aplica ao sistema Portalware; a Seção 7.2.2 compara a abordagem orientada a aspectos com uma abordagem baseada em padrões; também avalia as vantagens e desvantagens relativas da aplicação da proposta orientada a aspectos. A Seção 7.3 descreve o estudo de mapeamento envolvendo AspectJ e Hyper/J. A Seção 7.4 apresenta o quarto estudo de caso qualitativo. Já a Seção 7.5 discute os trabalhos relacionados. A Seção 7.6 apresenta algumas conclusões e direções para trabalhos futuros. O Capítulo 8 descreve o estudo quantitativo.

7.1
Arquitetura do simulador de tráfego

Este estudo de caso envolveu a implementação de uma arquitetura de agente orientada a aspectos para a construção de um simulador de tráfego inteligente [51]. Chamava-se TSA (Traffic Simulator Architecture - Arquitetura do Simulador de Tráfego) e fez parte de um projeto desenvolvido fora do nosso grupo de pesquisa. Esse projeto começou há dois anos no Laboratório de Sistemas Integrados [51] na Universidade de São Paulo (USP). O modelo do tráfego usado é um refinamento do modelo autômato celular para a simulação do tráfego [178, 177]. Os projetistas da arquitetura usaram o método orientado a aspectos, apresentado no Capítulo 4, uma vez que a reusabilidade e a manutenibilidade exerciam um papel essencial nos requisitos do sistema. Seu trabalho também baseava-se na primeira publicação sobre o método arquitetural [80]. Não usaram explicitamente a linguagem de padrões orientada a aspectos (Capítulo 5).

Os agentes de software representam veículos no ambiente, ou seja, o tráfego. O ambiente é composto de ruas, que são subdivididas em faixas. As faixas são divididas em células. As ruas, faixas e células são modeladas como objetos na implementação da TSA. Cada célula está vazia ou ocupada por um veículo com uma velocidade inteira de $V_i \in \{0, \ldots, v_{max}\}$. As células e os veículos têm o mesmo tamanho. Cada veículo é considerado individualmente como um agente na simulação e é capaz de tomar decisões em cada etapa da simulação.

Os agentes implementam os concerns de agência (Seção 3.3). A arquitetura baseia-se no modelo BDI (Seção 3.3.1) a fim de facilitar a representação e a compreensão do conhecimento do carro [51]. As crenças são as informações sobre a situação do tráfego, os objetivos descrevem que tipos de ações serão realizadas, e os planos implementam as estratégias do veículo para alcançar seus objetivos. Os planos do agente implementam regras que descrevem o movimento do veículo [51]. Além disso, implementam os procedimentos a fim de lidar com as situações típicas e as exceções no tráfego. Esses planos tratam o nível mais baixo da simulação do tráfego, ou seja, o movimento do carro, movendo-se passo a passo. Os agentes observam os eventos no ambiente para executar os planos reativos. Os agentes adaptam seu conhecimento e seus planos de acordo com a situação do tráfego.

O simulador de tráfego foi desenvolvido por um aluno de mestrado [51]. A avaliação foi qualitativa e feita com base em um questionário enviado a ele. De acordo com sua experiência e suas respostas, os principais benefícios do método arquitetural orientado a aspectos foram estes:

- Ele relatou que as classes e os aspectos ajudaram-no a se concentrar em diferentes problemas em diversos momentos, e isso foi útil no desenvolvimento da TSA. Usando suas próprias palavras, foi possível se concentrar em aspectos do agente específicos separadamente:

  "O uso de aspectos ofereceu suporte ao tratamento separado das propriedades do agente. Foi possível trabalhar incrementalmente em partes bem determinadas do comportamento do agente. Facilitou o processo de construção e a manutenção do comportamento dos "agentes

do veículo", permitindo que novos aspectos pudessem ser incorporados de forma incremental."

A ordem do tratamento de aspectos foi: o conhecimento intrínseco do agente foi criado primeiro, em seguida, veio a implementação do concern de interação, o concern de adaptação e o concern de autonomia.

- Isso ofereceu suporte a uma melhor separação de concerns porque os aspectos permitiram a introdução do comportamento de detecção (Seção 3.3.2) em classes do ambiente não primariamente criadas para serem detectadas. Os aspectos de interação foram usados para observar as classes que representam entidades do ambiente, como faixas, células e ruas, de forma transparente.
- Como o desenvolvimento da arquitetura evoluiu, foram apresentados novos planos para agentes. Esses planos deveriam enviar mensagens para outros agentes. A evolução arquitetural foi modular uma vez que somente precisava da extensão de uma classe Plan abstrata. Um aspecto de interação interceptou as novas classes Plan de forma transparente para essas classes.

Por outro lado, o desenvolvedor de SMAs ressaltou o alto nível de abstração do método arquitetural como uma desvantagem. Apesar de o método ter fornecido diretrizes para construir arquiteturas de agente com uma melhor separação de concerns, foi difícil produzir o projeto interno dos aspectos do agente. O método é genérico e não deve oferecer suporte ao refinamento de soluções arquiteturais orientadas a aspectos. Por exemplo, não é fácil saber como criar uma hierarquia de aspectos para diferentes especializações de um aspecto de autonomia.

