7
Estudo de Caso: Portalware

Este Capítulo apresenta o sistema multiagentes `Portalware`, um ambiente baseado na Web para o desenvolvimento de portais de comércio eletrônico (*e-commerce*), a fim de ilustrar a aplicabilidade das características descritas no Capítulo 5 para a modelagem de *concerns* transversais específicos a domínios.

Portalware é um estudo de caso de tamanho médio no domínio de Sistemas Multiagentes (SMA) que foi usado em diferentes escopos e contextos a fim de ilustrar as contribuições de vários trabalhos de pesquisa [48, 22, 23, 49, 51, 52, 118, 53, 54, 77] realizados no Grupo SoC+Agents [126] do Laboratório de Engenharia de Software (LES) da PUC-Rio. A primeira versão orientada a aspectos do SMA Portalware desenvolvida no LES seguia o método de projeto orientado a aspectos proposto por Garcia [48, 53, 54] para o domínio de sistemas multiagentes.

O SMA Portalware é um estudo de caso representativo para o projeto orientado a aspectos uma vez que, como um sistema multiagentes, ele incorpora um rico conjunto de *concerns* específicos a domínios – *concerns* de agência, como interação (*interaction*), autonomia (*autonomy*), adaptação (*adaptation*), colaboração (*collaboration*) etc. – que não são ortogonais e interagem com outros *concerns* de várias maneiras [48].

No contexto desta tese, usamos `aSideML` para apresentar uma *visão de projeto* para Portalware, que incorpore alguns artefatos de projeto criados durante o desenvolvimento orientado a aspectos. Nosso objetivo não é descrever o método de projeto propriamente dito, mas sim demonstrar que os diagramas e a notação de `aSideML` facilitam a visualização, a compreensão e a comunicação dos modelos de projeto orientado a aspectos do Portalware, dos *concerns* de agência envolvidos, seus relacionamentos e características aspectuais.

7.1 Concerns de Agência

Nesta Seção, apresentamos uma breve caracterização de agentes de software e descrevemos alguns *concerns* de agência relacionados ao projeto do SMA Portalware [48]. Uma descrição mais detalhada e abrangente de agentes de softwares pode ser encontrada em [120]. Nesse trabalho, introduzimos TAO, um framework conceitual desenvolvido no Laboratório de Engenharia de Software (LES) da PUC-Rio, que oferece as bases para a compreensão dos principais relacionamentos e conceitos de agente.

Os diferentes concerns de agência de Portalware são identificados e modelados usando diagramas de *features* [66]. Os diagramas de *features* oferecem suporte à expressão da variabilidade em um nível abstrato, que será implementado mais tarde nos modelos de análise, projeto e implementação por meio de mecanismos de variabilidade específicos.

Na abordagem proposta por Garcia [48], as técnicas orientadas a objetos convencionais são usadas para implementar algumas *features* de agência usando classes; os mecanismos de variabilidade para classes incluem composição de objeto e herança, entre outros. Outras *features* de agência são implementadas usando aspectos, e *crosscutting* é usado como um mecanismo de variabilidade.

7.1.1 Agentes de Software

Os *agentes de software* são muitas vezes vistos como objetos complexos com atitude [20], no sentido de serem objetos com algumas propriedades adicionais. Em geral, um agente é considerado uma entidade autônoma que possui capacidades, interage com seu ambiente e adapta seu estado e comportamento com base nessa interação [106]. Além do mais, os agentes de software podem colaborar com outros agentes, mover-se de um ambiente para outro ou aprender (refinar seu conhecimento) enquanto interagem com o ambiente.

A Figura 7.1 apresenta um modelo de *features* que define as características associadas ao conceito de agente: conhecimento (*knowledge*), propriedades básicas de agência (também conhecidas como agência ou *agenthood*), propriedades opcionais de agência e tipos de agente.

As *propriedades de agência* são características comportamentais que um agente pode possuir para alcançar seus objetivos. Em geral, Autonomia, Interação e Adaptação são consideradas propriedades básicas de agência

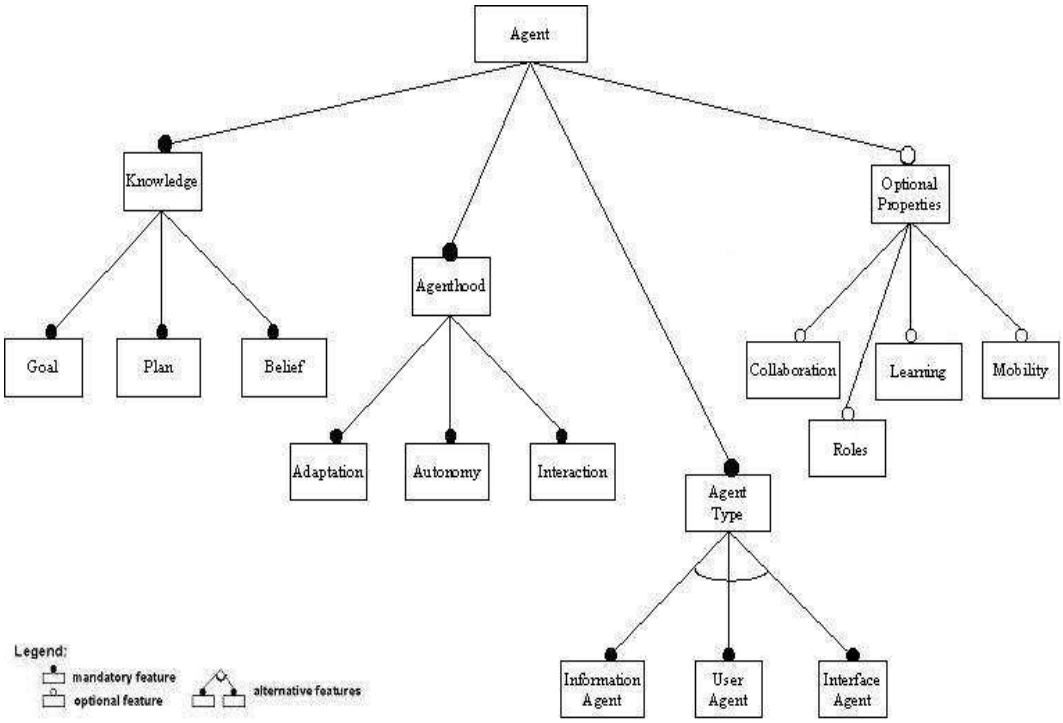

Figura 7.1: Concerns de agência.

(características obrigatórias (*mandatory features*) na Figura 7.1), enquanto Colaboração, Papéis, Mobilidade e Aprendizado são consideradas propriedades opcionais de agência – características opcionais (*optional features*) na Figura 7.1. A Tabela 7.1 resume as definições para as principais propriedades de agência.

7.1.2 Conhecimento

A característica *conhecimento* (*knowledge feature*) incorpora crenças (*beliefs*), objetivos (*goals*) e planos (*plans*) [106]. As crenças modelam o ambiente externo com os quais um agente interage. Os objetivos consistem em estados futuros, ou desejos, que o agente pretende alcançar ou satisfazer. Como os agentes são orientados a objetivos, um agente dentro de um SMA deve ter no mínimo um objetivo a ser alcançado [120]. Um agente alcança um objetivo ao executar um plano, que pode ser selecionado a partir de uma lista de diferentes planos. Os planos definem uma seqüência de ações executadas por um agente para alcançar objetivos. A seleção de planos baseia-se em suas crenças e preferências. Há diferentes tipos de planos, e eles são específicos a aplicações. Cada um deles é associado a precondições e pós-condições.

Propriedade de Agência	Definição
Interação	Um agente comunica-se com o ambiente e outros agentes através de sensores e efetuadores
Adaptação	Um agente adapta/modifica seu conhecimento e comportamento segundo eventos observados no ambiente
Autonomia	Um agente é capaz de atuar sem intervenção externa direta; ele possui sua própria thread de controle e pode aceitar ou recusar um pedido
Aprendizagem	Um agente pode aprender baseado em experiência prévia ao reagir e interagir com seu ambiente
Mobilidade	Um agente pode mover-se de um ambiente para outro em uma rede
Colaboração	Um agente pode cooperar com outros agentes para alcançar seus objetivos e os objetivos do sistema

Tabela 7.1: Visão geral de propriedades de agência.

7.1.3 Agência

A característica de *agência* (*agenthood feature*) incorpora as propriedades básicas de agência (Figura 7.1).

Interação

O *concern* de interação é a propriedade de agência que implementa a comunicação com o ambiente externo. Um *protocolo de interação* define a lógica básica do *concern* de interação. Ele consiste basicamente em detecção de eventos externos e *envio de mensagens* [54].

– **Detecção de eventos externos.**

O agente observa o ambiente a fim de adaptar seu conhecimento e verificar se um novo objetivo deve ser alcançado devido a alterações no ambiente. Esse comportamento sensorial é implementado por um *conjunto de sensores* que detectam eventos externos. Os eventos externos detectados são armazenados em uma caixa de entrada de agentes (*inbox*).

– **Envio de mensagens.**

O comportamento de envio de mensagens é implementado por *efetuadores* (*effectors*) que enviam mensagens ou geram estímulos no ambiente. Quando um agente está executando ações e planos, ele precisa enviar mensagens a outros agentes. Uma mensagem é enviada a

partir de uma ação simples ou de um plano associado ao agente propriamente dito, ou a papéis do agente. Antes de enviar a mensagem, ela é armazenada em uma caixa de saída de agentes (*outbox*).

Autonomia

O *concern* de Autonomia é a propriedade de agência que implementa a capacidade de agir sem a intervenção externa direta. Um agente pode ter algum grau de controle sob seu comportamento e estado interno com base em suas próprias experiências.

O *protocolo de autonomia* possui duas dimensões: *controle de execução* e *tomada de decisões*.

- **Controle de execução** (ou autonomia de execução [54]) é responsável pelo controle e pela incorporação de *threads* de agentes. As *threads* podem ser acopladas ao agente quando criado ou quando um evento for detectado (*concern* de interação).
- **Tomada de decisão** (ou autonomia de decisão [54]) é responsável pela tomada de decisões sobre a instanciação de objetivos. Os agentes podem decidir se aceitam ou recusam uma solicitação de serviço ou se iniciam ações sem solicitações externas (*proatividade*).

Adaptação

O *concern* Adaptação é a propriedade de agência que modifica o comportamento e o conhecimento do agente de acordo com os eventos internos e externos. O *protocolo de adaptação* consiste em observar os eventos internos ou ambientais relevantes, juntando as informações necessárias, selecionando e chamando os adaptadores associados [54]. Há dois tipos de adaptação: *adaptação de conhecimento* e *adaptação de comportamento*.

- **Adaptação de conhecimento** resulta na modificação de alguma parte do conhecimento de agente;
- **Adaptação de comportamento** resulta no cancelamento do plano ou na seleção de novos planos que devem ser executados em seguida.

A adaptação de agente ocorre em diversas circunstâncias: devido a eventos externos – por exemplo, recebimentos de mensagens – ou devido a eventos internos, alterações de crenças, definição de novos objetivos etc. Dessa forma, o agente deve ser capaz de observar o ambiente externo

(*concern* de interação) e seu próprio comportamento e estrutura interna a fim de se adaptar.

7.1.4 Propriedades de Agência Opcionais

As características opcionais de agência incorporam propriedades de agência que não são suficientes nem necessárias para caracterizá-la: Colaboração, Mobilidade e Aprendizado [106] (Figura 7.1).

Colaboração

Colaboração é a propriedade de agência que oferece a capacidade de cooperar com outros agentes a fim de alcançar seus objetivos e os objetivos de sistema. *Cooperação* significa a capacidade de coordenar com agentes não-antagonísticos a fim se alcançar um propósito comum (ser bem-sucedido ou falhar), enquanto *coordenação* significa a capacidade de realizar algumas atividades em um ambiente compartilhado com outros agentes.

Um *protocolo de coordenação* simples consiste em uma sincronização com o agente que está esperando uma resposta [53]. As atividades costumam ser coordenadas por meio de planos, fluxos de trabalho ou algum outro mecanismo de gerenciamento de processos [106].

