PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

K2 - UMA ARQUITETURA PARA A ADAPTAÇÃO DE AGENTES DE SOFTWARE AO CONTEXTO

ANA PAULA LEMKE

Tese apresentada como requisito parcial à obtenção do grau de Doutor, pelo programa de Pós Graduação em Ciência da Computação da Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Marcelo Blois Ribeiro

Porto Alegre
2011

Dados Internacionais de Catalogação na Publicação (CIP)

L554K Lemke, Ana Paula
K2 - uma arquitetura para a adaptação de agentes de software ao contexto / Ana Paula Lemke. – Porto Alegre, 2011.
197 f.

Tese (Doutorado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Marcelo Blois Ribeiro.

1. Informática. 2. Sistemas Multiagentes. I. Ribeiro, Marcelo Blois. II. Título.

CDD 006.3

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE TESE DE DOUTORADO

Tese intitulada "**K2 - Uma Arquitetura para a Adaptação de Agentes de Software ao Contexto**", apresentada por Ana Paula Lemke, como parte dos requisitos para obtenção do grau de Doutor em Ciência da Computação, Sistemas de Informação, aprovada em 28/03/2011 pela Comissão Examinadora:

Prof. Dr. Marcelo Blois Ribeiro – Orientador — PPGCC/PUCRS

Prof. Dr. Rafael H. Bordini - — UFRGS

Prof. Dr. Ricardo Choren Noya - — IME

Prof. Dr. Ricardo Melo Bastos – — PPGCC/PUCRS

Homologada em 14/06/11, conforme Ata No. 10 pela Comissão Coordenadora.

Prof. Dr. Fernando Luís Dotti
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P. 32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

“Nosso único patrimônio que realmente faz diferença é o conhecimento”.

Peter Drucker

AGRADECIMENTOS

Ao meu orientador, Prof. Marcelo Blois Ribeiro, por todos os ensinamentos, críticas e conselhos e, mais do que isso, pela constante presença e paciência nas minhas freqüentes "crises de identidade". Saiba que os desafios lançados colaboraram muito com o meu crescimento, tanto pessoal quanto profissional. Espero que possamos continuar trabalhando juntos por muito tempo.

Aos membros da banca, professores Ricardo Choren Noya, Ricardo Bastos e Rafael Bordini, pela aceitação do convite de participação na avaliação deste trabalho.

Ao meu namorado Túlio, agradeço pelo companheirismo, paciência, amor e carinho dedicados ao longo desta jornada. Tenho plena consciência de quão insuportáveis foram alguns momentos...

Ao meu irmão Daniel, que muitas vezes sem entender o porquê de minha dedicação incessante a esta pesquisa, sempre torceu incondicionalmente para que tudo desse certo. Aos primos Aline e Jader, que estão sempre na torcida, me incentivando a seguir em frente.

Ao meu padrinho Gilberto e sua família, por todo o apoio e preocupação com meu bem estar. Obrigada também pela acolhida em Porto Alegre.

A três professoras fantásticas que pude conviver durantes esses muitos anos de vida estudantil: Heloisa Krüger, Suzana Tust e Flávia Braga Azambuja. Obrigada pela amizade, exemplos de vida, conselhos e, principalmente, por sempre acreditarem e investirem em mim.

Aos colegas do antigo CDPe (agora CePES) e aos membros do grupo de pesquisa ISEG por terem compartilhado muitos momentos agradáveis e também estressantes durante o período do doutorado. Ao colega Maurício Escobar um agradecimento especial por toda a ajuda na implementação dos exemplos de uso que estão sendo apresentados neste trabalho.

Aos novos colegas do IFRS-Campus Feliz, pelo acolhimento nestes primeiros meses de trabalho.

Aos amigos, novos e antigos, pelo companheirismo, motivação e compreensão das constantes ausências. Aqui, um agradecimento especial aos "amigões" Rejane e Leandro, que estão sempre por perto nas horas certas e me desculpam quando eu me esqueço de seus aniversários...

Ao Convênio Dell-PUCRS por viabilizar a bolsa de estudos durante os quatro anos de doutorado.

Ao Programa de Pós-Graduação em Ciência da Computação e a todos os professores dos quais pude conviver durante estes quatro anos.

A Deus, pela sabedoria, persistência e oportunidades que tem posto em minha vida.

E, principalmente, aos meus queridos pais, Heroldino e Elaine, por me permitirem sonhar e compartilharem meus sonhos, sempre na torcida para que tudo acabe bem. A vocês dedico tudo, sempre.

K2 - UMA ARQUITETURA PARA A ADAPTAÇÃO DE AGENTES DE SOFTWARE AO CONTEXTO

RESUMO

A tecnologia de agentes é cada vez mais citada como uma abordagem atrativa para o desenvolvimento de aplicações em ambientes pervasivos [Gun08b]. No entanto, a maioria das plataformas disponíveis apóia apenas a criação de agentes capazes de lidar com um conjunto limitado de situações (os agentes precisam ser reprogramados quando se deparam com situações não previstas). A dificuldade de produzir software para ambientes complexos como o pervasivo vem justamente do fato de o projetista não poder prever todas as circunstâncias em que a aplicação poderá ser usada, e tomar todas as decisões em tempo de projeto. Assim, é necessário desenvolver agentes que consigam aprender e se adaptar de forma a satisfazer as condições de um novo ambiente, ou de um novo *contexto*. Considerando essa necessidade, o objetivo desta pesquisa é propor uma arquitetura para a criação de agentes adaptativos ao contexto - a arquitetura **K2**. De fato, a idéia é disponibilizar uma arquitetura que permita a modificação de partes de elementos estruturais de um agente de software, adaptando o seu comportamento e estrutura de acordo com as mudanças percebidas no contexto em que o agente está inserido. Uma das características da arquitetura proposta é a separação do comportamento adaptativo do comportamento padrão do agente. Para tanto, são criados elementos chamados de adaptadores, cuja implantação é feita com programação orientada a aspectos. Para demonstrar a aplicabilidade da arquitetura desenvolvida, três exemplos de uso são apresentados.

Palavras chave: agentes de software, consciência de contexto, adaptação ao contexto.

K2 - AN ARCHITECTURE FOR ADAPTING SOFTWARE AGENTS TO CONTEXT

ABSTRACT

Agent technology is increasingly seen as an attractive approach to develop applications for pervasive environments [Gun08b]. However, many existing agent platforms support only the development of agents able to deal with a limited set of situations (the agents need to be reprogrammed when faced with unexpected situations). Software development for complex environments such as pervasive environments is difficult since the developers cannot predict every possible execution context the application will have at design time. So, it is necessary to develop software agents capable of learning and adapting to meet the conditions of a new environment, or a new *context*. Considering this issue, this thesis proposes an architecture to create context adaptative agents – the **K2** architecture. The idea is to provide an architecture that supports the modification of structural elements of a software agent, adapting its behavior and structure according to perceived changes in the context. One of the characteristics of the proposed architecture is the separation between adaptative and non-adaptative (default) agent behaviors, which will be accomplished by using elements named adaptors, whose deployment is supported by aspect-oriented programming. Three examples illustrate the architecture feasibility and applicability.