De fato, o objetivo do método orientado a aspectos é fornecer uma solução arquitetural para uma ampla gama de domínios de aplicações. Ele não pretende apresentar diretrizes do projeto detalhado. Portanto, os resultados desse estudo foram a motivação para o desenvolvimento da linguagem de padrões (Capítulo 5). Apesar de não ter usado explicitamente os padrões orientados a aspectos no desenvolvimento da TSA, ele usou instâncias de três soluções de padrões em sua implementação do conhecimento de agente e nas propriedades de interação e adaptação.

Apesar desses resultados interessantes como uma investigação inicial, esse estudo era muito limitado em relação à avaliação devido a duas razões. Primeiro, só tratava de concerns de agência (Tabela 4). Segundo, não havia uma solução orientada a objetos para o mesmo sistema com o qual poderíamos estabelecer uma comparação. Essas limitações motivaram a realização do segundo estudo de caso, descrito na próxima seção.

7.2 O estudo comparativo

Conforme discutido anteriormente, os benefícios do método orientado a aspectos (Capítulo 4) e os padrões (Capítulo 5) pareciam bastante atraentes em relação à construção, evolução e reutilização. Entretanto, uma avaliação realística e sistemática deve ser conduzida a fim de avaliar a utilidade potencial, os benefícios e as desvantagens da abordagem orientada a aspectos em termos de alguns critérios qualitativos. Assim, foi desenvolvido um estudo de caso [82] para avaliar a abordagem orientada a aspectos e uma abordagem orientada a padrões para o projeto e a implementação do Portalware. O sistema Portalware foi descrito na Seção 4.1.

O principal objetivo desse estudo de caso era caracterizar, avaliar e comparar essas abordagens construídas a partir da perspectiva de projetistas de SMAs no escopo de um único projeto. Esse estudo não usou o framework de avaliação quantitativa (Capítulo 6) uma vez que incorporava uma comparação qualitativa. Ele foi realizado por duas equipes diferentes em paralelo, cada uma usando uma abordagem específica. Eles construíram o sistema Portalware seguindo os mesmos requisitos.

As equipes usaram as abordagens para criar e implementar soluções orientadas a aspectos e padrões para o Portalware. A “equipe orientada a aspectos” usou o método arquitetural (Capítulo 4), a linguagem de padrões orientada a aspectos (Capítulo 5) e a linguagem de programação AspectJ. A “equipe orientada a padrões” usou a abordagem de projeto e a arquitetura descrita na Seção 7.2.1 e a linguagem de programação Java. As duas equipes compartilhavam o objetivo de desenvolver soluções com manutenibilidade e reusabilidade. A comparação considerou critérios

qualitativos (como gravabilidade, leiturabilidade, manutenibilidade e reusabilidade) como as principais qualidades dos projetos. A Seção 7.2.1 apresenta o método orientado a padrões, e a Seção 7.2.2 discute a comparação qualitativa.

7.2.1
A abordagem orientada a padrões

Esta seção apresenta a abordagem orientada a padrões para o desenvolvimento do Portalware. Foi usada uma abordagem arquitetural baseada em mediadores (Seção 3.7.3) para alcançar modularidade e extensibilidade. Foram aplicados padrões de projeto bem estabelecidos [19, 35, 79] ao projeto do sistema. Os padrões de projeto ofereceram soluções para estruturar e separar a representação de concerns de agente com base em projetos orientados a objetos. A idéia era garantir que o sistema pudesse evoluir de forma previsível e específica.

Muitos padrões tinham algumas alternativas e variantes de implementação. Se um padrão oferecesse mais de uma implementação possível, era escolhida aquela solução que parecesse ter mais reusabilidade e manutenibilidade. Houve uma tentativa de superar as desvantagens da orientação a objetos apresentadas na Seção 3.6 e no Capítulo 5. Tentaram maximizar a chance de obter as melhores soluções orientadas a objetos para cada concern do agente. Por exemplo, os sujeitos usaram classes separadas para cada propriedade do agente. O padrão Mediador foi usado para compor as propriedades.

É válido mencionar que os sujeitos não usaram qualquer um dos padrões orientados a objetos apresentados nas seções “Exemplo de motivação” no Capítulo 5. Em alguns pontos específicos do projeto, o concern era simples e não precisava de uma solução sofisticada. Nesses casos específicos, o enfoque estava na simplicidade do projeto. O uso de padrões provavelmente aumentaria o número de componentes e o nível de acoplamento.

Doze instâncias dos padrões de projeto foram aplicadas ao contexto do Portalware e foram combinadas de acordo com os requisitos do sistema. Incluíam: (i) padrões genéricos - Avulso, Composição, Observador, Mediador, Adaptador, Estratégia [79], (ii) padrões de concorrência, como Objeto Compartilhado [154] e

Objeto Ativo [153] e (iii) padrões para o domínio do agente, como Papel do Objeto [19] e Blackboard [35]. As subseções a seguir enfatizam os padrões mais importantes. Dessa forma, incorpora a descrição das propriedades do agente e dos papéis usando padrões de projeto, assim como questões relacionadas à composição do aspecto e a evolução do agente. O padrão do conhecimento do agente e os tipos de agente seguem as mesmas decisões de projeto já apresentadas no Capítulo 4.

Concerns de agência

O padrão de projeto Mediador [79] foi usado para modelar a arquitetura básica dos agentes do Portalware (Figura 81). Ele captura as propriedades de agência básica incorporadas pelo agente e como elas interagem com objetos Agent. A intenção do padrão de projeto Mediador é definir um objeto (o *mediador*) que encapsula como um conjunto de objetos (os *colegas*) interage. O padrão Mediador permite aos projetistas variar como e quais objetos interagem entre si de forma elegante. A interface do mediador classifica o kernel do agente, assim como a conexão entre as propriedades de agência.