Papéis

Um agente pode exercer diferentes papéis em SMAs. *Papéis* capturam como os agentes interagem entre si no contexto de uma colaboração.

Cada papel possui o conhecimento e o comportamento necessários para realizar as colaborações com outros agentes: um papel possui crenças, objetivos, ações e planos; pode ter comportamentos específicos para interagir com outros agentes, algoritmos de decisão específicos e estratégias de adaptação específicas. Como conseqüência, o comportamento e a estrutura do papel são semelhantes ao comportamento e à estrutura do agente [54].

Mobilidade

Mobilidade é a propriedade de agência que fornece ao agente a capacidade de se transportar de um ambiente para outro. Ela não é um requisito para a agência [106]. No entanto, é uma propriedade importante para muitos sistemas baseados em agentes e necessária para uma determinada classe de aplicações.

Aprendizagem

Aprendizagem é a propriedade de agência que fornece ao agente a capacidade de mudar seu comportamento com base na experiência. É uma forma avançada de Adaptação.

7.1.5 Tipos de Agente

Um SMA, em geral, possui vários tipos de agentes de software. Um *tipo de agente* é uma categoria específica de agentes em um SMA.

Cada tipo de agente incorpora os *concerns* de agência, mas possui diferentes propriedades de agência específicas a aplicações e oferece diferentes serviços.

Os tipos de agentes são classificados de acordo com as diferentes características ou serviços que os distinguem entre si. Os tipos de agente predominantes encontrados em quase todos os sistemas baseados em agentes incluem agentes de interface, agentes de informação e agentes de usuário [54] (Figura 7.1).

7.1.6 Sobre a natureza dos concerns de agência

As propriedades de agência não são ortogonais – elas interagem entre si de diversas maneiras (Figura 7.2).

Por exemplo, a Adaptação depende da Autonomia porque é necessário adaptar o conhecimento e o estado do agente quando a propriedade Autonomia decide aceitar uma mensagem recebida. Além disso, algumas propriedades de agente são sobrepostas, como a Interação e a Colaboração. A Colaboração é vista como uma forma de interação mais sofisticada, que incorpora comunicação e coordenação. Durante a colaboração inter-agente, as mensagens são recebidas de agentes participantes e enviadas para eles.

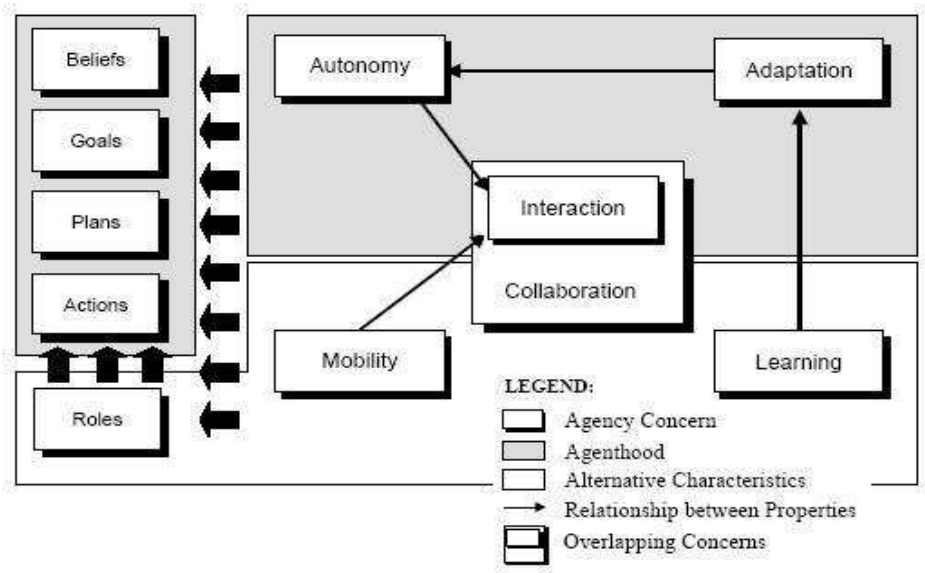

Figura 7.2: Relacionamento entre propriedades de agência [53].

Ademais, a propriedade de Colaboração também define como colaborar; ela trata do protocolo de coordenação. A propriedade Interação só está preocupada com a comunicação, ou seja, o envio e o recebimento de mensagens.

Concerns de agência transversais

Os sistemas multiagentes podem ser vistos como sistemas orientados a objetos que incorporam muitas propriedades, inclusive interação, adaptação, autonomia, colaboração, aprendizagem e mobilidade.

As abstrações de programação e projeto orientado a objetos oferecem suporte à separação adequada de alguns *concerns* de agência, como conhecimento (crenças, objetivos e planos) e tipos de agente. Contudo, elas não oferecem suporte à separação de propriedades de agência e, portanto, estas ficam entrelaçadas e espalhadas no projeto e na implementação de sistemas multiagentes [51, 52, 118].

7.2 Projeto Orientado a Aspectos de Portalware

O desenvolvimento do Portalware seguiu um método orientado a aspectos proposto por Garcia [53, 54] para o domínio de sistemas multiagentes. Primeiro o método orienta a organização dos *concerns* que são mais naturalmente suportados por abstrações orientadas a objetos e, em seguida, os *concerns* que não são, especificados por meio de aspectos (Figura 7.3).

Essa estratégia é um refinamento da regra geral para o projeto orientado a aspectos apresentada no Capítulo 8:

> *Tente manter um modelo de objetos independente, que possa ser estendido por aspectos* [69].

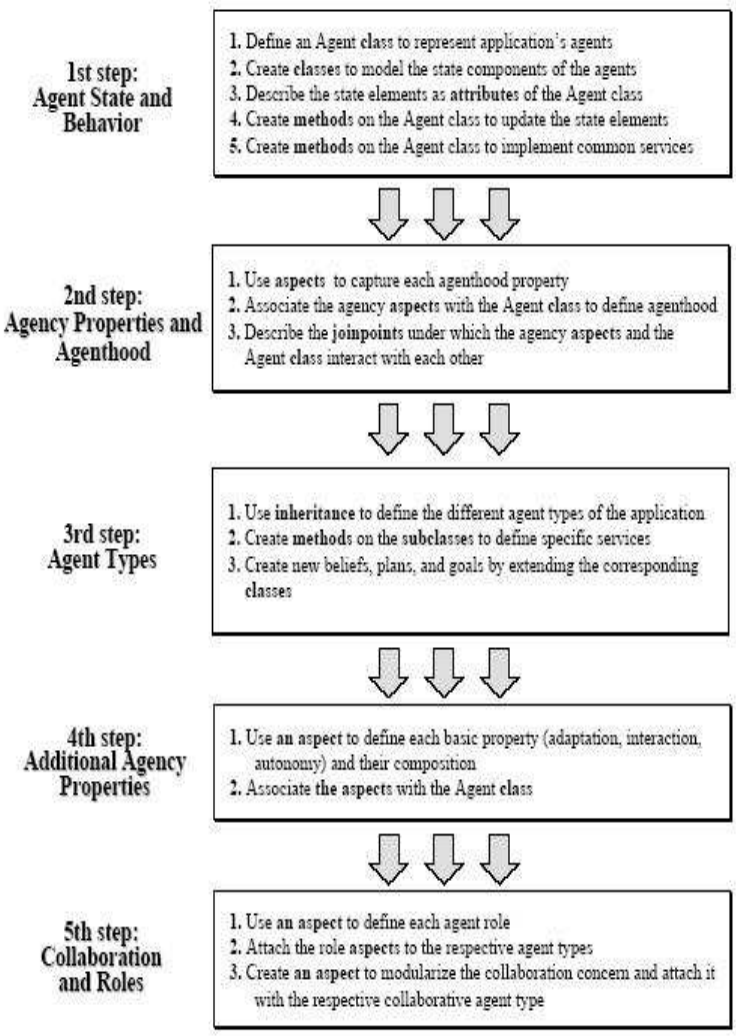

Figura 7.3: O método orientado a aspectos para o desenvolvimento de SMAs [53].

7.2.1 O SMA Portalware

Portalware é um ambiente baseado na Web que oferece suporte ao desenvolvimento e gerenciamento de portais de comércio eletrônico [50]. Esse ambiente introduz uma abordagem sistemática para a produção e a manutenção de portais por meio da separação clara dos papéis de usuário.

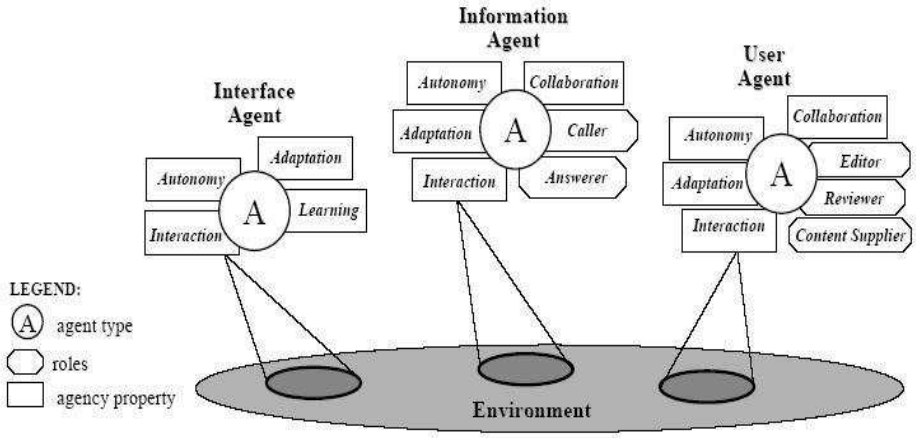

Figura 7.4: Agentes no SMA Portalware [53].

Portalware também oferece suporte a recursos de comunicação para a coordenação e a organização de atividades de desenvolvedores de portais.

Os agentes de software foram introduzidos ao Portalware [48] a fim de auxiliar seus usuários nas atividades que consomem tempo e nas tarefas repetitivas. O SMA Portalware incorpora todos os *concerns* de agência descritos anteriormente (Figura 7.1) e define os serviços e os papéis colaborativos específicos aos diferentes tipos de agente. A Figura 7.4 apresenta os tipos de agente no ambiente Portalware e as diferentes propriedades de agência e os papéis associados a eles.

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

Agentes de usuário representam os usuários do Portalware e são implementados para reduzir a necessidade de conversa cruzada entre os usuários. Os usuários podem exercer o papel de fornecedor de conteúdo (*content supplier*), revisor ou editor. Um *fornecedor de conteúdo* é responsável pelo fornecimento de informações que serão divulgadas no portal da Web. Um *revisor* examina cada um dos segmentos do conteúdo e pode alterá-los. O *editor* é responsável pela seleção de alguns segmentos de conteúdo disponíveis para publicação e pela atribuição dos papéis de revisor ou fornecedor de conteúdo a agentes individuais. Como os editores, revisores e os fornecedores de conteúdo precisam se comunicar entre si para manter o portal da Web, os agentes de usuário incorporam esses papéis e recursos colaborativos para automatizar e oferecer suporte à colaboração em diferentes contextos.

Agentes de informação são responsáveis pela busca de informações armazenadas em banco de dados. Seu conhecimento inclui um plano de busca que determina o serviço de busca do agente. Os agentes de informação podem exercer o papel de *caller* (chamador) ou *answerer* (respondedor). Sempre que o agente de informação não conseguir encontrar a informação no banco de dados disponível, deve ser usado um plano colaborativo alternativo. Esse plano ativa o *papel de chamador* a fim de chamar os outros agentes de informação e pedir as informações desejadas. De forma similar, o agente chamado usa um plano colaborativo que ativa um *papel respondedor*, de forma que possa receber a solicitação e enviar de volta o resultado da busca.

7.2.2 Uma Visão Preliminar de Portalware

A primeira versão de um protótipo orientado a aspectos, baseado em agentes do Portalware foi implementada em AspectJ entre janeiro e fevereiro de 2001.