Keywords: software agents, context awareness, context adaptation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

EMA – *Exponential Moving Average*

GPS – *Global Positioning System*

OMG – *Object Management Group*

OO – Orientado a Objetos

OWL – *Ontology Web Language*

PDA – *Personal Digital Assistant*

POA – Programação Orientada a Aspectos

SGBD – Sistema Gerenciador de Banco de Dados

SMA – Sistema Multiagentes

UML – *Unified Modeling Language*

SUMÁRIO

1. INTRODUÇÃO

Indivíduos, conscientemente ou não, além de seus conhecimentos prévios, utilizam informações de contexto para tomar decisões, para realizar ações e conduzir conversações. A habilidade de perceber o contexto e agir de acordo com esta percepção, no entanto, não é inerente a sistemas computacionais tradicionais. Na maioria das vezes, os sistemas computacionais não conseguem lidar com situações não previstas: toda ação que um computador executa deve ser antecipadamente explicitada, planejada e codificada por um programador. Se por um lado é conveniente ver os computadores como servos obedientes, por outro lado há um grande número de aplicações que requerem sistemas capazes de decidir por si só o que fazer para atingir determinados objetivos [Wei99].

Em Ciência da Computação, a utilização da noção de contexto no desenvolvimento de aplicações conscientes de contexto é consideravelmente recente - o termo está vinculado à área desde a década de 90 - e geralmente remete a pesquisadores que trabalham com computação pervasiva e ubíqua [Dou04]. A computação pervasiva e a computação ubíqua englobam a proposta de um novo paradigma computacional que visa disponibilizar ao usuário acesso computacional de modo invisível (sem que o usuário conheça a tecnologia para desfrutar de seus benefícios) e onipresente (a tecnologia deve estar disponível em qualquer lugar e a qualquer hora) [Sat01]. Devido aos aspectos dinâmicos e à natureza aberta dos ambientes pervasivos, há a possibilidade de flutuações na demanda e oferta de recursos, o que torna necessário o uso de estratégias de adaptação para manter serviços e dispositivos operando sem grandes intervenções do usuário [Sat01].

Contexto, de acordo com Dey e Abowd [Dey99], pode ser definido como “qualquer informação que pode ser usada para caracterizar a situação de uma entidade, sendo que uma entidade pode ser uma pessoa, um lugar ou um objeto considerado relevante para a interação entre um usuário e a aplicação, incluindo os próprios usuário e aplicação”.

Embora a utilização da noção de contexto seja recente, a adaptação de sistemas ao contexto ou ambiente em que estão inseridos já é estudada há muitos anos em outras áreas da Ciência da Computação, como em Sistemas Multiagentes (SMAs). SMAs geralmente são considerados complexos em relação a sua estrutura e funcionalidade [Wei96]. Em muitas aplicações, até mesmo naquelas em que o ambiente parece extremamente simples, é difícil ou mesmo impossível, determinar corretamente, em

tempo de projeto, os possíveis comportamentos e atividades realizados pelos SMAs [Wei96, Mar06, Gun08a]. Para fazer esse tipo de previsão, seria necessário ter conhecimento, por exemplo, de quais os requisitos do ambiente que poderiam aparecer no sistema em tempo de execução e também de como os agentes disponíveis no sistema iriam interagir para atender a esses requisitos. De acordo com Simpkins e colaboradores [Sim08], diferentemente dos programas tradicionais, agentes de software operam em ambientes que são muitas vezes percebidos de forma parcial e que mudam constantemente. A percepção incompleta unida às constantes mudanças ocorridas no ambiente cria uma forte necessidade de aprendizado e/ou adaptação (alguns autores consideram que a adaptação é “coberta” pelo aprendizado, não atribuindo conceitos distintos aos termos).

Segundo Weiss [Wei96], o aprendizado em SMAs (assim como o aprendizado humano [Jar05]) é composto por duas categorias principais: aprendizado de agentes e aprendizado multiagentes. No aprendizado isolado (ou de agentes) agentes adquirem novos conhecimentos de forma individual e completamente independente de outros agentes. Já no aprendizado interativo (ou multiagentes), é necessária a presença e interação de múltiplos agentes. Nesse caso, o conhecimento é adquirido através da troca de informações, pelo compartilhamento de suposições, pelo desenvolvimento de visões comuns do ambiente, entre outros. A presença de um grande número de agentes, o aumento de complexidade de seus comportamentos, a observação parcial do ambiente e a adaptação simultânea de comportamentos em múltiplas entidades fazem do processo de aprendizagem em agentes ainda hoje um desafio [Gun09b].

A definição e o uso de contexto é um aspecto importante em se tratando de aprendizagem e adaptação, uma vez que, segundo Klein e co-autores [Kle08], para se adaptar, um sistema precisa ter consciência de si próprio (autoconsciência) e consciência do ambiente em que está inserido (ou consciência de seu *contexto*). Alguns autores, como Harroud e Karmouch e Amara-Hachmi [Har05; Ama06], tratam contexto do ponto de vista de entidades móveis, ou seja, um agente muda de contexto quando necessita, por exemplo, migrar de um PDA para um celular. Outros autores já trabalham com o conceito de informações contextuais (como os autores dos artigos [Cha06, Lee07, Kap08, Wey08, Sim08]). Essas informações englobam localização geográfica, interesses, atividades comumente executadas pelo indivíduo, objetos próximos, entre outros.

Na literatura, soluções têm sido propostas para facilitar o uso de contexto e para aumentar o desenvolvimento de aplicações baseadas em contexto[1]. Em geral, o foco ainda tem sido considerar situações pré-definidas e métodos de representação de contexto diretos como meio para desenvolver aplicações ou distribuir serviços baseados em contexto (como já indicado por Khedr em [Khe05]). Atualmente, não há uma forma difundida e aceita para representar o contexto[2], o que deixa os desenvolvedores livres para desenvolver esquemas *ad hoc* e limitados para registrar e manipular informações contextuais. Em adição, não há métodos adequados para raciocínio sensível ao contexto, o que permitiria a adaptação dinâmica do funcionamento de um sistema [Gon07, Cha06]. De acordo com Rashid e La [Ras04], "adaptação é sempre uma característica crucial de aplicações na computação pervasiva". Os autores atribuem essa dificuldade principalmente ao fato de a adaptação afetar múltiplos elementos (dispositivos, serviços, e outros) em um ambiente.

A necessidade de contexto, no entanto, é clara: ele é usado tanto para determinar quais serviços prover para os usuários que estão tentando obter informações de serviços em uma variedade de situações quanto para prover informações ou dados de entrada para uma possível adaptação do sistema.

Permitir aos agentes analisar o contexto em que estão inseridos, mudando seus comportamentos quando necessário, possibilita que essas entidades utilizem de fato a sua habilidade de adaptação. Por esse motivo, esta pesquisa tem como foco o estudo de aspectos relacionados à adaptação de agentes de software, considerando o contexto no qual o agente está inserido e eventos internos ocorridos em sua arquitetura. Como resultado da pesquisa, será apresentada uma nova arquitetura para a criação de agentes adaptativos ao contexto - a arquitetura **K2**. O nome **K2** foi inspirado na sonoridade do acrônimo formado a partir das iniciais de *Context-Aware Adaptive Agent* (C-A AA).