Os concerns do agente são modularizados como classes e exercem o papel de colegas no padrão — ou seja elas se comunicam, mas não se referem entre si diretamente, somente por meio do mediador. Esse padrão facilita a adição de novas propriedades de agência. Ele apenas requer a subclassificação da classe Property (Figura 81). Ele oferece um bom grau de separação de concerns porque as classes modularizam e encapsulam de forma adequada as propriedades de agência. Finalmente, ele disciplina a composição de propriedades, uma vez que a comunicação entre propriedades localiza-se no mediador, ou seja, na classe Agent.

Concerns adicionais

Concerns adicionais (Seção 3.4) também são representadas como colegas no padrão Mediador. Novas propriedades são adicionadas dividindo-se em subclasses a classe Property. Entretanto, são necessários meios expressivos adicionais na visão estrutural para declarar que a referência ao mediador (classe Agent) conterá apenas o tipo específico de agente (classe InformationAgent, UserAgent ou InterfaceAgent) ao qual a nova propriedade deve ser associada.

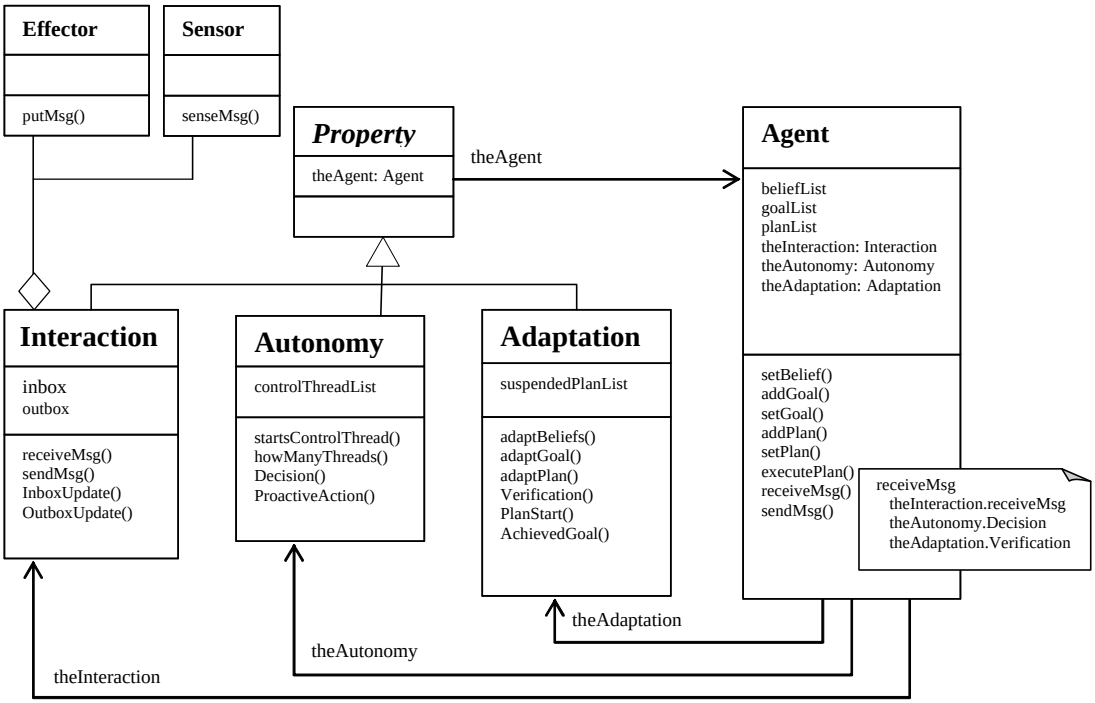

Figura 81. Usando o padrão Mediador para desenvolver concerns de agência.

Ademais, a adição ou remoção de uma propriedade requer modificação invasiva do tipo de agente correspondente a fim de adicionar ou remover um link da nova propriedade. Uma solução alternativa foi o uso de herança para implementar um novo tipo de mediador que incorpora a nova propriedade (subclassificação Agent, o mediador, para criar CollaborativeAgent, por exemplo). Infelizmente, nesse estudo de caso, a subclassificação do mediador não funcionaria de forma adequada, uma vez que a classe Agent já possui subclasses (os tipos de agente).

Papéis

A idéia era isolar e encapsular cada papel do agente. O projeto deveria permitir a composição de vários papéis com o kernel do agente no contexto das colaborações específicas. Ele deveria promover a evolução do agente modular independente da adição e remoção de papéis. O padrão de projeto Objeto do Papel [19] foi uma escolha de projeto adequada uma vez que permite que os projetistas variem como os objetos se comportam em um determinado contexto, permitindo o acoplamento e

desacoplamento dos objetos do papel do kernel do agente. O agregado do objeto resultante representa um objeto lógico, apesar de consistir em objetos do papel fisicamente diferentes.