Nessa época, a programação orientada a aspectos ainda era uma tecnologia recente, e havia muito pouca experiência no seu uso. Apesar de a usabilidade de POA estar aumentando no nível da implementação, por meio da estabilidade fornecida por algumas linguagens e o desenvolvimento de novos ambientes de programação, não havia suporte adequado para a modelagem e projeto orientados a aspectos.

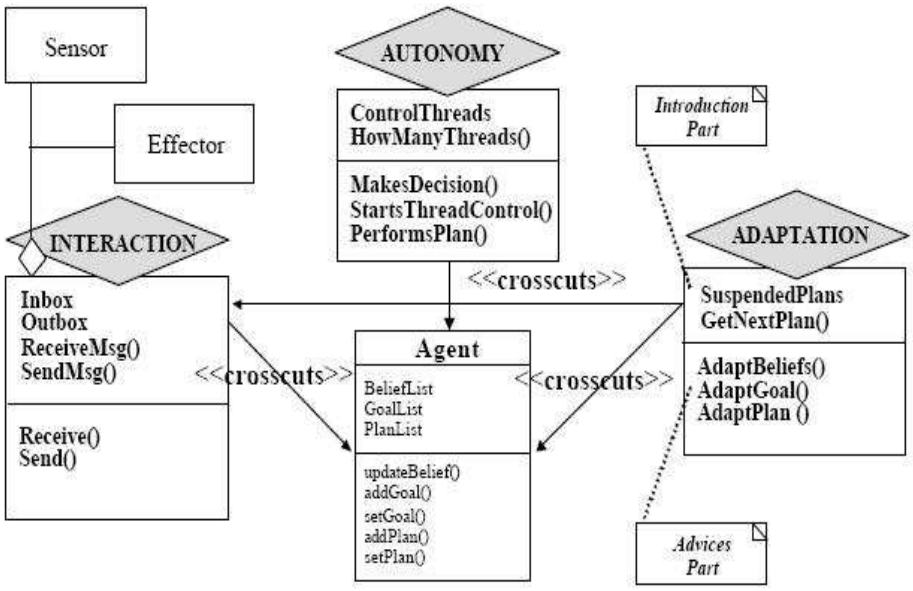

Figura 7.5: Visão estática preliminar de agência para Portalware [47].

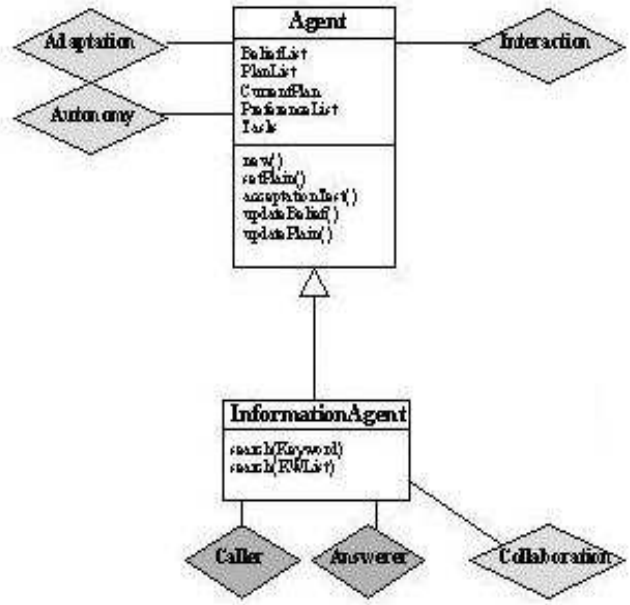

Figura 7.6: Visão estática preliminar para agentes de informação [47].

As Figuras 7.5, 7.6 e 7.7 apresentam os primeiros diagramas no nível do projeto usados para descrever visões estáticas e dinâmicas no Portalware [47, 48]. A notação fornecida para as duas visões faz a distinção entre aspectos e classes.

Na Figura 7.5, os aspectos são apresentados como caixas, que separam as *introductions* de advices usando dois compartimentos. Os nomes no compartimanto de advice documentam as ações que serão realizadas quando um ponto de combinação é alcançado. A visão estática também apresenta um conjunto de relacionamentos `<<crosscuts>>` que imita o conjunto de dependências de *concerns* apresentado na Figura 7.2. Contudo, a partir do diagrama apresentado, também não conseguimos entender por que e como os aspectos afetam a classe Agente e afetam uns aos outros, e os pontos de combinação e os tipos de advice não são expostos. Finalmente, se buscarmos a descrição de *concerns* de agência apresentada na Seção 7.1, Interação, Autonomia e Adaptação implementam protocolos específicos que podem ser decompostos (por exemplo, detecção de eventos e envio de mensagem para Interação). Esses aspectos afetam diferentes classes de forma heterogênea e, portanto, requerem suporte a *crosscutting* horizontal. Entretanto, não é possível expressar de maneira clara essas facetas sem o suporte de interfaces transversais. A notação da visão estática apresentada na Figura 7.5 não é adequada para a descrição do projeto orientado a aspectos do Portalware. De fato, conforme mencionado em [47, 48], esse diagrama de classes melhorado é usado com o objetivo de fornecer uma visão arquitetural do Portalware. A Figura 7.6 apresenta uma visão estática de agentes de informação e aspectos

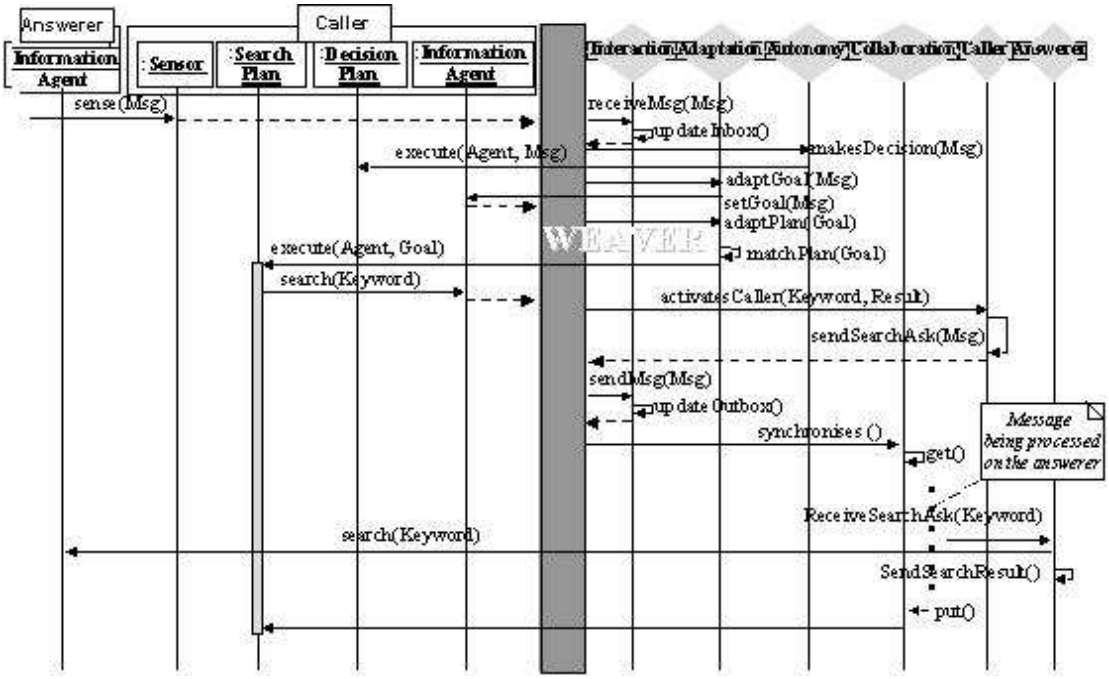

Figura 7.7: Visão dinâmica preliminar para Portalware [47].

relacionados.

A Figura 7.7 apresenta um tipo de diagrama de seqüência de UML estendido com uma barra vertical que representa um "combinador de aspectos". Essa notação expressa uma solução que lembra a separação fornecida pelas arquiteturas metanível, organizando os objetos (os objetos "base") à esquerda e os aspectos (os "metaobjetos") à direita. A barra vertical que representa o "combinador" sugere o tipo de semântica de "interceptação" que não é necessariamente adequado para a expressão de interações entre aspectos e objetos devido aos mecanismos de combinação de POA.

As setas tracejadas são usadas para denotar o desvio do fluxo de controle do contexto de execução de objetos até o contexto de execução de aspectos. Há dois problemas. Primeiro, as setas tracejadas são uma notação de UML padrão usada para expressar o retorno de um método. Segundo, a notação não explica se esse desvio ocorre antes, durante ou depois da execução do método "interceptado". Os pontos de combinação não são expostos, e o leitor deve buscar as extremidades das setas tracejadas a fim de encontrá-los. Além do mais, o diagrama apresenta algumas inconsistências de acordo com a semântica de AspectJ, como a associação da execução do método `receiveMsg` (introduzido na classe `Agent`) à linha de vida do aspecto `Interaction`.

Para as visões dinâmicas do projeto orientado a aspectos, é um desafio

prover uma semântica independente da linguagem e padrão, como aquela da orientação a objetos, o envio de mensagens e despacho dinâmico. A razão disso é que ainda não chegamos exatamente a um consenso em relação a uma semântica padrão para a POA.

Evolução de Portalware

O desenvolvimento do Portalware seguiu o método orientado a aspectos proposto por Garcia [53] no domínio de sistemas multiagentes; contudo, as primeiras versões do Portalware foram desenvolvidas sem a ajuda de idiomas ou diretrizes e sem qualquer experiência documentada na programação e projeto orientado a aspectos com AspectJ em domínios arbitrários. Por exemplo, algumas classes de Portalware não eram transparentes e eram acopladas a aspectos, mesmo quando esse tipo de acoplamento podia ser evitado. Algumas decisões relativas aos primeiros projetos de aspectos individuais tiveram de ser reconsideradas devido a limitações impostas por AspectJ. Nesse contexto, usamos algumas diretrizes gerais apresentadas no Capítulo 8. Ademais, alguns pointcuts dependiam de uma convenção de nomenclatura usada para facilitar a escolha de pontos de combinação, por exemplo, usando "send" como um prefixo aos nomes das diversas operações definidas em diferentes classes e aspectos (a classe `Plan` e suas subclasses, os aspectos `Caller` e `Answerer` etc.). Essa convenção de nomenclatura ainda ocorre na implementação de Portalware, uma vez que facilita o *crosscutting* das operações (send*) pelo aspecto Interaction.

A linguagem AspectJ evoluiu e está sujeita a alterações semânticas e sintáticas desde sua criação. Por exemplo, a primeira versão de AspectJ de Portalware usava o designador de pointcut *receptions*, que foi eliminado da linguagem desde a versão 1.0 [11]. Como conseqüência, o projeto do Portalware incorpora apenas pontos de combinação de *execução de métodos*.

O projeto orientado a aspectos do Portalware também evoluiu desde então, às vezes para migrar para uma nova versão de AspectJ, às vezes para manutenção corretiva e outras para implementar um novo requisito. O projeto do Portalware foi sendo melhorado através da experiência adquirida ao longo do tempo.

7.3 Revisitando Portalware com aSideML

Nas próximas Seções, revisitamos o projeto orientado a aspectos do Portalware a fim de ilustrar seus principais relacionamentos e elementos de projeto usando diagramas de aSideML. A ordem de apresentação dos diagramas nesta Seção não segue necessariamente as etapas de desenvolvimento apresentadas na Figura 7.3.

Em cada subseção, usamos o termo "composição progressiva" a fim de enfatizar que cada aspecto foi composto em etapas independentes. A versão Portalware apresentada aqui pode não ser a mais recente disponível no Laboratório de Engenharia de Software (LES) da PUC-Rio.

7.3.1 Conhecimento

As características de conhecimento (*knowledge features*) são concretizadas por classes. A Figura 7.8 apresenta um diagrama de classes de UML que descreve as classes Agent, Belief, Goal e Plan.

A classe Agent agrega crenças, objetivos e planos. Os métodos de agente são usados para atualizar esses atributos e implementar os recursos de agentes. A classe Agent especifica o comportamento e o estado de um agente e pode ser estendida para criar os diferentes tipos de agente da aplicação.