Esta pesquisa é relevante no sentido de estudar a adaptação de agentes ao contexto sob uma perspectiva prática. Na literatura, foram encontradas várias iniciativas de pesquisas em adaptação de agentes. Algumas dessas iniciativas, como as descritas em [Bar10, Aru10, Sim08, Ima04, Sar04, Oza04], incorporam algoritmos e técnicas da comunidade de aprendizagem de máquina em agentes, possibilitando assim algum

[1] Um levantamento sobre o uso de informações contextuais em sistemas computacionais, datado de 2007, pode ser encontrado em [Lem07b].

[2] De fato, existem *workshops*, como o CoMoRea (patrocinado pela IEEE), realizados anualmente para a apresentação e discussão de abordagens para a modelagem e raciocínio sobre informações contextuais.

comportamento “inteligente” nessas entidades. Outras iniciativas, por sua vez, discorrem sobre novas abordagens para permitir que os agentes modifiquem seus comportamentos frente a novas situações [Wey08, Spl03, Amo09, Gun08a]. Além disso, há uma vasta gama de trabalhos relacionados a sistemas adaptativos ao contexto, cujas implementações seguem diferentes padrões arquiteturais, como arquiteturas orientadas a serviços, baseadas em componentes, sistemas multiagentes, e outras [3]. Inclusive, há trabalhos (como [Lac09] e [Lee10]) que utilizam a abordagem de agentes para adaptar sistemas com outros padrões arquiteturais (em [Lac09], por exemplo, agentes de software são utilizados para interpretar as descrições de componentes para que esses possam ser adaptados). Por fim, há também trabalhos (como [Jia03]) que propõe a construção de sistemas multiagentes adaptativos, onde o próprio sistema se reorganiza adicionando ou removendo agentes, redefinindo novos protocolos de interação e assim por diante (nesse caso, o sistema é considerado adaptativo, mas os agentes que o compõe não).

Mesmo com todos estes trabalhos, vários autores destacam que, para as aplicações conscientes de contexto adaptativas ganharem impulso (sejam elas baseadas em agentes ou não), devem ser disponibilizados métodos de desenvolvimento e ferramentas apropriadas [Kle08, Mik06, Sit07]. Também, há poucas pesquisas relacionadas a agentes sensíveis ao contexto (*context-aware agents*) e adaptação baseada em contexto em agentes. Em [Gan04] é apresentado um trabalho sobre sensibilidade ao contexto e privacidade em aplicações utilizando agentes de software. No referido trabalho, as respostas as consultas são elaboradas de acordo com regras de privacidade (que indicam o que pode ser respondido dependendo do contexto em que o indivíduo está inserido). Modificar as respostas é, de maneira simplificada, adaptar o comportamento externo do agente (que será observado pelo usuário), de acordo com o contexto. O objetivo da pesquisa aqui proposta é, mais do que adaptar respostas, adaptar o comportamento e estrutura do próprio agente.

Nos grandes desafios da pesquisa em computação 2006-2016[4] (estudo apresentado no contexto do Brasil pela Sociedade Brasileira de Computação - SBC, inspirado em iniciativas internacionais como “*Grand Research Challenges in Computing*”, apoiada pela *National Science Foundation*), comenta-se que um dos grandes desafios técnicos e científicos envolvidos com a gestão da informação em grandes volumes de

---

[3] Em [Awa09], Awang e co-autores apresentam uma avaliação comparativa de abordagens para a adaptação de software. Segundo os autores, as quatro abordagens mais conhecidas para adaptação de software são as baseadas em arquiteturas, componentes, *middlewares* e agentes.

[4] O documento pode ser capturado em http://143.54.83.4/ArquivosComunicacao/Desafios_portugues.pdf

dados distribuídos é a definição e uso da noção de contexto para a recuperação de informação. Também, fala-se sobre o estudo de infra-estruturas adaptáveis e inteligentes para o processamento distribuído de informações. Neste sentido, a pesquisa realizada visa colaborar tanto na definição de um modelo para categorização das informações contextuais quanto no desenvolvimento de uma arquitetura para auxiliar na criação de entidades adaptáveis e inteligentes.

1.1. Questão de Pesquisa

Uma vez apresentado e justificado o foco de estudo, introduz-se a questão de pesquisa utilizada: "*Como modificar comportamento e estrutura de um agente de software com base em seu contexto de execução e em eventos ocorridos em sua arquitetura interna?"*. Por adaptar comportamento, entende-se a ativação de determinada estrutura de um agente, sem que haja alteração no seu conjunto de estruturas internas. Já por adaptação estrutural, entende-se a adição, remoção ou atualização das estruturas ou informações constituintes de um agente.

1.2. Metodologia de Pesquisa

Mesmo que as informações sobre o contexto já possam estar disponíveis às aplicações, ainda não se sabe ao certo qual o impacto que essas informações exercem sobre o sistema (como as informações devem ser interpretadas para a adaptação do sistema). Neste sentido, esta pesquisa se caracterizou como um estudo exploratório, sendo a principal estratégia de pesquisa utilizada, de acordo com a classificação de Oates [Oat06], *projeto e criação* (do inglês *design and creation*).

Pesquisas guiadas pela estratégia *projeto e criação* têm foco no desenvolvimento de novos produtos de TI (Tecnologia da Informação), também chamados de artefatos [Oat06]. Geralmente (como no caso desta pesquisa), o novo produto de TI é um sistema baseado em computador, mas ele também pode ser algum elemento relacionado ao processo de desenvolvimento, como um modelo ou um método [Oat06]. Cabe salientar que, para projetos que seguem essa estratégia serem considerados de fato uma pesquisa (e não somente uma prova de conhecimento técnico), eles devem demonstrar, segundo Oates, "qualidades acadêmicas, como análise, discussão, justificação e avaliação crítica" [Oat06].

Para o desenvolvimento da arquitetura proposta, foi utilizado o processo de desenvolvimento prototipal. Já para a avaliação da arquitetura, foram feitos testes de software (com respeitos aos aspectos técnicos) e provas de conceito (para ilustrar o comportamento apresentado por agentes adaptativos ao contexto desenvolvidos com a arquitetura **K2**).

1.3. Contribuições da Tese

Do ponto de vista de contribuição teórica, esta tese contribui para a Engenharia de Software em dois sentidos: (1) na definição de um modelo de referência para a classificação das informações contextuais; e (2) na criação de primitivas para o mapeamento da influência do contexto na adaptação de agentes de software. Além disso, contribui no sentido de melhorar os estudos atualmente existentes, disponibilizando uma arquitetura para a criação de agentes adaptativos ao contexto desenvolvida com base em estudos qualitativos e quantitativos.

A arquitetura proposta satisfaz uma série de requisitos[5] que, de acordo com vários trabalhos encontrados na literatura, são desejáveis em uma arquitetura para o desenvolvimento de aplicações adaptativas ao contexto. Em relação ao gerenciamento das informações contextuais, são características da arquitetura desenvolvida: (i) a utilização de um modelo de referência para a categorização das informações contextuais, permitindo a indicação de propriedades, origem e tipos de informações contextuais; e (ii) a criação de entidades capazes de interpretar as informações contextuais, deduzindo novas informações com base no contexto já conhecido. Os estudos relacionados à adaptação de agentes ao contexto identificados ao longo da pesquisa (como apresentado na Seção 2.2 e sumarizado na Seção 2.3) não costumam dar grande ênfase ao gerenciamento das informações contextuais e são poucos os casos onde há um metamodelo de contexto.