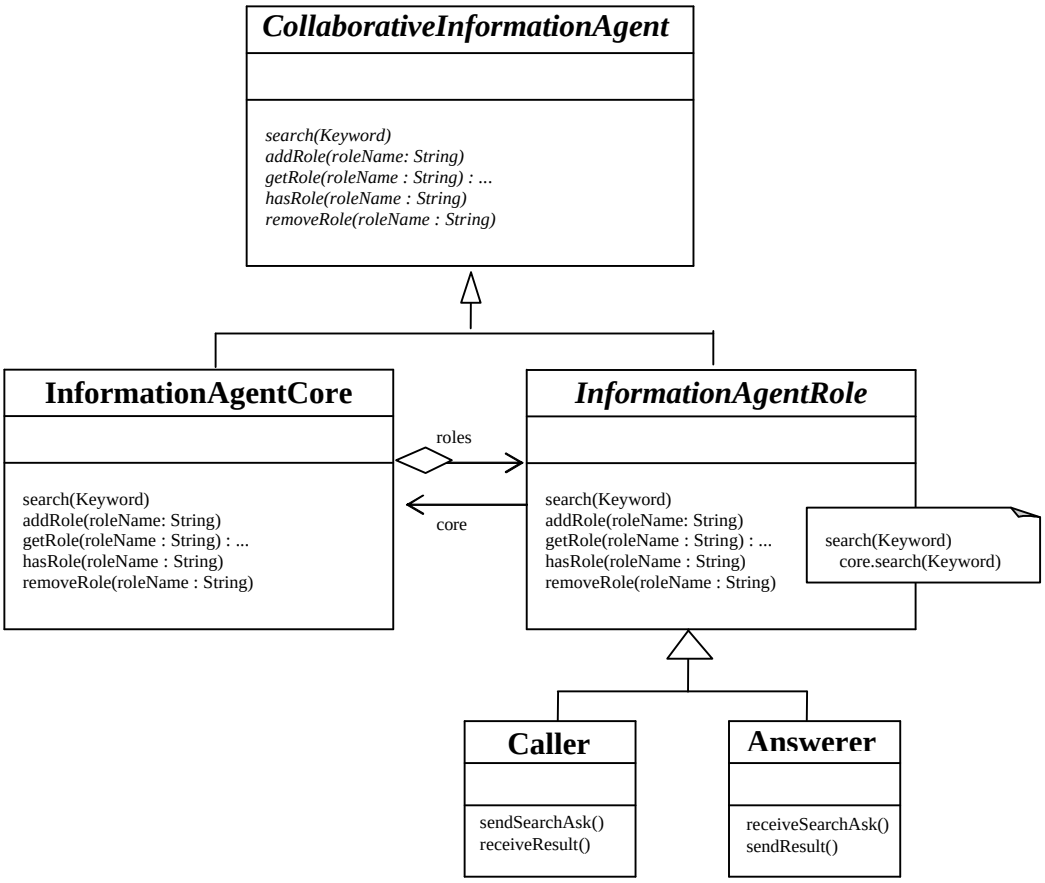

Figura 82. Usando o padrão Objeto do Papel para desenvolver papéis do agente.

O padrão Objeto do Papel evita a explosão combinatória do número de classes (Seção 3.6.2), como resultaria do uso de várias heranças para compor diferentes papéis em uma única classe [19]. No entanto, os clientes da classe Agent provavelmente ficarão mais complexos, uma vez que trabalhar com um objeto por meio de uma de suas interfaces do papel implica uma leve codificação em comparação ao uso da interface fornecida pela própria classe Agent. Por exemplo, os papéis de chamador e respondedor precisam ser explicitamente criados e ligados a objetos do agente de informação, e o cliente tem de verificar se o objeto exerce o

papel desejado antes da ativação explícita de algumas capacidades introduzidas por ele (Figura 82).

Composição dos concerns do agente

A necessidade de capturar e substituir as características dos vários concerns do agente (Seção 3.5) também foi tratada na abordagem baseada em padrões. Os relacionamentos entre propriedades de agência são capturados nos métodos que incorporam a interface do mediador (classe Agent). Por exemplo, o método receiveMsg() definido na classe Agent serve como um meio para descrever o protocolo da recepção de mensagens do agente, envolvendo as propriedades Interação, Autonomia e Adaptação (Figura 81). A modificação desse protocolo, incluindo a alteração do relacionamento precedente entre propriedades ou a inclusão de novas propriedades, pode precisar de alterações invasivas em relação à classe original ou algum uso falso de herança para refinar o comportamento do protocolo. Os projetistas também usaram a herança para capturar a natureza sobreposta entre os concerns de colaboração e interação. A classe Collaboration foi definida como uma subclasse da classe Interaction.

Evolução de agente

Em geral, o uso dos padrões de projeto requer um pré-planejamento para oferecer o suporte adequado à evolução sem alterações invasivas. Como conseqüência, muitas classes podem ser criadas apenas para lidar com essa demanda (Seção 3.6.3). No entanto, algumas alterações invasivas ainda podem ser necessárias. Por exemplo, considere novamente o cenário descrito na Seção 4.4.7, em que um agente de informação colaborativo é necessário para se mover através de uma rede para encontrar algumas informações, em vez de colaborar com outros agentes. Na solução baseada em padrões, essa alteração precisou destas ações: (i) a remoção de um link da classe InformationAgent à propriedade de Colaboração; (ii) a inclusão da propriedade de Mobilidade no projeto (subclassificação Propriedade); (iii) a definição de um link de associação da classe InformationAgent a essa nova propriedade; e (iv) a modificação do código exceto quando é feita uma chamada explícita ao método startCaller(), substituindo-o por outra chamada explícita a um método que começa o

processo de mobilidade. Infelizmente, exceto em (ii), todas essas alterações são invasivas.