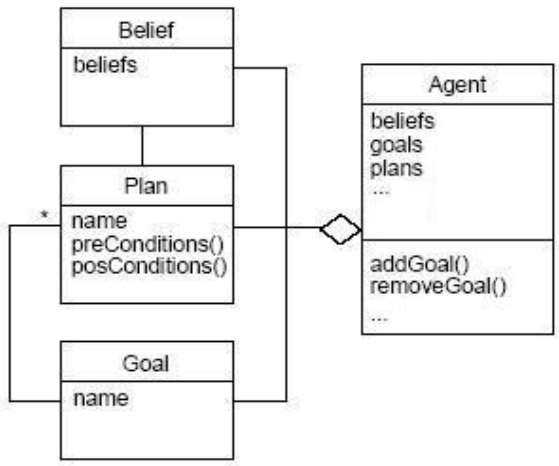

Figura 7.8: Estado e comportamento de Agent (Conhecimento).

7.3.2 Tipos de Agentes

Os diferentes tipos de agentes são organizados hierarquicamente como subclasses que herdam da classe Agent. A Figura 7.9 ilustra as subclasses que representam diferentes tipos de agentes do SMA Portalware: a classe InterfaceAgent, a classe InformationAgent e a classe UserAgent.

Os métodos dessas subclasses implementam as ações de cada tipo de agente. Por exemplo, o método `search(keyword)` da classe InformationAgent implementa os serviços de agentes de informação buscando informações de acordo com a palavra-chave especificada.

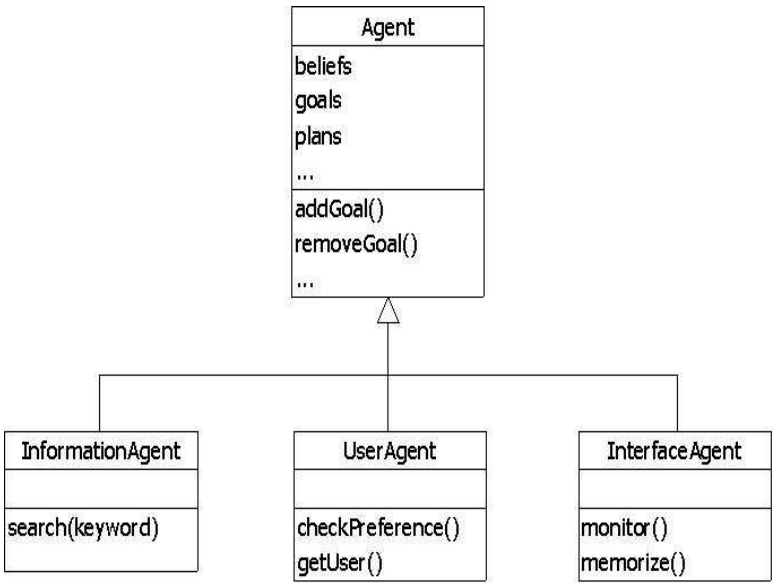

Figura 7.9: Tipos de agente.

7.3.3 Interação

O aspecto Interaction é usado para capturar de forma clara o protocolo de interação. A Figura 7.10 apresenta a visão completa do aspecto Interaction.

O aspecto Interaction modulariza o comportamento relacionado ao protocolo de interação e recorre a duas interfaces transversais chamadas `IMessaging` e `ISensoring` a fim de localizar os comportamentos sensorial e de envio de mensagem descritos na Seção 7.1.

Cada instância de agente deve ter sua própria caixa de entrada, saída, sensores e efetuadores. A interface transversal IMessaging define esses atributos no compartimento Additions que devem ser introduzidos em alguma

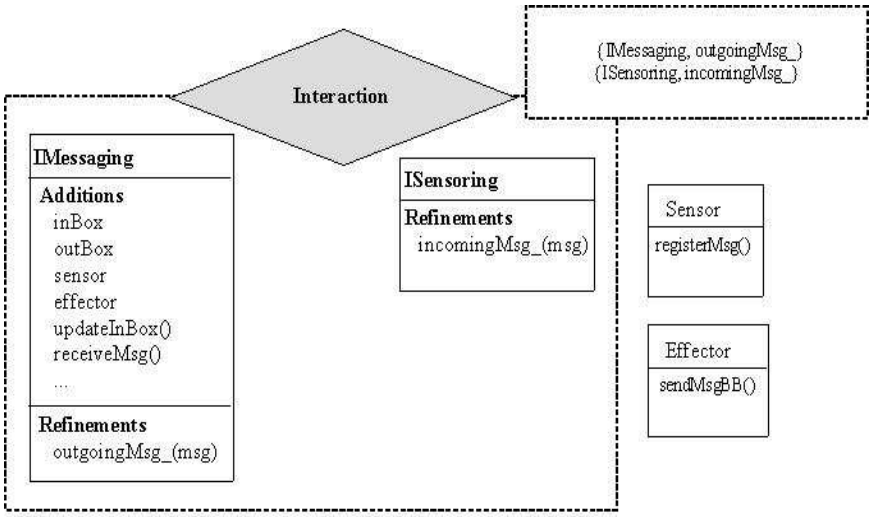

Figura 7.10: Visão estática do aspecto Interaction.

classe base. A operação de *crosscutting* `outgoingMsg_()` refina operações base que enviam mensagens, com comportamento para atualizar a caixa de saída do agente por um efetuador. A Figura 7.11 apresenta um diagrama de seqüência que descreve a operação de *crosscutting* `outgoingMsg_()`, e a Figura 7.12 apresenta uma interação aspectual para `outgoingMsg_()`. A Figura 7.13 apresenta uma interação aspectual para `incomingMsg_()`. Ela especifica que `incomingMsg_()` refina a operação `Sensor.registerMsg` depois de sua execução.

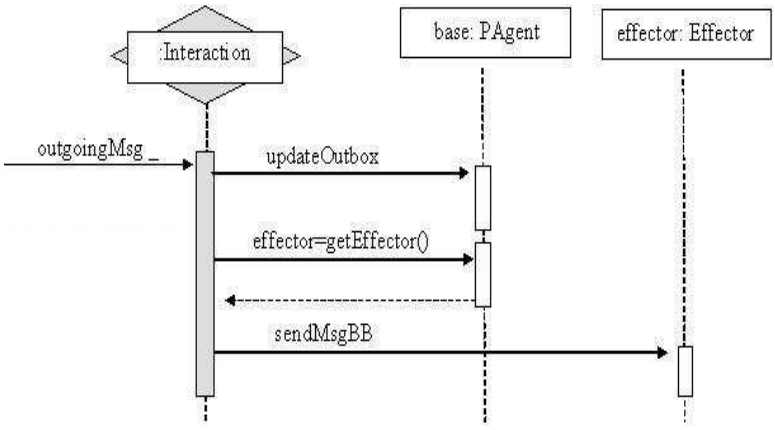

Figura 7.11: Diagrama de sequência para `outgoingMsg_()`.

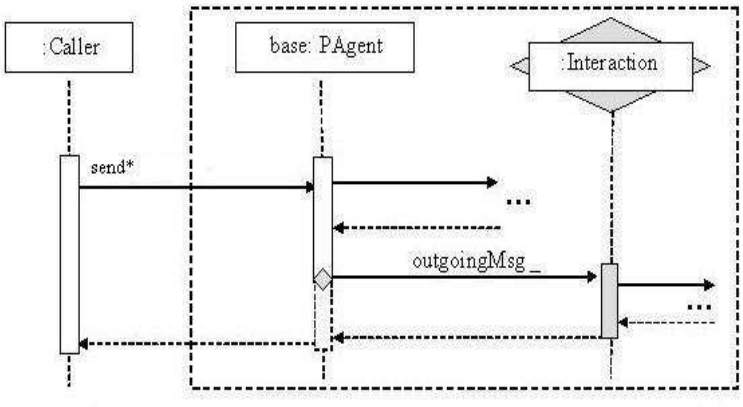

Figura 7.12: Interação aspectual para `outgoingMsg_()`.

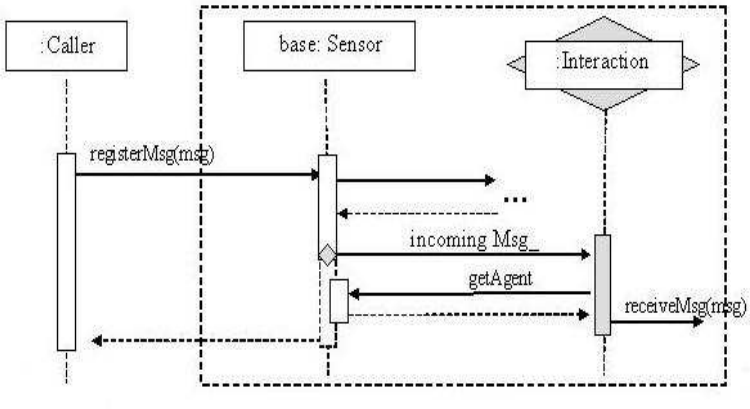

Figura 7.13: Interação aspectual para `incomingMsg_()`.

Composição Progressiva: associando o aspecto Interaction à classe Agent

A Figura 7.14 mostra que o aspecto Interaction afeta a classe Agent e a classe Sensor.

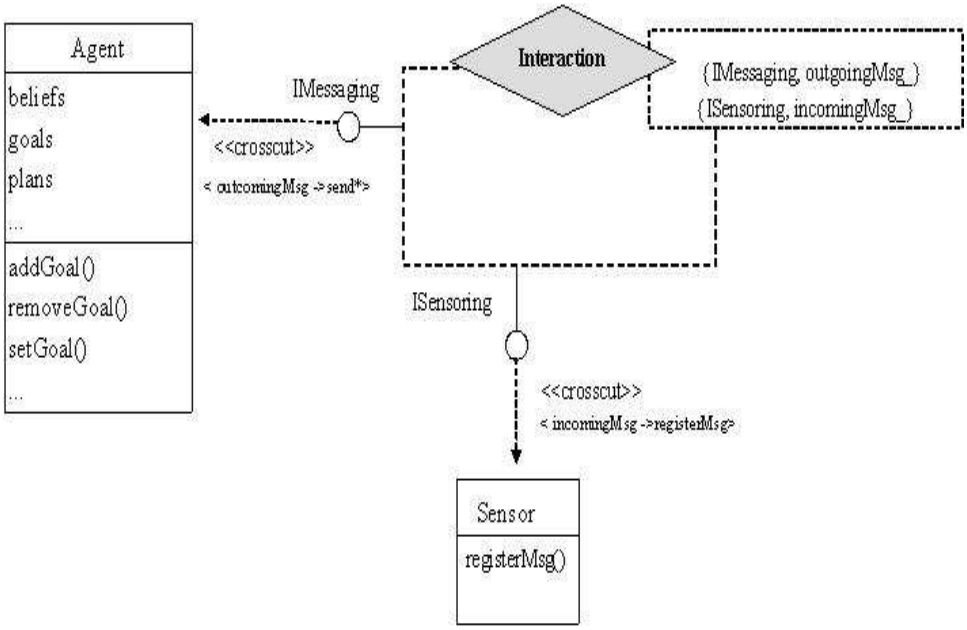

Figura 7.14: Aspecto Interaction afeta Agent e Sensor.

7.3.4 Autonomia

O aspecto Autonomy é usado para capturar de forma clara o protocolo de autonomia. O protocolo de autonomia possui duas dimensões, *controle de execução* e *tomada de decisão*, descritas por meio de duas interfaces transversais: `IThreads` e `IDecisionMaking`. A Figura 7.15 apresenta a visão completa do aspecto Autonomy a Figura7.16 apresenta uma interação aspectual para `makesDecision_()`.

Composição progressiva: associando o aspecto Autonomy à classe Agent

A Figura 7.17 mostra que o aspecto Autonomy afeta a classe Agent. A dependência de requisitos (seta tracejada) especifica que o aspecto Autonomy requer o aspecto Interaction. O último introduz a operação `receiveMsg` na classe Agent, e Autonomy o afeta.