Em relação ao processo de adaptação propriamente dito, a arquitetura fornece meios para o desenvolvimento de agentes capazes de adaptar comportamento e estrutura, sendo que o conjunto de adaptações realizáveis (encapsuladas por adaptadores) pode ser ampliado durante o ciclo de vida do agente. Por último, a arquitetura possibilita a verificação da satisfação obtida com a adaptação realizada. Embora esses requisitos já tenham sido citados na literatura, não se tem conhecimento de

[5] Na Seção 2.3 é apresentada a lista completa de requisitos com as respectivas referências bibliográficas.

outra arquitetura que contemple a todos sob a mesma perspectiva da proposta neste trabalho.

1.4. Organização da Tese

Este documento está dividido da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica necessária para um bom entendimento do trabalho, onde são dadas as definições dos conceitos relacionados (como contexto e adaptação) e é apresentado o estado da arte em agentes adaptativos ao contexto; o Capítulo 3 descreve o modelo conceitual da arquitetura proposta, enfatizando também o metamodelo para classificação das informações contextuais utilizado e o processo de identificação dos locais passíveis de adaptação na arquitetura de um agente; o Capítulo 4 apresentada a implementação da arquitetura, indicando as técnicas utilizadas para a construção de suas principais estruturas; o Capítulo 5 apresenta três exemplos de uso que demonstram a aplicabilidade da arquitetura em diferentes domínios de negócio; e, por fim, no Capítulo 6 são apresentadas as considerações finais seguidas das referências bibliográficas.

2. AGENTES ADAPTATIVOS AO CONTEXTO

Antes mesmo do desenvolvimento de uma aplicação adaptativa ao contexto, pesquisadores e desenvolvedores se deparam com uma limitação da área: a falta de definições e diretrizes para auxiliá-los em suas tarefas. De fato, atualmente não há sequer uma definição clara e consensual do que seria contexto e a utilização das informações contextuais para a adaptação dos sistemas é, normalmente, um ponto obscuro na literatura disponível. Desta forma, muitos pesquisadores acabam trabalhando com uma idéia geral de contexto e adaptação, sem considerar que essa noção vaga pode tornar-se insuficiente para auxiliar, por exemplo, na análise dos tipos de dados que devem ser coletados e armazenados pelo sistema e na verificação dos mecanismos necessários para processar tais dados.

Para desenvolver aplicações adaptativas ao contexto de forma eficaz, primeiro é necessário entender os termos relacionados à área e conceituá-los apropriadamente. Assim, na próxima seção será apresentada uma revisão na literatura buscando contextualizar o uso da noção de contexto em sistemas computacionais. Em resumo, são debatidos os conceitos de contexto, adaptação e sistemas adaptativos ao contexto. Em cada subseção, no caso de não haver consenso na literatura acerca do conceito debatido, é enfatizada a definição que reflete a opinião dos autores deste trabalho.

Na Seção 2.2 discorre-se sobre uma série de estudos relacionados ao desenvolvimento de agentes adaptativos ao contexto encontrados na literatura. Para a apresentação dos estudos relacionados, utilizou-se a classificação de McKinley e co-autores [McK04], que indicam a existência de duas abordagens gerais para o desenvolvimento de sistemas adaptativos: a abordagem parametrizada e a composicional. Como alguns dos estudos relacionados utilizam programação orientada a aspectos para a implementação da solução proposta (tecnologia que também foi utilizada na arquitetura **K2**), no Apêndice A são descritos os conceitos fundamentais da programação orientada a aspectos, com ênfase na linguagem AspectJ [Ecl11b].

Por último, na Seção 2.3, além das considerações sobre o capítulo, são descritos vários requisitos que uma arquitetura para o desenvolvimento de aplicações adaptativas deveria contemplar.

2.1. Definições gerais

2.1.1. Contexto

Na literatura de Ciência da Computação[6] há uma grande variedade de definições para contexto. Muitas dessas definições, no entanto, utilizam sinônimos da própria palavra ao invés de conceituá-la: é o caso das definições que descrevem contexto como ambiente ou situação. Em [Hen02], por exemplo, tem-se que contexto "refere-se às circunstâncias ou situações em que uma tarefa computacional ocorre". Segundo Dey [Dey00], definições que utilizam apenas sinônimos da palavra contexto são extremamente difíceis de aplicar na prática.

Outras definições de contexto são, na verdade, enumerações de possíveis informações contextuais. De acordo com Chen e Kotz [Che00], não satisfeitos com uma definição geral de contexto, muitos pesquisadores tentam defini-lo através de exemplos de informações contextuais. Ryan *et al.* [Rya98] definem contexto como a localização do usuário, o ambiente, a identidade e o tempo. Já Chen e co-autores [Che03] acreditam que contexto seja "a compreensão de uma localização, seus atributos de ambiente (volume do som, intensidade da luz, temperatura e movimento) e as pessoas, dispositivos, objetos e agentes de software contidos nesse ambiente". Em alguns de seus trabalhos mais antigos (como em [Dey98]), Dey também definia contexto através de uma enumeração de exemplos de informações contextuais (neste caso, todas as informações referem-se à entidade usuário): estado emocional, foco de atenção, localização e orientação, data, tempo, e objetos e pessoas localizados no ambiente.

A comparação da noção de contexto à localização é comum na computação ubíqua [Sit07]. Desde a publicação da visão de Mark Weiser [Wei91], a computação ubíqua está associada a sistemas móveis e prover serviços adequados de acordo com o ambiente em que o dispositivo está inserido requer o conhecimento de sua localização exata. Mas contexto é mais do que apenas localização [Sit07]. Aspectos como participantes, atividades e tempo executam um importante papel para caracterizar uma situação e não podem ser negligenciados. Então, para cobrir esses e outros aspectos, é necessária uma visão mais ampla de contexto.

Chen e Kotz [Che00], referindo-se a contexto de forma mais geral, descrevem-no como "o conjunto de estados e configurações do ambiente que determinam o

[6] O objetivo geral deste trabalho não é definir contexto de maneira ampla, cobrindo várias áreas de conhecimento. O foco aqui é a aplicação e o uso de contexto em subáreas da Ciência da Computação.

comportamento de uma aplicação ou em que um evento significante para o usuário ocorre". Há ainda autores que relacionam contexto diretamente à interação entre diferentes entidades. Por contexto, Mani e Sundaram [Man07] entendem "o conjunto finito e dinâmico de condições multi-sensitivas e inter-relacionadas que influenciam a troca de mensagens entre duas entidades durante uma comunicação". Winograd [Win02], por sua vez, define contexto como "o conjunto de conjuntos de informações que são relevantes para a comunicação corrente".