7.2.2
A comparação qualitativa

O estudo de caso comparativo levou a resultados interessantes e *insights* relativos aos benefícios e a utilidade gerais da abordagem baseada em aspectos. A solução orientada a aspectos resolve muitos problemas que surgem quando padrões são concretizados na implementação de agentes do Portalware. O uso da abordagem orientada a padrões apresentou alguns efeitos relacionados indesejáveis. Os desafios mais importantes estavam relacionados à esquizofrenia do agente, gravabilidade, reutilização, expressividade, evolução e flexibilidade. Como uma avaliação qualitativa dos resultados reunidos, foram obtidas as conclusões a seguir:

A solução orientada a padrões aumentou a esquizofrenia do agente. O uso de uma arquitetura baseada em mediadores introduziu uma maior esquizofrenia do agente (Seção 3.6): um agente do Portalware é explicitamente dividido em quatro objetos (um objeto para o kernel do agente e vários objetos para cada uma das propriedades do agente), cada um dos quais possui sua própria identidade de objeto. Para criar um agente, quatro objetos precisam ser explicitamente criados e inicializados, de acordo com o padrão.

A solução orientada a padrões não fornece uma separação clara de concerns do agente. Os relacionamentos interpropriedade não eram explícitos no nível do projeto e da arquitetura. Reconheceram-se dentro do método apenas implementações na classe Agent do mediador. Esses métodos foram explicitamente chamados de métodos definidos na interface da classe Agent.

O método orientado a aspectos ofereceu suporte a uma maior gravabilidade. O uso de padrões de projeto, com sua demanda de um pré-planejamento para as alterações, precisava da definição de várias classes e métodos com um comportamento e uma estrutura trivial, por exemplo, classes abstratas e encaminhamento explícito de mensagens para outros objetos e métodos. Isso podia levar a despesas significativas para o desenvolvedor de software em termos de

gravabilidade e também uma menor compreensibilidade do código resultante. Com a abordagem baseada em aspectos, os sujeitos tinham de escrever menos códigos e, além disso, conseguiam (i) isolar e encapsular concerns de forma mais apropriada e (ii) compô-los sem grandes esforços.

O método orientado a aspectos ofereceu suporte a uma maior reutilização. Os padrões de projeto não têm representações de primeira classe no nível da implementação. A implementação de um padrão de projeto, portanto, não pode ser reutilizada e, apesar de o projeto ser reutilizado, o desenvolvedor de software é forçado a implementar o padrão várias vezes. Diferente dos padrões, as abordagens do DSOA oferecem representações de primeira classe no nível da implementação para os aspectos no nível do aspecto e para mecanismos de composição que se misturam pelo sistema, oferecendo suporte à reutilização nos níveis da implementação e projeto. Ademais, a abordagem proposta ofereceu suporte a uma melhor reutilização como um efeito colateral do mecanismo crosscutting da POA, que define a composição do comportamento implícito em join points bem definidos. Por exemplo, a reutilização da propriedade de colaboração no contexto dos agentes de usuário requer a associação do aspecto Collaboration à classe UserAgent, representando join points de interesse. A abordagem baseada em padrões precisou de algumas modificações adicionais para introduzir a associação e chamadas explícitas a métodos definidos na interface da classe Collaboration.

A abordagem orientada a padrões levou a problemas de documentação. Os padrões influenciam a estrutura do sistema, e suas implementações são influenciadas por ela. Como conseqüência, as implementações do padrão são normalmente adaptadas à instância de uso. Isso levou a padrões usados que “desapareciam dentro do código”, perdendo, assim, sua modularidade [115], o que dificulta a distinção entre o padrão, a instância concreta e o modelo do objeto envolvidos. A adição ou remoção dos padrões a/de um sistema é uma alteração invasiva e de difícil reversão. Conseqüentemente, embora o padrão de projeto fosse reutilizável, suas implementações, em geral, não eram. A natureza invasiva do código de padrões e seu espalhamento ou entrelaçamento com outros concerns do agente criaram problemas de documentação. Como vários padrões foram usados no sistema, é difícil traçar

instâncias particulares de um padrão de projeto, em especial, se houver classes envolvidas em mais de um padrão (ou seja, se houver uma composição/sobreposição do padrão). Esse foi o caso na combinação dos padrões Papel e Mediador.

O método orientado a aspectos ofereceu suporte a uma melhor evolução. Os sujeitos experimentais introduziram o concern de mobilidade a suas implementações como uma atividade de manutenção. A introdução da propriedade de mobilidade foi muito mais simples no projeto baseado em aspectos do que no projeto baseado em padrões com o uso do padrão Mediador. Além disso, o projeto de alteração, conforme descrito pelos padrões, às vezes impunha uma estrutura nem tão simples para oferecer a flexibilidade necessária para a evolução. Finalmente, a combinação de vários padrões de projeto no mesmo projeto não era uma tarefa trivial. A composição de padrão causou mais do que apenas problemas de documentação. Era inerentemente difícil raciocinar sobre vários padrões que envolviam as mesmas classes, porque a composição criava grandes clusters de classes mutuamente dependentes. Esse tópico é importante porque alguns padrões de projeto usavam explicitamente outros padrões na solução.