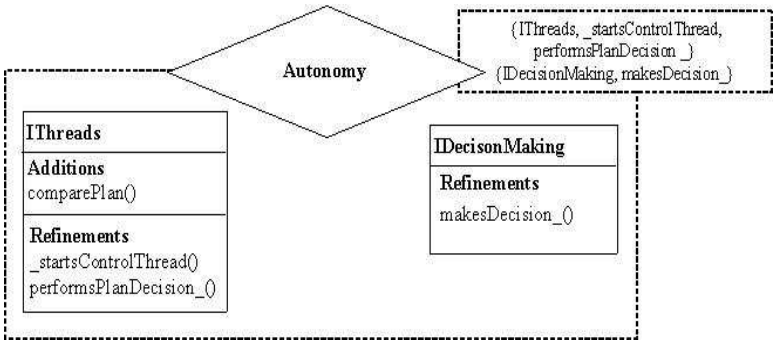

Figura 7.15: A visão estática do aspecto Autonomy.

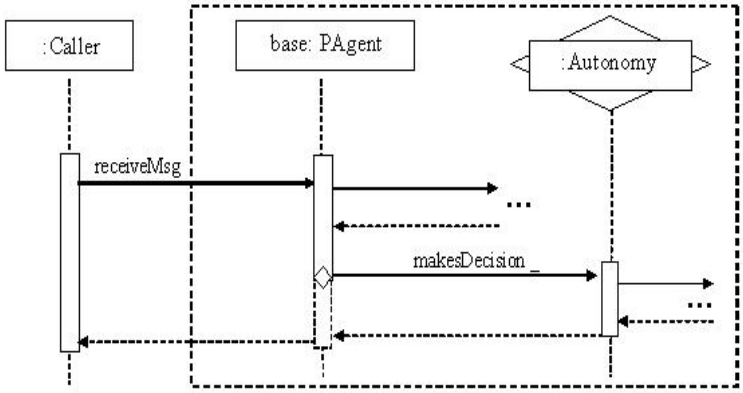

Figura 7.16: Interação aspectual para `makesDecision_()`.

7.3.5 Adaptação

O aspecto Adaptation modulariza o comportamento relacionado ao protocolo de adaptação e recorre a duas interfaces transversais chamadas `IBehaviorAdaptation` e `IKnowledgeAdaptation` a fim de localizar os comportamentos de *adaptação de comportamento* e *adaptação de conhecimento* descritos na Seção 7.1. A Figura 7.18 apresenta a visão completa do aspecto Adaptation.

A Figura 7.19 apresenta uma interação aspectual para `adaptPlan_()`. Ela especifica que `adaptPlan_()` refina a operação base `setGoal` depois de sua execução.

Composição progressiva: associando o aspecto Adaptation à classe Agent

A Figura 7.20 mostra que o aspecto Adaptation afeta a classe Agent. A dependência de requisitos (seta tracejada) especifica que o aspecto

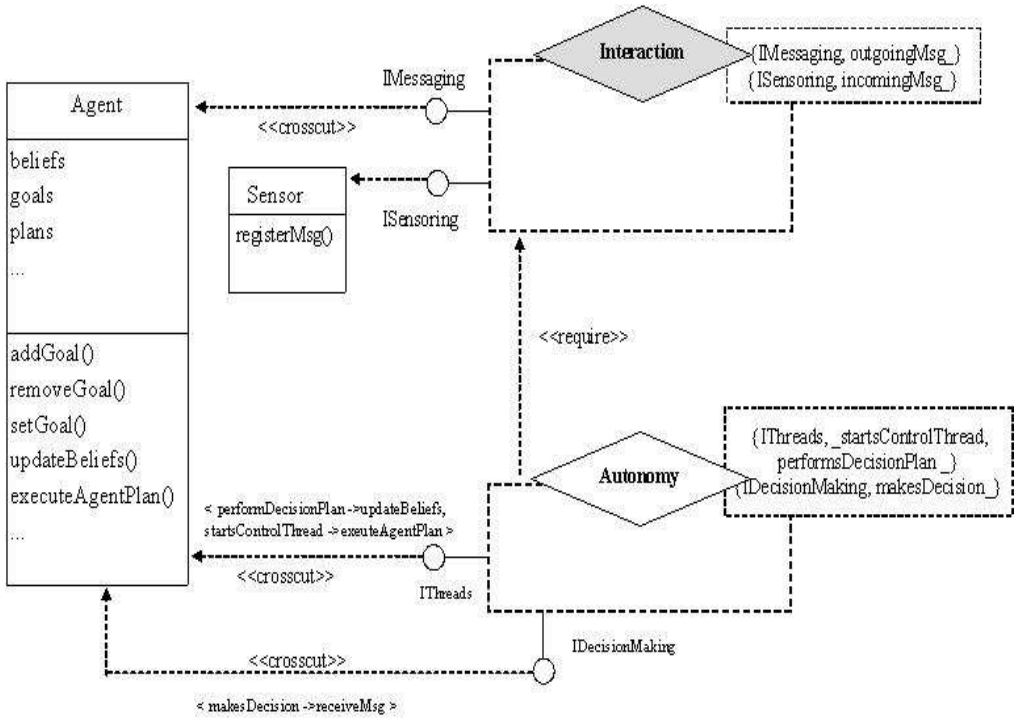

Figura 7.17: Autonomy afeta Agent e requer Interaction.

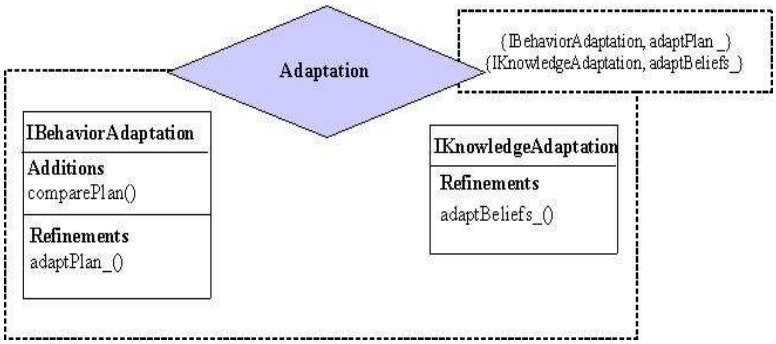

Figura 7.18: A visão estática do aspecto Adaptation.

Adaptation requer o aspecto Interaction. O último introduz a operação `receiveMsg` na classe Agent, e o aspecto Adaptação o afeta.

7.3.6 Agência

A Figura 7.21 apresenta uma perspectiva centrada em base para a classe Agent. O aspecto Interaction, o aspecto Autonomy e o aspecto Adaptation afetam a classe Agent. Autonomy e Adaptation compartilham um ponto de combinação, a operação `receiveMsg`, e Autonomy precede Adaptação, isto é, a operação de *crosscutting* `makesDecision` é executada antes da operação de *crosscutting* `adaptBeliefs`. Como a operação

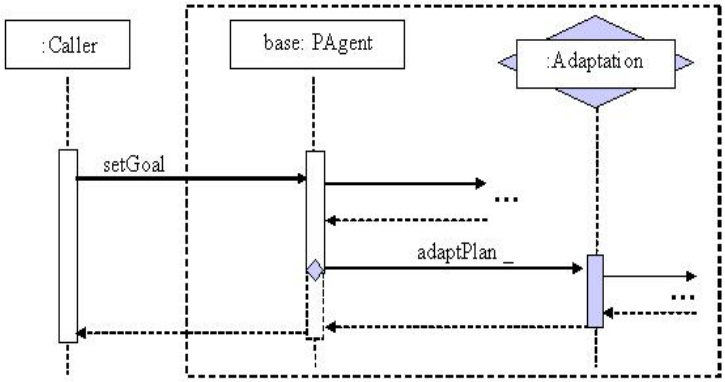

Figura 7.19: Interação aspectual para `adaptPlan_()`.

`receiveMsg` é introduzida na classe Agent pelo aspecto Interaction, tanto Autonomy quanto Adaptation estão relacionados a Interaction por um relacionamento de `requirement`.

7.3.7 Papéis

No SMA Portalware, os agentes de informação e os agentes de usuário possuem papéis colaborativos. Os agentes de informação podem exercer o papel chamador ou respondedor, e os agentes de usuário podem exercer o papel de fornecedor de conteúdo, revisor ou editor.

Aspectos são usados a fim de implementar os papéis colaborativos do agente. Esses aspectos são chamados de *aspectos de papel* (*role aspects*) [54]. Como os agentes podem precisar realizar vários papéis, os diferentes aspectos de papel podem ser associados a eles. Além disso, os aspectos de papel podem afetar a execução de planos colaborativos relacionados [54].

Nesta Seção, apresentamos os papéis chamador e respondedor especificados como aspectos. Os aspectos de papel Caller e Answerer são associados à classe InformationAgent.

7.3.7.1 Caller

`Caller` é um aspecto de papel que afeta os agentes de informação. O aspecto Caller oferece suporte ao protocolo de chamador: ele fornece a um agente de informação a capacidade de (1) enviar (encaminhar) a solicitação de busca ao agente que esteja respondendo e (2) receber e processar o

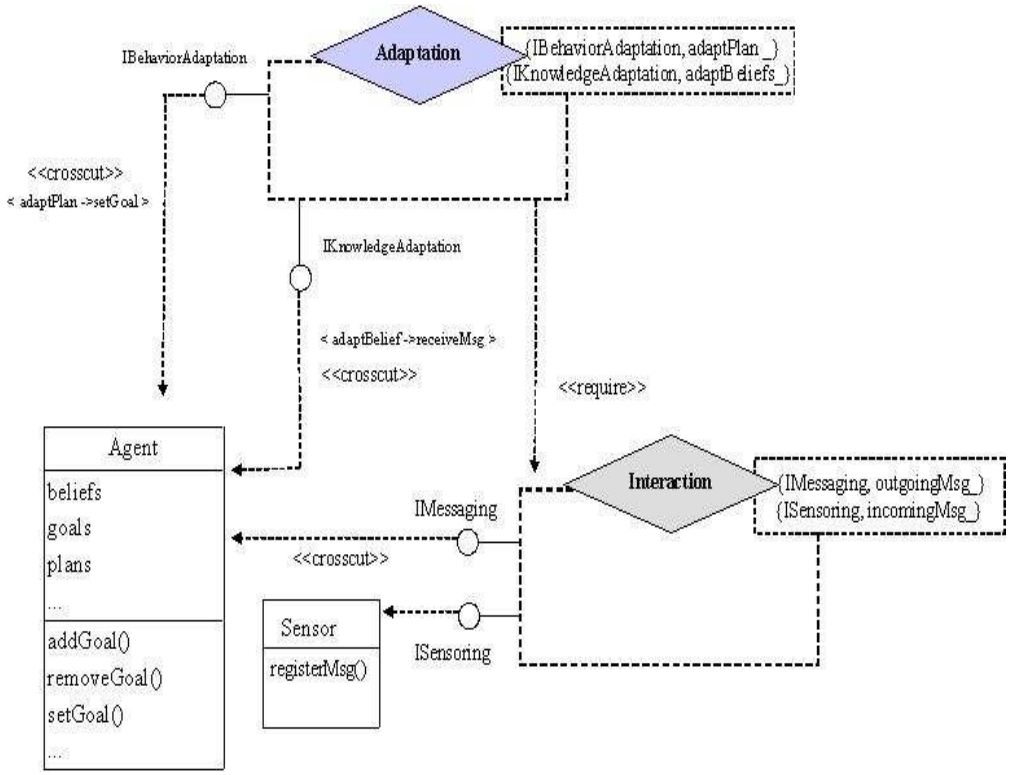

Figura 7.20: Adaptation afeta Agent e requer Interaction.

resultado da busca. A Figura 7.22 apresenta a visão completa do aspecto Caller.