Por último, tem-se a definição de Dey e Abowd [Dey99][7] onde contexto é descrito da seguinte forma:

> "Contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade, sendo que uma entidade pode ser uma pessoa, um lugar ou um objeto considerado relevante para a interação entre um usuário e a aplicação, incluindo os próprios usuário e aplicação" [Dey99].

Essa é uma das definições de contexto mais citadas pela comunidade científica que trabalha com computação sensível ao contexto. Nessa definição não é estabelecido um conjunto de informações contextuais relevantes (que poderia ser usado em qualquer aplicação), pois, segundo os autores, o conjunto de informações relevantes para a interação entre o usuário e a aplicação é dinâmico, variando de aplicação para aplicação.

Embora bastante citada na literatura, a definição de Dey e Abowd não é um consenso. Autores como Winograd [Win02] e Chaari *et al.* [Cha06] acreditam que o uso do termo "qualquer informação" se torna tão amplo que contexto passa a ser qualquer coisa. Nesse sentido, Winograd [Win02] afirma que alguma informação é contexto devido à maneira como é usada e não por suas propriedades inerentes. Assim, a voltagem de uma sala é considerada contexto se existe alguma ação do usuário ou do computador cuja interpretação dependa dessa informação, caso contrário, a voltagem é considerada apenas parte do ambiente.

Com base na definição proposta por Dey e Abowd e considerando-se as críticas de Chaari e co-autores e Winograd, a seguir é apresentada a definição de contexto utilizada neste trabalho.

Definição 1: *Contexto* é qualquer informação relevante à execução de uma entidade que pode ser utilizada para descrever o seu estado interno ou o ambiente em que a entidade está inserida.

---

[7] Esta definição é encontrada em vários trabalhos de Dey e colaboradores, mas acredita-se que ela tenha sido introduzida inicialmente na referência citada.

2.1.2. Adaptação

Não há uma única definição para o termo adaptação. O estudo de processos de adaptação e de entidades adaptativas, sejam elas computacionais ou não, é feito por várias áreas de conhecimento. No dicionário on-line Merriam-Webster[8], a palavra adaptação é definida como:

> "1: o ato ou processo de adaptar: a situação de ser adaptado. 2: ajuste as condições ambientais: como (a) ajuste de um órgão sensorial de acordo com a intensidade ou a qualidade de um estímulo; (b) modificação de um organismo ou de suas partes tornando-o mais adequado para sobreviver sob as condições de seu ambiente. (...)".

Já no dicionário Aurélio da Língua Portuguesa [Fer99], em uma das definições da palavra (que dá o seu significado de acordo com a categoria "Biologia"), tem-se: "(...) Ajustamento de um organismo, particularmente do homem, às condições do meio ambiente (...)". O mesmo dicionário define a palavra adaptar como harmonizar-se, adequar-se, ambientar-se (entre outras definições).

Ao longo da história da Ciência da Computação, o termo adaptação tem sido utilizado em uma variedade de diferentes contextos. Enquanto não há unanimidade na definição, é comum a utilização de definições mais gerais, como a exposta em [Kle08]. Klein e co-autores indicam que adaptabilidade, de forma geral, é a habilidade de modificar comportamento, estrutura ou realização observados de um sistema [Kle08]. Como se pode perceber, definições de adaptação em Ciência da Computação não divergem muito das apresentadas por outras áreas, como a Biologia.

A seguir, alguns aspectos de aplicações adaptativas conscientes de contexto (ou aplicações adaptativas ao contexto) são descritos. Essas aplicações utilizam informações contextuais para mudar seus comportamentos em resposta às mudanças ocorridas nos ambientes em que estão inseridas, assunto que é o foco deste estudo.

2.1.3. Sistemas adaptativos ao contexto

Antes de se introduzir o conceito de sistema adaptativo ao contexto é preciso entender o significado de adaptação ao contexto. Para tanto, utilizou-se a definição dada por Klein e co-autores em [Kle08].

---

[8] Web site: www.m-w.com.

Definição 2: Adaptação ao contexto (*context adaptation*) é a habilidade do sistema de capturar informações do domínio ao qual ele pertence, avaliar essas informações e modificar o seu comportamento de acordo com a situação corrente percebida [Kle08].

Adaptação ao contexto apresenta múltiplos aspectos. Serviços de informação podem utilizar o contexto durante o processamento da informação, de forma a apresentar informações adaptadas ao usuário. Como exemplo, restaurantes próximos ao usuário poderiam ser determinados com base em sua localização atual. A forma como a informação é compartilhada também pode ser adaptada de acordo com o contexto de comunicação. Nesse caso, aplicações poderiam enviar mensagens compactadas ao perceber que a banda de rede é limitada. A adaptação ao contexto também deve considerar aspectos internos ao sistema, como os recursos de energia disponíveis. Parar o envio de mensagens quando os recursos de energia internos estão se esgotando (o que preserva alguma energia para o recebimento de mensagens), parece ser uma atitude coerente [Men08]. A seguir, é apresentada a definição de sistemas adaptativos ao contexto dada por Sitou e Spanfelner em [Sit07].

Definição 3: Sistemas adaptativos ao contexto são sistemas baseados em computador capazes de reconhecer mudanças no domínio em que estão inseridos (domínios com os quais compartilham interfaces) e, ao mesmo tempo, capazes de modificar seus comportamentos para se adaptar às novas condições sem a necessidade de interação direta com o usuário [Sit07].

O “domínio” indicado na definição de Sitou e Spanfelner é caracterizado por informações percebidas pelo sistema que são relevantes para a sua adaptação. Essa caracterização é, na verdade, um modelo do ambiente do sistema, o que é comumente chamado de *contexto*.

O uso de informações contextuais para a adaptação de aplicações é visto em diferentes áreas da Ciência da Computação, como *autonomic computing*, computação pervasiva e sistemas multiagentes. De acordo com Klein e co-autores [Kle08], a adaptação ao contexto pode ser vista como a tecnologia que possibilitará, no futuro, o desenvolvimento de aplicações no campo de *autonomic computing*. As habilidades individuais de sistemas autonômicos (isto é, as habilidades intrínsecas nos quatro *self*-*) requerem que eles sejam capazes de deduzir e ativar medidas adequadas com base na situação corrente e nas políticas de ajuste definidas.

De acordo com Menon [Men06], a adaptação em ambientes pervasivos é um fenômeno complexo, uma vez que envolve múltiplas entidades, como usuários, dispositivos e serviços, que podem ser autônomas e estar dispersas espacialmente no ambiente. De acordo com Hu *et al.* [Hu08], aplicações conscientes de contexto utilizam informações contextuais para adaptar os seus comportamentos e se ajustar às mudanças ocorridas no ambiente computacional ou no ambiente em que o usuário está inserido.

Alguns autores indicam que as aplicações conscientes de contexto são, por natureza, adaptativas ao contexto (como em [Hu08]). Mas há autores (como [Che00]) que indicam dois tipos de sensibilidade ao contexto: sensibilidade ativa, onde é feita adaptação; e sensibilidade passiva, onde o contexto é apenas recuperado e armazenado. Efstratiou [Efs04], de forma ainda mais clara, indica que as aplicações conscientes de contexto adaptativas são um subconjunto das aplicações conscientes de contexto. Assim, para evitar uma interpretação incorreta, consideram-se neste estudo as aplicações conscientes de contexto adaptativas ou aplicações adaptativas ao contexto.