O método orientado a aspectos oferece suporte a uma melhor flexibilidade para acomodar diferentes definições para agência. Apesar de termos apresentado uma definição para agência (Seção 3.3) que tenta identificar características comuns dos agentes de software, essa definição não é amplamente aceita e varia de pesquisador para pesquisador. Essa variação requer uma arquitetura de agente que seja flexível o suficiente para incorporar composições disciplinadas de concerns do agente. Felizmente, a abordagem baseada em aspectos pode acomodar diferentes definições uma vez que os aspectos do agente podem ser facilmente acoplados e removidos da classe Agent.

Assim, esse estudo ofereceu a primeira exploração substancial do método arquitetural orientado a aspectos e a linguagem de padrões. Conseguimos entender os pontos fortes e fracos da abordagem proposta em contraste com uma abordagem orientada a padrões. Também foi importante o amadurecimento das diretrizes arquiteturais propostas e os padrões de projeto. Além disso, o sistema implementado

envolvia concerns adicionais e de agência. Também foi útil identificar melhorias para os próximos estudos: (i) até esse ponto, apenas a linguagem AspectJ foi usada – não tínhamos certeza de que a abordagem proposta poderia ser usada em conjunto com outras linguagens orientadas a aspectos, (ii) essa versão do Portalware não incorporou os concerns de mobilidade e aprendizagem e (iii) como foi uma análise qualitativa, poderia sofrer a influência de nossas intuições. A primeira restrição motivou o desenvolvimento do terceiro estudo de caso, descrito na próxima seção. A segunda e a terceira limitação motivaram o quarto e o quinto estudo experimental.

7.3 Mapeamento de uma implementação de AspectJ I para uma implementação de Hyper/J

O terceiro estudo de caso dedicou-se ao amadurecimento da abordagem orientada a aspectos e garantiu que as soluções propostas fossem implementáveis usando diferentes ferramentas orientadas a aspectos. Neste estudo, alguns *insights* foram reunidos por meio da implementação do sistema Portalware usando as ferramentas IBM Hyper/J [235]. O ponto de partida foi os padrões de projetos orientados a aspectos (Capítulo 5) e sua implementação de AspectJ no sistema Portalware. Foi usado um subconjunto de elementos de código implementados em AspectJ no estudo de caso anterior (Seção 7.2).

Esse estudo também foi útil para compreender os pontos fortes e fracos das abordagens e ferramentas que oferecem suporte à POA e MDSoC, as mais conhecidas técnicas de programação (Seção 2.2.2). Contudo, como esta tese não se preocupa com a comparação de ferramentas orientadas a aspectos, o Apêndice II descreve as principais descobertas dessa comparação. Alguns resultados adicionais foram obtidos e discutidos [42]. A principal motivação por trás desse exercício de desenvolvimento é uma avaliação prática de cada similaridade e diferença da abordagem para ajudar na avaliação dos padrões orientados a aspectos, verificando se podem ser mapeados para os modelos de implementação que têm o suporte de AspectJ e Hyper/J.

7.3.1
De aspectos a hyperslices

O modelo de programação de AspectJ [12] oferece suporte à dicotomia baseada em aspectos (Seção 2.2.2). Os crosscutting concerns são modularizados por aspectos. A composição entre base (classes) e aspectos é definida em termos de join points relacionados à base. O comportamento crosscutting pode ser adicionado antes, depois ou próximo aos join points. A composição é definida dentro dos aspectos. A precedência entre aspectos é resolvida implicitamente (antes, depois, em volta das regras) ou explicitamente (a condição *dominante*).

Hyper/J oferece suporte a Hyperspaces [235], uma evolução em relação aos primeiros trabalhos sobre programação orientada a objetos (POS) [116] que não faz necessariamente a distinção entre crosscutting concerns e base. Os concerns são modularizados usando *hyperslices*. As regras de composição são definidas em termos das *unidades correspondentes*. Essas unidades estão relacionadas por relacionamentos de *merge* ou *sobreposição*. A composição é definida independente das hyperslices. A precedência entre hyperslices é resolvida explicitamente (declaração de hyperslices no hipermódulo, usando a condição *ordem*). A Tabela 5 resume essas características.

		AspectJ	**Hyper/J**
Crosscutting Concerns		aspectos	hyperslice
Composição	Onde	join point	unidade correspondente
	Como	antes, depois, em volta	merge, sobreposição
Especificação da composição		aspectos internos	hyperslices externos
Precedência da composição		regras implícitas dominantes	declaração de hyperslice de “ordem”
Tempo de composição		Tempo de compilação	Tempo de compilação

Tabela 5. Comparação de AspectJ e Hyper/J

7.3.2
O processo de transformação

Este estudo propôs um conjunto simples de regras de transformação, com base (i) na reescritura de cada aspecto para uma ou mais classes encapsulados por uma hyperslice separada, (ii) na transformação do código do advice em um código de método ordinário e (iii) adoção da estratégia de composição geral *mergeByName*. Esta decisão sofreu muita influência do fato de alguns construtos de Hyper/J serem limitados[7] ou ainda não estarem disponíveis (por exemplo o relacionamento da composição *merge* a ser usado com o relacionamento *equate*). É importante ressaltar que as conclusões deste estudo limitaram-se às versões específicas das ferramentas usadas e os recursos usados na implementação do Portalware.

O estudo envolveu um subconjunto da implementação de AspectJ e do Portalware (Figura 83). O conjunto resultante incorporava 3 aspectos (Interação, Autonomia e Adaptação) e quase 50 classes. O Apêndice II descreve o conjunto de regras de transformação. A seguinte condição inicial é verdadeira, simplificando o processo de transformação:

- os arquivos-fonte de Java estão disponíveis e são compiláveis;
- cada aspecto/classe foi definido em um arquivo separado;
- cada pointcut usava apenas o pointcut primitivo de “chamada”;
- para cada definição de aspecto, não foram definidos before/after advices no mesmo pointcut e não havia around advices.