A interface transversal `IAgent` define novas operações no compartimento Additions que devem ser introduzidas na classe InformationAgent. A operação de *crosscutting* chamada `activatesCaller_()` refina a execução do método `search(keyword)`. Ela verifica o resultado da busca de forma que o papel caller possa ser ativado sempre que as informações necessárias não sejam encontradas (Figura 7.23). A interface transversal `Iplan` define uma operação de *crosscutting* chamada `receiveSearchResult_()`. Essa operação refina a execução de `SearchResultReceivingPlan` a fim de oferecer suporta à segunda parte do protocolo chamador.

7.3.7.2 Answerer

`Answerer` é um aspecto de papel que afeta os agentes de informação. O aspecto Answerer oferece suporte ao protocolo respondedor: ele fornece a um agente de informação a capacidade de (1) receber e processar a solicitação de busca do chamador e (2) enviar de volta o resultado da busca. A Figura 7.24 apresenta a visão completa do aspecto Answerer.

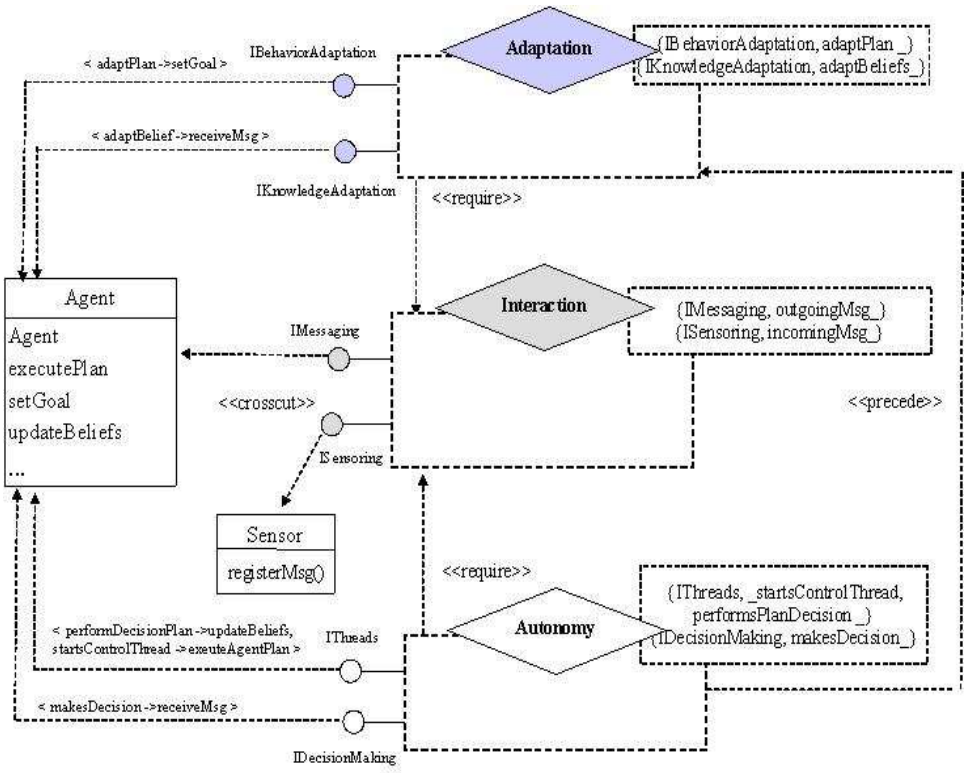

Figura 7.21: Agência revisitada.

A interface transversal `IPlan` define a operação de *crosscutting* chamada `receiveSearchAsk_` que implementa o comportamento de recebimento e processamento da solicitação de busca (1). A interface transversal `IAgent` contém a operação `sendSearchResult`, definida no compartimento Additions que deve ser introduzida na classe InformationAgent. Essa operação implementa o comportamento de envio de volta do resultado da busca (2).

Composição progressiva: associando Caller e Answerer a InformationAgent

A Figura 7.25 mostra que o aspecto Caller e o aspecto Answerer afetam a classe InformationAgent e os planos de colaboração associados.

7.3.8 Colaboração

O aspecto Collaboration implementa a sincronização entre os agentes que participam de uma colaboração (protocolo de coordenação). Ele oferece duas operações de *crosscutting* – `synchronization_` e `unlock_`. A operação `synchronization_` oferece o comportamento para impedir que o agente

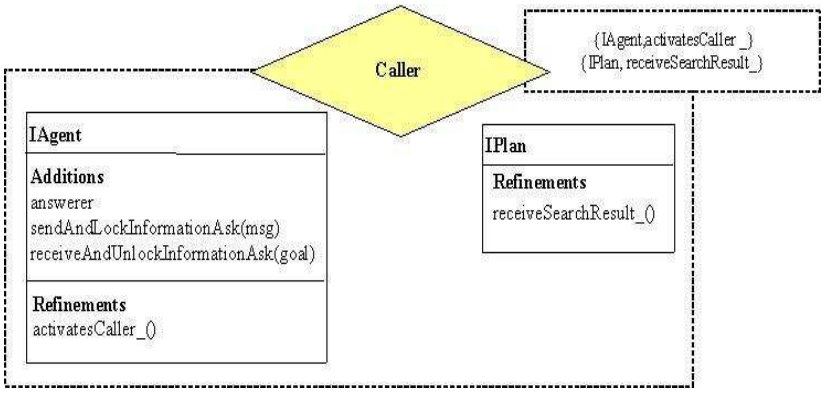

Figura 7.22: A visão estática do aspecto Caller.

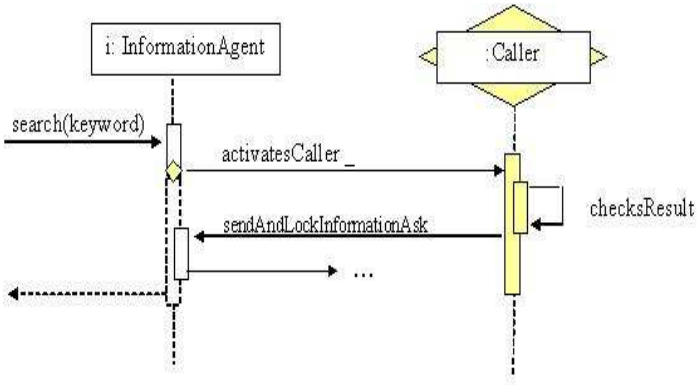

Figura 7.23: Refinamento de `search()`.

envie uma mensagem. A operação `unlock_` oferece o comportamento para desbloquear o agente em espera quando ele receber uma resposta. A Figura 7.26 apresenta a visão completa do aspecto Collaboration.

Composição progressiva: associando Collaboration a InformationAgent

A Figura 7.27 mostra a associação dos aspectos Caller, Answerer e Collaboration a InformationAgent e a subclasses de CollaborationPlan. O aspecto Collaboration afeta duas operações adicionadas a InformationAgent pelos papéis de aspecto. Ele afeta a operação `sendAndAsk*` para bloquear um agente de informação e afetar a operação `receiveAndAsk*` para desbloquear alguns agentes de informação que estão esperando uma resposta. O aspecto Collaboration requer os papéis Caller e Answerer de forma que seu protocolo de coordenação possa ser aplicado às instâncias de InformationAgent.

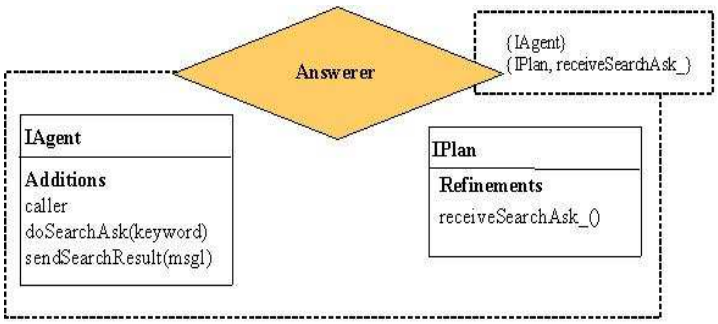

Figura 7.24: Visão estática do aspecto Answerer.

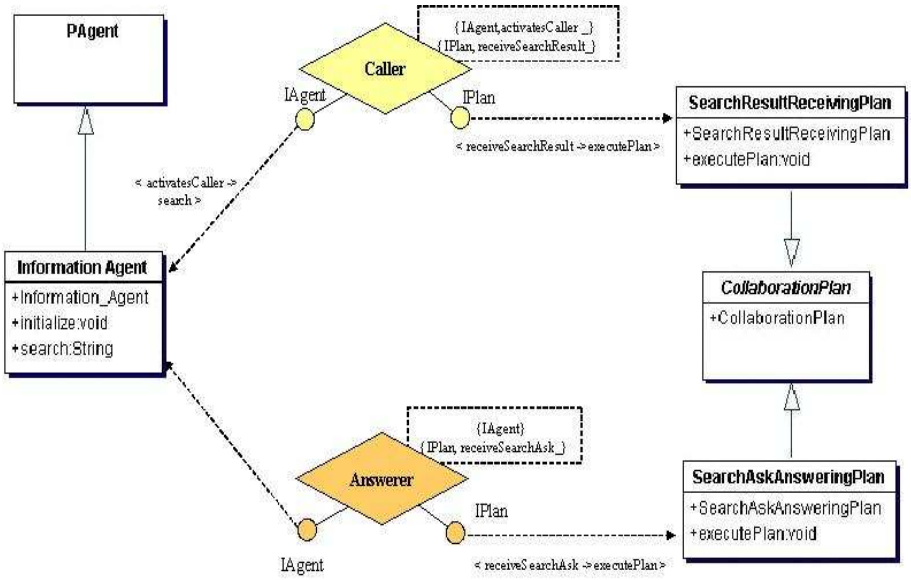

Figura 7.25: Caller e Answerer afetam InformationAgent.

7.3.9 Portalware

A Figura 7.28 apresenta uma visão centrada em base para a classe InformationAgent. Essa visão estática descreve os aspectos que afetam a classe InformationAgent, bem como os aspectos associados à superclasse Agent.

Os diferentes tipos de agentes de software herdam os aspectos de agência associados à superclasse Agent. Como conseqüência, os agentes de informação herdam as características de agência e somente definem seus aspectos colaborativos e serviços específicos.

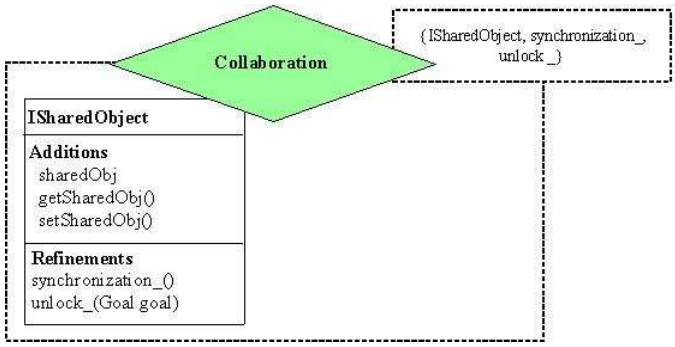

Figura 7.26: Visão estática do aspecto Collaboration.

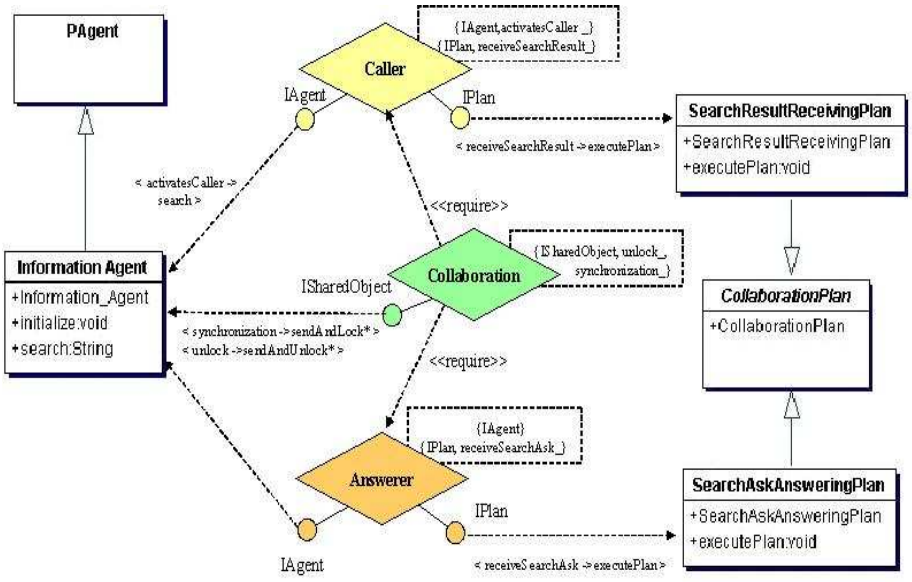

Figura 7.27: Collaboration afeta InformationAgent.