Agentes de software [Wei99], de forma similar às aplicações adaptativas ao contexto, devem ser capazes de responder às mudanças ocorridas em seus ambientes[9], melhorando seus desempenhos ou tornando seus comportamentos mais apropriados. A importância da adaptação em uma sociedade de agentes vem sendo citada há alguns anos na literatura. Haynes e Sem, em [Hay96], já indicavam que a adaptação é um componente-chave em qualquer sociedade de agentes. Em [Ima04] é encontrada a seguinte definição para agentes adaptativos inteligentes:

> "Agentes adaptativos inteligentes são sistemas ou máquinas que utilizam metodologias computacionais complexas ou baseadas em inferência para modificar ou transformar parâmetros de controle, bases de conhecimento, metodologias para a resolução de problemas, o curso de ação, ou qualquer outro objeto de forma a efetuar satisfatoriamente um conjunto de tarefas que são de interesse do usuário" [Ima04].

Imam [Ima04] ainda indica que a adaptação de agentes inteligentes pode ser agrupada em três categorias com base no relacionamento entre a adaptação interna e o comportamento externo (observável dos agentes). As categorias são:

- Adaptação interna: quando os sistemas internos usados pelos agentes são adaptativos, mas suas ações externas não refletem qualquer comportamento adaptativo.

[9] Em sistemas multiagentes, o contexto em que os agentes estão inseridos é normalmente referenciado como ambiente.

- Adaptação externa: quando os sistemas internos não são adaptativos, mas as ações externas refletem comportamento adaptativo.
- Adaptação completa: quando os sistemas internos são adaptativos e as conseqüências das adaptações são refletidas nas ações externas do agente.

Splunter e co-autores [Spl03] indicam que algumas vezes o comportamento reativo dos agentes, ou seja, a capacidade de abandonar um objetivo ou plano e adotar outro objetivo ou plano mais adequado à situação, é apelidado de “comportamento adaptativo”. Mas para os autores, adaptar um agente significa fazer mudanças em sua estrutura, o que inclui os conhecimentos e fatos disponíveis no agente.

Os autores deste trabalho acreditam que, independentemente do tipo e das características da arquitetura, um sistema pode ser considerado adaptativo ao contexto se for capaz de reconhecer mudanças no domínio em que atua e de modificar seu comportamento de forma automática para se adaptar a essas mudanças. Assim, corrobora-se a definição de Sitou e Spanfelner [Sit07] para sistemas adaptativos ao contexto (apresentada na definição 3).

2.2. Estudos relacionados

A organização dos estudos relacionados à adaptação de agentes ao contexto foi feita de acordo com a classificação proposta em [McK04]. McKinley e co-autores indicam que há duas maneiras comuns de implementar adaptação de software, que são a adaptação composicional e a parametrizada [McK04]. Na abordagem composicional, os sistemas são capazes de se reconfigurar dinamicamente, em tempo de execução, para se adequar ao ambiente corrente. Seguindo essa abordagem, é possível modificar algoritmos ou componentes estruturais do sistema. São exemplos de adaptação composicional as abordagens propostas em [Amo09, Gun08a, Han07, Ama05]. De acordo com Gunasekera e co-autores [Gun09b], enquanto o aprendizado de agentes tem sido foco de um número considerável de pesquisas, a adaptação composicional de agentes tem recebido pouca atenção da academia.

A adaptação parametrizada permite a modificação de variáveis do programa que determinam o seu comportamento. Assim, utilizando a adaptação parametrizada é possível modificar parâmetros de forma a adotar diferentes estratégias pré-definidas, mas não é possível adotar novas estratégias (não conhecidas *a priori*) [McK04]. Exemplos de

abordagens baseadas em adaptação parametrizada podem ser encontrados em [Sim08, Wey08, Ler03, Ram03].

2.2.1. Estudos baseados em adaptação parametrizada

Lerman e Galstyan [Ler03] propõem um mecanismo para adaptação em SMAs onde os agentes são capazes de modificar seus comportamentos com base em memórias de eventos passados. De fato, os agentes utilizam a memória para estimar o estado global do sistema e utilizam essa estimativa para ajustar seus comportamentos através da modificação de parâmetros que governam suas ações. O trabalho descrito apresenta, portanto, um tipo de adaptação parametrizada.

Para ilustrar o mecanismo proposto, os autores utilizam um mercado eletrônico hipotético composto de agentes de compra com capacidade de se mover. Nesse mercado, agentes devem fazer coalizões para comprar mercadorias com preços reduzidos (na compra de grandes quantidades, o preço da unidade diminui). O comportamento adaptativo dos agentes pode ser visto na decisão pela adesão ou abandono de uma coalizão (decisão tomada com base no tamanho das coalizões já conhecidas pelo agente).

Simpkins e co-autores, em [Sim08], apresentam uma linguagem de programação para o desenvolvimento de agentes de software adaptáveis. A linguagem foi definida com base nos conceitos de programação parcial (*partial programming*) e aprendizado por reforço (*reinforcement learning*). Agentes desenvolvidos com a linguagem proposta são responsáveis por perceber as mudanças no ambiente e manter o modelo de contexto. Aparentemente, as informações contextuais utilizadas são relacionadas exclusivamente a aspectos internos da aplicação (eventos ocorridos durante a execução do agente). Os possíveis estados do contexto são representados em uma máquina de estados, com regras de transição associadas a cada estado possível.

No referido trabalho, o *A^2BL runtime system* é o responsável pela verificação e combinação de estados, ações e prêmios, determinando o comportamento de um agente. As variações no comportamento do agente são resultado da ativação de diferentes ações (de forma a aumentar o número de "prêmios" recebidos). Não há um conjunto de regras de adaptação pré-fixado - no construtor *state* são indicadas as informações contextuais que devem ser observadas para o aprendizado de uma política efetiva, que é então construída com a utilização de um algoritmo para aprendizado supervisionado.

Weyns e co-autores [Wey08] comparam duas abordagens para a atribuição dinâmica de tarefas utilizando SMAs situados. Para a comparação das abordagens foi desenvolvida uma aplicação no domínio da manufatura (sistema de transporte industrial composto por veículos guiados automaticamente). Os autores não indicam explicitamente as fontes de informações contextuais utilizadas pela aplicação, mas oferecem uma representação virtual do ambiente (uma entidade de software que representa e mantém o estado relevante do ambiente físico e permite que os agentes se comuniquem) de onde os agentes podem recuperar as informações contextuais. A recuperação de informações é feita sob demanda e é guiada pelo mecanismo decisório do agente. O mecanismo de adaptação utilizado pelos agentes também não é detalhado no artigo. Sabe-se, no entanto, que o escopo de adaptação compreende a seleção e execução de ações, como as direções a seguir (o que indica uma abordagem parametrizada). As regras de adaptação são descritas em um formato proprietário.

Em [Ran03], é proposto um *middleware* para facilitar o desenvolvimento de agentes conscientes de contexto (para os autores, consciência de contexto inclui a habilidade de adaptação a diferentes situações). O *middleware* é integrado ao Gaia[10] e, portanto, utiliza CORBA para a comunicação entre os agentes distribuídos.