A solução resultante em Hyper/J foi estruturada em uma hyperspace com quatro dimensões (Agente, Interação, Autonomia, Adaptação), quatro hyperslices principais (Agent.Kernel, Interaction.Kernel, Autonomy.Kernel, Adaptation.Kernel), um arquivo de hyperspace, um arquivo de mapeamento de concern para cada dimensão, um arquivo de hipermódulo e um conjunto de classes Java padrão. Cada arquivo do aspecto-fonte com uma declaração de aspecto de AspectJ gerou um ou mais arquivos Java, dependendo dos requisitos de integralidade declarativa impostos por Hyper/J.

[7] Por exemplo, a diretiva *bracket* que se parece com os construtos before/after advice de AspectJ.

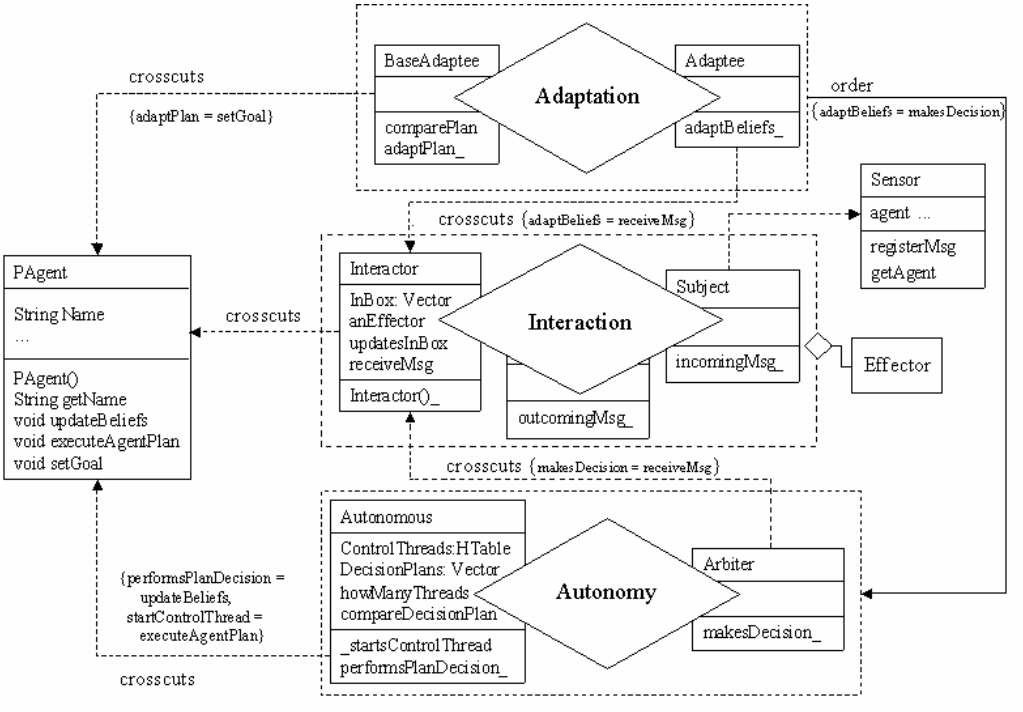

Figura 83. O modelo de projeto para aspectos de agência de agentes do Portalware.

Como uma conclusão, a arquitetura de agente proposta (Capítulo 4) e os padrões de projeto (Capítulo 5) também podem ser implementados em Hyper/J. As regras de transformação (Apêndice II) oferecem suporte a um mapeamento direto das soluções de projeto orientadas a aspectos para soluções baseadas no modelo de programação de Hyper/J. As duas ferramentas ofereciam suporte à separação e composição de concerns de agência conforme descrito pelas diretrizes arquiteturais e a linguagem de padrões. O Apêndice II descreve outras conclusões obtidas neste estudo.

7.4
Expert Committee

A motivação do quarto estudo foi construir um sistema multiagentes maior com vários concerns de agência. Ele deveria incluir propriedades de agente não implementadas nos estudos de caso anteriores (Seção 7.2.2). Este estudo envolveu o Expert Committee, um sistema multiagentes para o gerenciamento do envio de trabalhos e os processos de seleção em conferência. Ele foi descrito na Seção 5.1. O projeto dos agentes do EC foi expresso na apresentação dos padrões de projeto

(Capítulo 5). Esse exemplo ilustra como usar a linguagem de padrões orientada a aspectos para construir uma aplicação que oferecesse diferentes concerns de agente, permitindo que os engenheiros de software reutilizassem as soluções propostas e desenvolvessem seus próprios agentes. O Apêndice I mostra a implementação dos agentes do EC.

Esse sistema incorporava todos os concerns de agência, exceto a propriedade de colaboração. Este estudo de caso envolveu agentes híbridos, ou seja, agentes reativos e cognitivos (Seção 3.2.3). A abordagem orientada a aspectos também apresentava um bom escalonamento para agentes híbridos. Os aspectos de autonomia e aprendizagem apenas incorporavam outros comportamentos. Os nomes de classe foram adicionados à lista de join points aos quais os aspectos estavam associados. Este estudo foi importante para o refinamento das soluções de padrões (Capítulo 5). Ele incorporou o uso de formas mais sofisticadas de padrões uma vez que subaspectos foram usados para representar as especializações dos aspectos de agência em diferentes tipos de agente e papéis. Além disso, foi importante detectar as limitações da linguagem AspectJ, conforme discutido no Apêndice I.