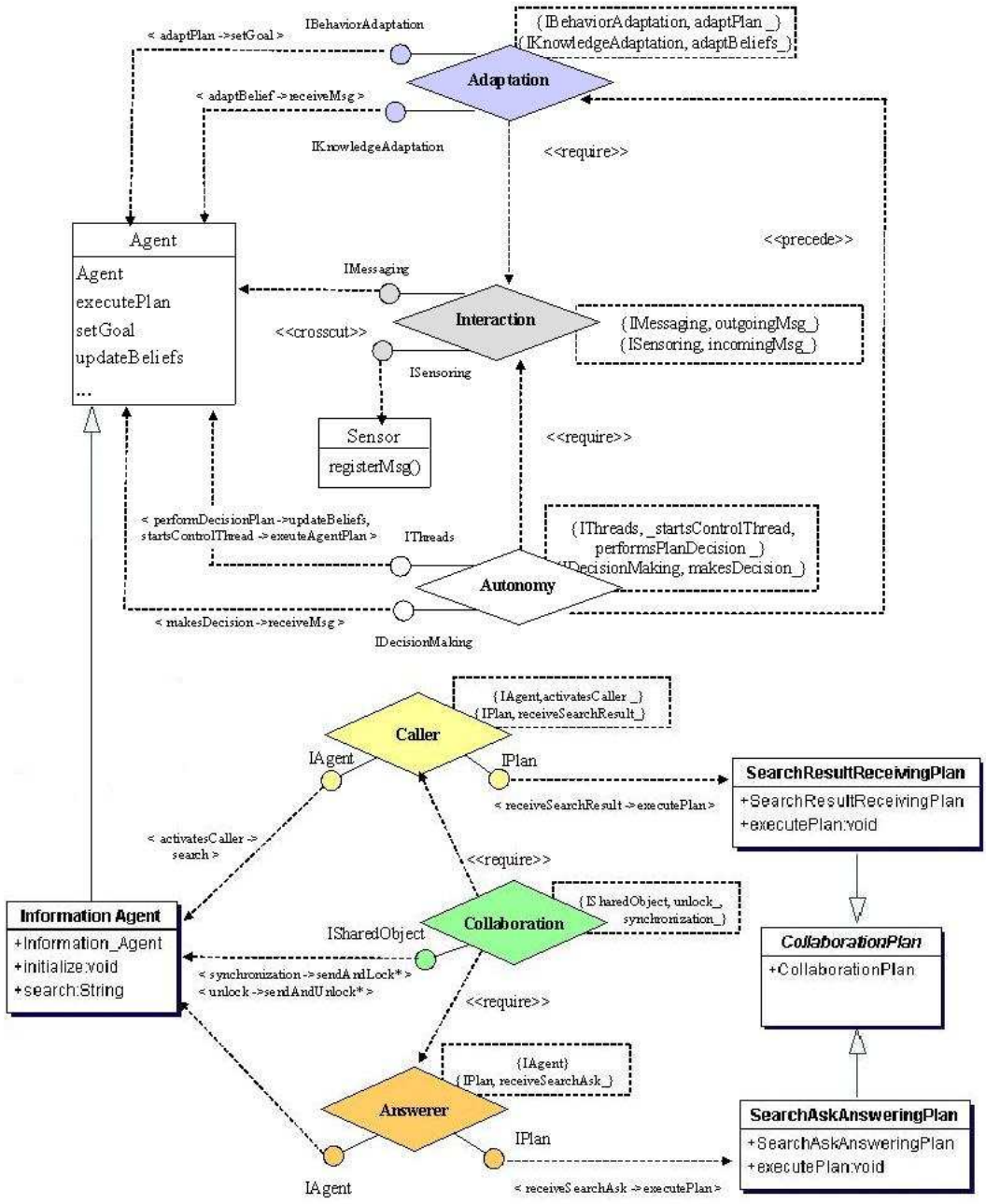

Figura 7.28: Visão estática de InformationAgent revisitada.

As Figuras 7.29, 7.30 e 7.31 apresentam diagramas de processo de combinação que ilustram um cenário em que o agente de informação "alex" busca algumas informações e colabora (como um *caller*) com o agente de informação "chris" (o *answerer*).

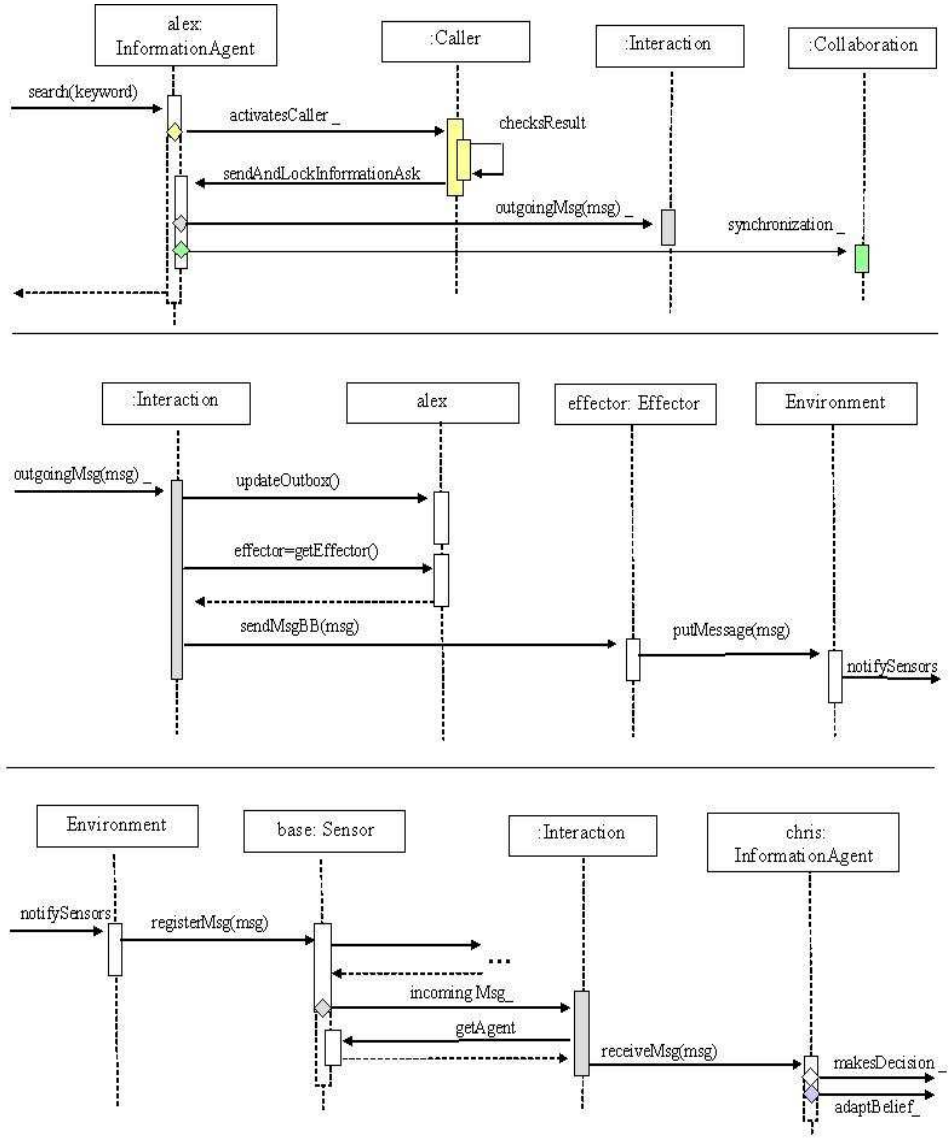

Figura 7.29: Visão dinâmica de Portalware revisitada.

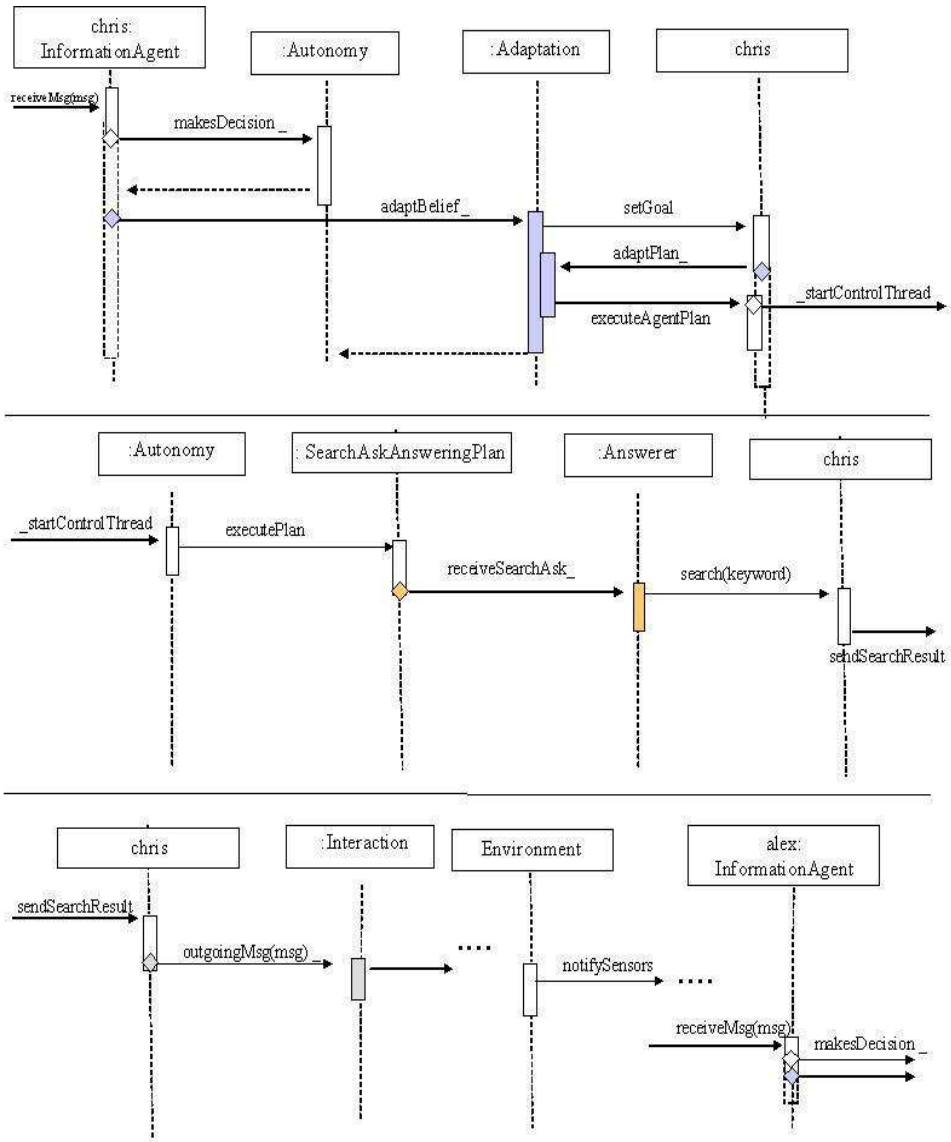

Figura 7.30: Visão dinâmica de Portalware revisitada.