De acordo com Ranganathan e Campbell [Ran03], uma questão-chave em ambientes pervasivos/ubíquos é permitir que agentes autônomos e heterogêneos tenham um entendimento compartilhado das informações contextuais. Nesse sentido, são utilizadas ontologias para definir a estrutura e as propriedades dos diferentes tipos de informações contextuais. As informações contextuais propriamente ditas são representadas como predicados - a ontologia especifica apenas as estruturas dos diferentes predicados de contexto, sendo utilizada para verificar suas validades.

Os serviços para consciência de contexto oferecidos pelo *middleware* são de responsabilidade de diferentes agentes. Os agentes *provedores de contexto* encapsulam sensores ou outras fontes de informações contextuais. Os agentes *sintetizadores de contexto* processam os dados coletados pelos provedores de contexto de forma a deduzir informações contextuais em um nível de abstração mais alto. Há também outros três serviços providos por agentes, que são: busca por provedores de contexto, acesso a contextos históricos e acesso ao servidor que mantém as ontologias. Por último, há os

[10] http://gaia.cs.uiuc.edu/

consumidores de contexto, que são os agentes que usufruem dos serviços prestados pelos agentes do *middleware* e utilizam as informações contextuais para se adaptar.

Os agentes integrantes do *middleware* podem raciocinar sobre as informações contextuais utilizando regras escritas com diferentes tipos de lógica. Porém, de acordo com os autores, abordagens baseadas em regras não são flexíveis e não conseguem se auto-adaptar às mudanças no ambiente. Assim, também é possível utilizar técnicas de aprendizado de máquina para lidar com o contexto. De qualquer forma, o escopo de adaptação compreende apenas a ativação de ações (métodos) do agente, o que caracteriza uma abordagem parametrizada.

Ainda na solução proposta por Ranganathan e Campbell [Ran03] destacam-se os seguintes aspectos: (i) a utilização de um serviço para a localização de provedores de contexto; (ii) a utilização de ontologias para a criação de um vocabulário comum; (iii) a definição de prioridade nas regras de adaptação, o que ajuda a gerenciar os casos de conflito; e (iv) a utilização de lógica temporal nas regras, permitindo a ativação de ações por determinados períodos.

2.2.2. Estudos baseados em adaptação composicional

VERSAG (acrônimo para *VERsatile Self-Adaptive Agents*) é uma arquitetura de agentes leves onde um agente é visto como um “portador” de componentes de software (os comportamentos de um agente são implementados como componentes de software portáveis que podem ser compartilhados e implantados em tempo de execução) [Gun08a, Gun08b, Gun09a, Gun09b]. Os componentes compartilhados pelos agentes são também chamados de *capabilities* pelos autores. Exemplos de comportamentos que podem ser adicionados por um agente na forma de *capability* incluem a habilidade de agir como um comprador ou um vendedor em um leilão, a habilidade de minerar dados de uma base de dados, entre outros. Na arquitetura VERSAG, uma *capability* pode ser definida pela tupla *<id, F, credentials, Env>*, sendo [Gun08b]:

- *id*: identificador único.
- *F*: conjunto de funções (normalmente na forma de código compilado).
- *credentials*: metadados, como origem e versão.
- *Env*: conjunto de ambientes em que a *capability* pode ser utilizada.

De acordo com os autores, são características da arquitetura VERSAG: (i) a capacidade de compartilhamento de *capabilities* entre pares de agentes, o que significa

que todos os agentes são potenciais provedores de *capabilities*; (ii) a habilidade dos agentes de se adaptar com base em informações contextuais; e (iii) a mobilidade ao nível de componentes com mecanismos eficientes de migração [GUN08a]. A última característica citada (mobilidade) é bastante marcante na solução proposta. Agentes VERSAG são guiados por um itinerário, o qual especifica os locais que o agente deve passar e as atividades que devem ser realizadas em cada localização. Para conseguir realizar essas atividades, os agentes devem possuir diferentes habilidades, que estão disponíveis no sistema na forma de *capabilities*. Já a suposta habilidade de adaptação ao contexto apresentada pelos agentes (característica 2), é apenas citada em [Gun08a] e [Gun09a].

Uma implementação da arquitetura VERSAG está sendo feita com base na plataforma Jade (agentes VERSAG constituem uma fina camada sobre agentes Jade). No entanto, os autores indicam que um agente VERSAG poderia ser portado para diferentes plataformas.

Vários pontos ficam sem resposta (ou com resposta vaga) na literatura disponível sobre a arquitetura VERSAG (a saber [Gun08a, Gun08b, Gun09a, Gun09b]). Por exemplo, os autores indicam que os agentes devem localizar as *capabilities* necessárias para que as tarefas sejam cumpridas, mas não indicam como é o casamento entre as descrições de tarefas e as *capabilities* disponíveis. Na verdade, o exemplo apresentado em [Gun08b] indica que a descrição das tarefas (no itinerário do agente) já contém a identificação das *capabilities* que devem ser localizadas (então não é necessário fazer casamento algum). Os autores também criticam a existência de um centralizador para o compartilhamento de comportamentos entre os agentes, mas utilizam o *Jade Directory Facilitor* como forma de localizar agentes com comportamentos específicos.

Para finalizar, as características da arquitetura VERSAG são bastante similares as características do *framework* Ontowledge, apresentado pelos autores desta pesquisa em trabalhos anteriores [Lem07a, Lem09]. O *framework* Ontowledge, assim como a VERSAG, permite o compartilhamento de “componentes” entre pares de agente, sendo que esses “componentes” são encapsulados em entidades chamadas “objetos de conhecimento”. Entretanto, diferentemente da arquitetura VERSAG, o Ontowledge não foi concebido com o intuito de permitir a adaptação de agentes ao contexto, mas sim para possibilitar aos agentes compartilhar conhecimento de uma maneira sistemática através do uso de um processo de Gestão de conhecimento.

Han e co-autores [Han07] propõem uma abordagem para a adaptação dinâmica de agentes com o uso de reconfiguração em tempo de execução. Além da definição do processo de adaptação, os autores desenvolveram uma plataforma para a criação de agentes que utiliza uma representação de dependência de componentes baseada em matrizes. Com base na matriz de dependência, o plano de reconfiguração de componentes pode ser avaliado antes da realização da reconfiguração propriamente dita.

A Figura 2.1 ilustra o processo de adaptação proposto em [Han07]. O *componente de monitoramento* é responsável por observar o estado do agente. O *repositório de políticas* gerencia a estratégia de adaptação utilizando uma matriz de dependência. O *gerenciador de reconfiguração* tem duas responsabilidades: (1) decidir, com base na informação extraída pelo componente de monitoramento, a estratégia de adaptação apropriada; e (2) inicializar o processo de reconfiguração através da transformação da matriz de dependências. O *repositório de dependências* armazena a matriz utilizada em agentes remotos (informação que é manipulada juntamente com o repositório de agentes para prover informações detalhadas de implementação). Por último, depois de a política ser determinada, é feita uma verificação a nível arquitetural para conferir a validade da reconfiguração proposta (responsabilidade do *componente de verificação a nível arquitetural*).