A implementação do Expert Committee precisou usar diferentes arquiteturas e técnicas. JADE foi usada como uma plataforma de agente compatível com FIPA para oferecer alguns serviços de middleware, como nomeação e mobilidade. A implementação do concern de aprendizagem usou as técnicas LMS [172] e TD-Learning [172]. Para lidar com questões de colaboração interagente, foram usados JADE e TSpaces. O uso dessas arquiteturas não exigiu a implementação de um aspecto de colaboração.

7.5 Trabalhos relacionados

Até agora, alguns estudos sistemáticos foram conduzidos para investigar a relação entre a engenharia de software orientada a objetos e agentes. Alguns frameworks de implementação orientados a objetos (Seção 3.7) foram propostos para o desenvolvimento de SMAs. No entanto, não há qualquer relato na literatura de estudos empíricos para melhorar a compreensão de como as abstrações orientadas a objetos oferecem suporte à separação de concerns recorrentes no desenvolvimento do

SMAs. Ademais, não está claro até que ponto as abstrações em desenvolvimento e amplamente usadas reforçam a observância de outros princípios desejáveis na engenharia de SMAs, como o baixo acoplamento e a alta coesão.

As pesquisas na área da engenharia de software baseadas em agentes concentraram-se basicamente no desenvolvimento de metodologias orientadas a agentes e linguagens de modelagem, sem enfoque nos relacionamentos entre agentes e objetos [16, 20]. Muitos pesquisadores [20, 21] argumentam persuasivamente que os concerns associados aos SMAs costumam ser muito diferentes daqueles tradicionalmente associados aos sistemas orientados a objetos; e, assim, as abstrações orientadas a objetos normalmente não conseguem capturar os concerns relevantes dos SMAs. No entanto, essa afirmação tem origem nas opiniões dos especialistas e nas observações baseadas na experiência informal, mas não é um resultado de comprovações empíricas.

Alguns trabalhos na literatura apresentaram estudos sistemáticos que envolviam soluções baseadas em aspectos. Além disso, como a programação orientada a aspectos ainda está começando, há poucas experiências relacionadas ao uso desse paradigma. Atualmente, a programação orientada a aspectos foi usada para implementar aspectos genéricos como persistência, tratamento/detecção de erros, auditoria, rastreamento, armazenamento e sincronização. Entretanto, cada um desses trabalhos baseia-se em geral em apenas um desses aspectos genéricos. Este capítulo fornece alguns estudos de caso que: (i) lidam com aspectos específicos a SMAs e (ii) incorporam alguns aspectos diferentes e seus relacionamentos. Contudo, esses estudos baseiam-se em critérios qualitativos. O Capítulo 8 descreve um estudo quantitativo.

7.6 Resumo

Para experimentar a abordagem orientada a aspectos, foram usados três pequenos sistemas representativos por meio de cinco estudos de caso. Este capítulo apresentou de forma resumida quatro deles. Esses sistemas forneceram uma combinação equilibrada das possibilidades relacionadas aos requisitos e às estratégias

de implementação para cada concern do agente. Eles incluem agentes híbridos, cognitivos e reativos e incorporam diferentes relacionamentos entre concerns de agência. O primeiro estudo de caso é uma arquitetura de simulador de tráfego, desenvolvida na USP. Este estudo forneceu o primeiro feedback sobre o método arquitetural orientado a aspectos (Capítulo 4) e a motivação para o desenvolvimento da linguagem de padrões (Capítulo 5).

O Portalware foi um assunto de investigação que uniu o segundo e o terceiro estudo. Ele incorpora um conjunto de agentes que gerenciam um ambiente de desenvolvimento de portal. O segundo estudo de caso comparou a abordagem orientada a aspectos com uma abordagem orientada a padrões com base em critérios qualitativos. Relatamos alguns *insights* reunidos durante a implementação da mesma aplicação, o sistema Portalware, usando as ferramentas AspectJ e Hyper/J, tendo como ponto de partida a implementação de AspectJ. Apresentamos um mapeamento preliminar entre os elementos de programação AspectJ e Hyper/J, e a solução Hyper/J resultante. Também discutimos algumas alternativas para deduzir a solução baseada am hyperslices a partir da solução orientada a aspectos. Descrevemos alguns problemas potenciais e soluções relacionadas ao processo de transformação. O Apêndice II descreve o conjunto selecionado de transformações que aplicamos. A mais importante descoberta não se refere à maturidade ou desempenho de cada ferramenta, mas sua capacidade de oferecer suporte à separação e composição de concerns de agência, conforme descrito pelos padrões (Capítulo 5). O segundo e o terceiro estudo são contribuições originais como discutido na Seção 7.5.

O quarto estudo usou um sistema multiagentes diferente, chamado Expert Committee. Foi usado basicamente para avaliar a linguagem de padrões proposta. Esse sistema é apresentado ao longo do Capítulo 5 e do Apêndice I. O próximo capítulo descreve a quinta investigação, um experimento mais controlado que usa um framework de avaliação quantitativa (Capítulo 6). Esse experimento foi realizado a fim de confirmar ou refutar os resultados dos estudos qualitativos apresentados no capítulo.