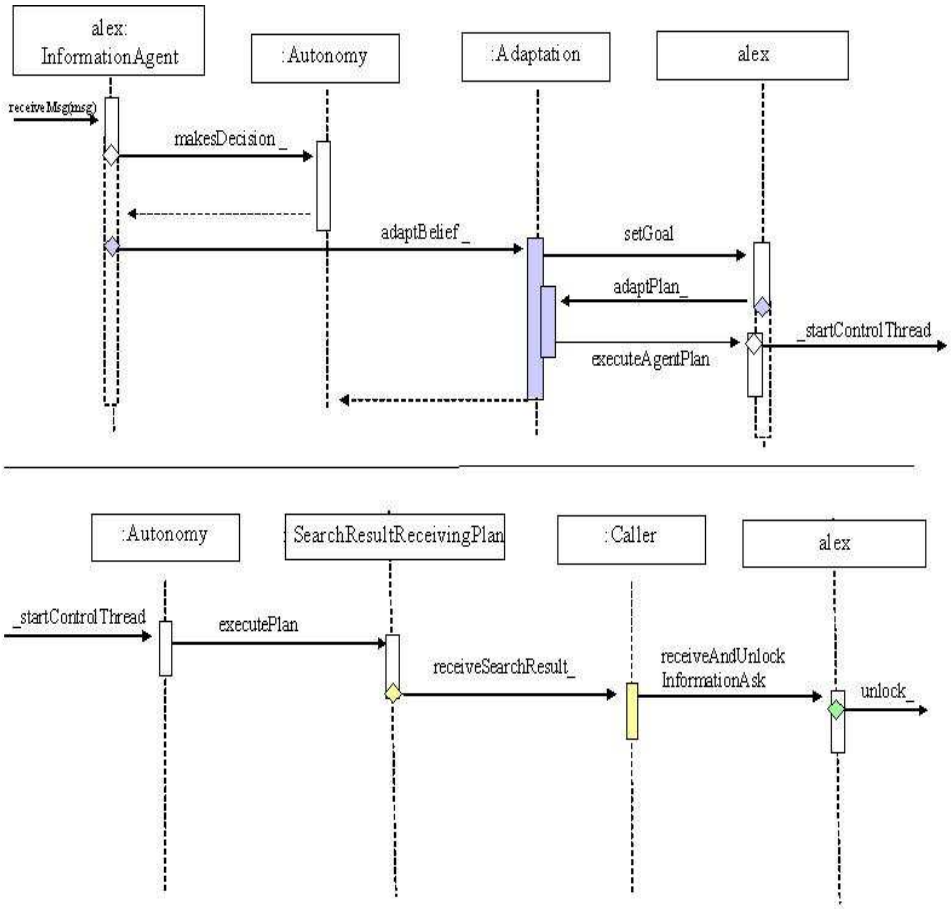

Figura 7.31: Visão dinâmica de Portalware revisitada.

7.4
Trabalhos Relacionados

Nesta Seção, descrevemos os trabalhos de pesquisa realizados no Grupo SoC+Agents [126], na PUC-Rio, que adotaram aSideML como linguagem de modelagem.

7.4.1
Padrões de Projeto Orientados a Aspectos para SMA

A linguagem `aSideML` foi usada para ilustrar a linguagem de padrões proposta em [54], que compreende sete padrões: (1) o padrão Kernel, (2) o padrão Interaction, (3) o padrão Adaptation, (4) o padrão Autonomy, (5) o padrão Role, (6) o padrão Mobility e (7) o padrão Learning.

O problema tratado pela linguagem de padrões envolve o projeto de agentes de software, mas enfatiza a separação de *concerns*: o propósito básico da linguagem de padrões é oferecer suporte à modularização de *concerns* transversais de agência [54].

Para fins de ilustração, apresentamos aqui dois diagramas para o padrão Interaction. Esse padrão e outros padrões são descritos em [54] usando a `aSideML`.

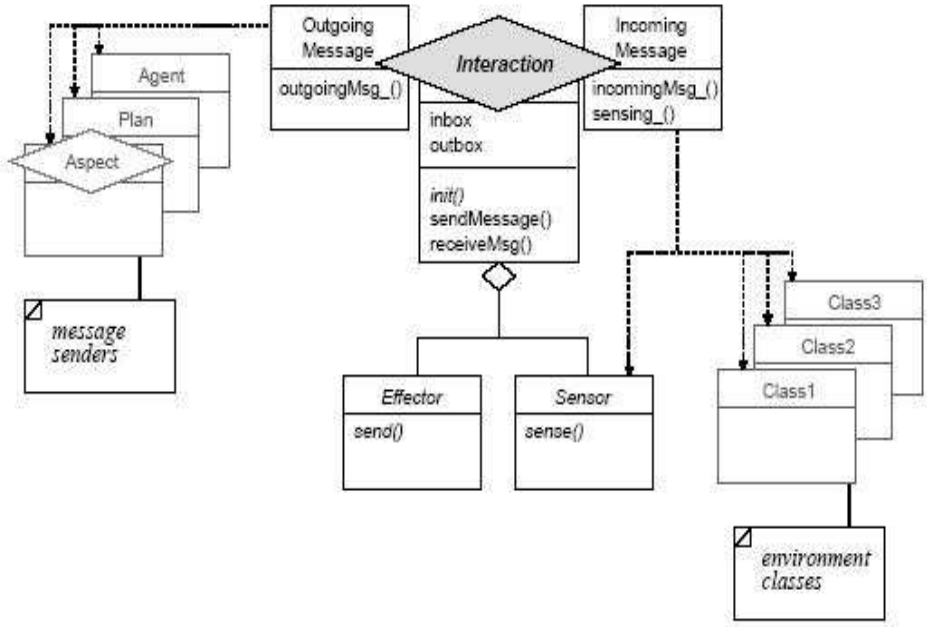

Figura 7.32: Visão estática do padrão Interaction [54].

A Figura 7.32 mostra o padrão Interaction [54]. O padrão é apresentado usando uma perspectiva centrada em aspectos que descreve o aspecto

Interaction, as classes base que ele afeta e outros aspectos relacionados a ele. A organização explícita de características transversais em várias interfaces transversais enfatiza o fato de o aspecto Interaction afetar classes de forma heterogênea. A decomposição da interface do aspecto em duas ou mais subinterfaces promove a previsibilidade da composição e melhora a compreensão.

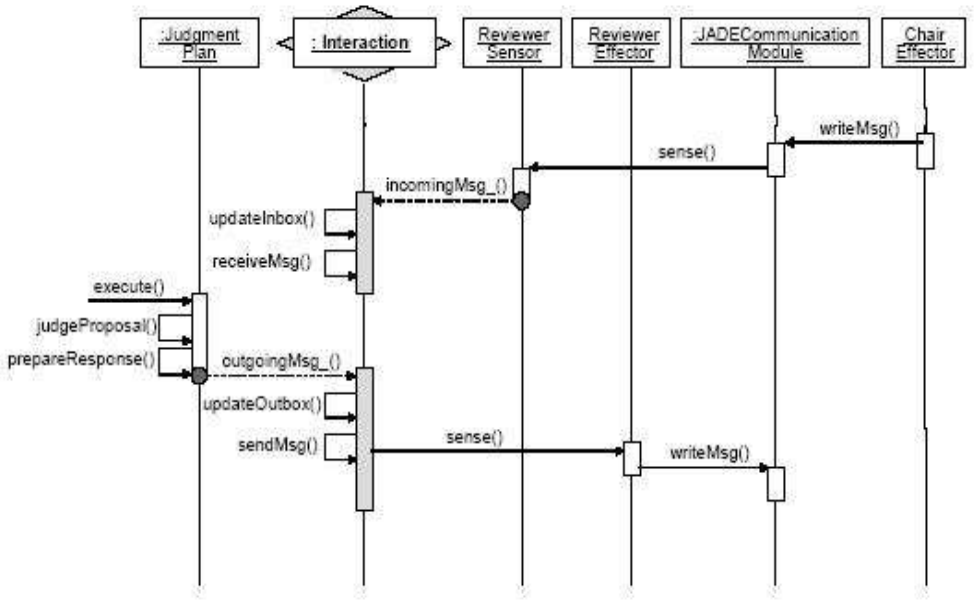

Figura 7.33: Visão dinâmica do padrão Interaction: enviando e recebendo mensagens [54].

A Figura 7.33 apresenta um diagrama de seqüência aspectual que descreve os *pontos de combinação* (execução do método `Sensor.sense` e execução do método `Plan.prepareResponse`) e as chamadas explícitas aos métodos dos aspectos (`incomingMsg_()` e `outcomingMsg_()`). Por meio da notação de `aSideML`, os pontos de combinação são adornados e as instâncias de aspecto são enfatizadas, aumentando a compreensibilidade.

7.4.2 Arquitetura de Agentes Orientada a Aspectos

Kulesza *et al.* estão trabalhando em busca de uma abordagem gerativa para o suporte uniforme da modularização das características de SMAs desde as primeiras etapas de desenvolvimento. Nesse contexto, eles propuseram uma arquitetura de agentes orientada a aspectos [77], que refina o trabalho anteriormente desenvolvido no Laboratório de Engenharia de Software (LES) da PUC-Rio [51, 53].

A arquitetura proposta é composta por dois tipos de componentes: (i) um *componente central* que modulariza as características ortogonais associadas ao *conhecimento de agente* e (ii) os *componentes aspectuais* que separam as propriedades de agência *crosscutting* umas das outras e do componente de conhecimento.

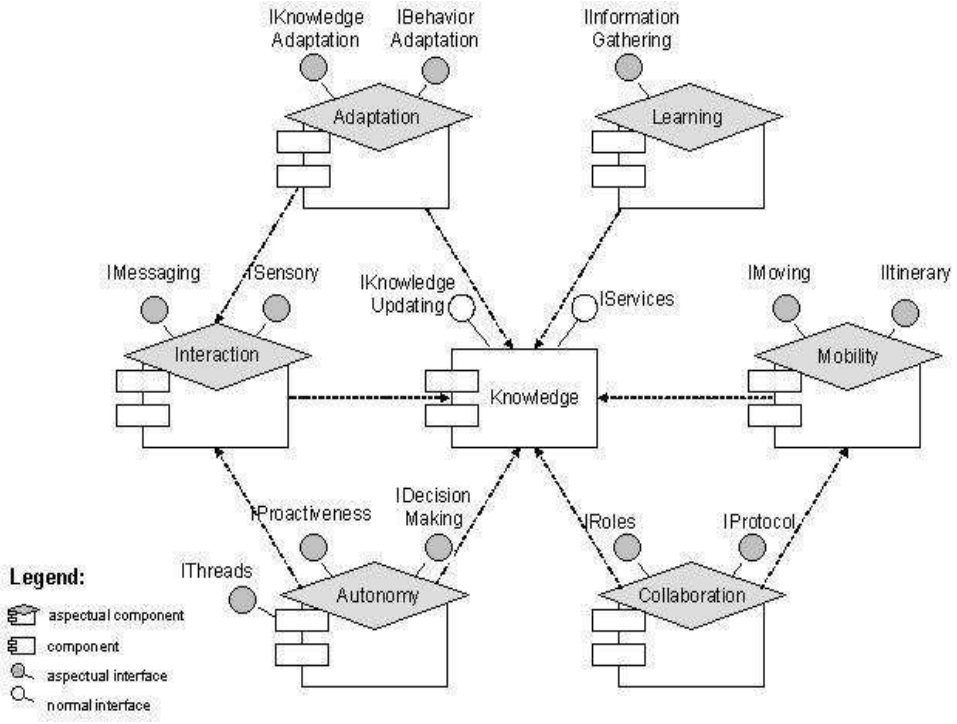

Figura 7.34: Arquitetura de agentes orientada a aspectos [77].

Kulesza *et al.* estenderam a notação de `aSideML` para lidar com os aspectos no nível da arquitetura de software. A Figura 7.34 descreve uma arquitetura em que cada componente possui uma ou mais interfaces. Há dois tipos de interfaces: interfaces regulares (coloridas de branco) e interfaces transversais (coloridas de cinza). As interfaces regulares fornecem serviços a outros componentes. As interfaces transversais especificam como um aspecto no nível arquitetural (ou componente aspectual, usando sua terminologia) afeta outros componentes arquiteturais. A adoção de `aSideML` como uma linguagem de modelagem oferece suporte a uma interação completa entre artefatos arquiteturais e aspectos e classes no nível do projeto.