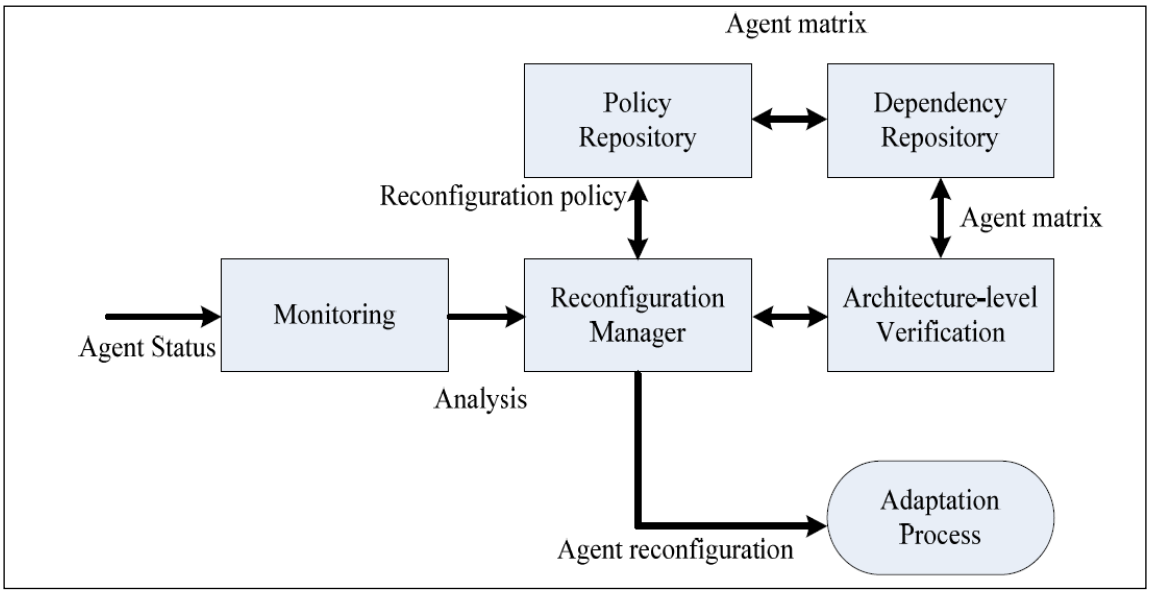

Figura 2.1 – Processo de adaptação de agente controlado pelo Gerenciador de Reconfiguração (proposto em [Han07]).

Uma questão interessante levantada por Han e co-autores [Han07] diz respeito ao comportamento apresentado pelo agente após sua adaptação. De acordo com os autores, “é importante notar os potenciais efeitos que a troca dinâmica de componentes pode causar nos serviços dos agentes” [Han07]. Assim, no trabalho proposto por eles, antes

que qualquer adaptação seja colocada em prática, são avaliadas as dependências entre os componentes. Além disso, os resultados obtidos com o processo de adaptação são avaliados pelo usuário. Uma das principais críticas a abordagem proposta por Han e co-autores é a forma de descrever as adaptações possíveis. Aparentemente, as adaptações são codificadas, em tempo de projeto, no método de configuração do agente.

Em [Amo09], Amor e Fuentes descrevem uma arquitetura de agentes baseada em aspectos e componentes chamada de Malaca. A arquitetura foi desenvolvida para atender a algumas questões, que são: (i) reusabilidade, no sentido de utilizar componentes de software comerciais “de prateleira” (*commercial-off-the-shelf components*); (ii) separação de interesses, principalmente para separar as funcionalidades dos aspectos de interação, aumentando a coesão dos elementos arquiteturais e diminuindo seu acoplamento; (iii) interoperabilidade, para que os agentes desenvolvidos com a arquitetura possam ser executados sobre qualquer plataforma para o desenvolvimento de agentes compatível com a FIPA (na verdade, são definidos *adaptors* para garantir a interoperabilidade); (iv) extensibilidade, para permitir a adição de novos interesses de comunicação, como criptografia de mensagens, mobilidade e outros; e (v) facilidade de programação, evitando a apresentação de código de baixo nível para a combinação de componentes e aspectos.

O modelo de agente da arquitetura Malaca se concentra em prover uma melhor modularização e separação dos interesses relacionados com o aspecto comunicação (no modelo proposto, um aspecto é uma entidade utilizada para modularizar qualquer propriedade de agente identificada no contexto da comunicação entre agentes). Isto significa que a arquitetura não tem compromisso com nenhum modelo de agente em particular, mas é genérica e extensível para apoiar a criação de agentes com diferentes propriedades.

O comportamento apresentado por cada papel de agente (agentes executam papéis na arquitetura proposta) é descrito por uma máquina de estados finita, e são executadas ações a cada transição de estado. É o aspecto de coordenação o responsável por invocar as ações dos agentes.

A configuração de agentes Malaca é feita a partir de um arquivo XML escrito na linguagem MaDL (linguagem de descrição de arquiteturas orientadas a aspectos e componentes), também definida por Amor e Fuentes em trabalhos anteriores (vide [Amo06]). Para facilitar o desenvolvimento de agentes na arquitetura proposta, foi desenvolvida a ferramenta *Malaca Agent Description* (MAD), que permite a descrição de

agentes sem a manipulação direta do arquivo XML. Na ferramenta citada também é possível especificar as regras para composição de aspectos e componentes.

Aspectos são executados e compostos apenas na recepção e no envio de mensagens pelo agente. Já os componentes podem ser modificados no contexto de uma conversação (parte da execução de um protocolo de interação controlado pelo aspecto de coordenação). Para cada ponto de interceptação é preciso definir uma regra que indica como o aspecto será composto. Desta forma, é definida uma regra para compor aspectos quando uma mensagem é enviada e outra regra para quando mensagens são recebidas. Uma entidade chamada *Mediator* é responsável pelo processo de combinação de aspectos e componentes. Na verdade, a composição propriamente dita é feita pelo *Agent Configuration Service*, que faz parte da arquitetura da entidade *Mediator*. De acordo com os autores, a modificação da arquitetura interna dos agentes é feita em tempo de execução sem a necessidade de parar ou recompilar o agente.

As regras de composição também podem ser modificadas em tempo de execução. Como as regras são consultadas sempre que é recebida ou enviada uma mensagem, o conjunto de regras pode ser alterado em tempo de execução e, a partir da interceptação de uma nova mensagem, já será utilizado o novo conjunto de regras. A arquitetura proposta está disponibilizada na forma de um *plugin* para o Eclipse.

O trabalho de Amor e Fuentes não descreve, exatamente, uma arquitetura para a adaptação de agentes ao contexto (embora se pudesse considerar as mensagens recebidas pelos agentes e o seu estado interno - conjunto de componentes e aspectos ativos - como o contexto dos agentes). Além disso, os interesses transversais identificados pelos autores compreendem apenas aspectos arquiteturais dos agentes, e não os seus comportamentos e estrutura como o proposto neste trabalho. Cabe salientar que a definição dos aspectos utilizados pelos autores é similar a proposta em [Gar08, Lob04]. Para Lobato e co-autores, sistemas multiagentes possuem um rico conjunto de interesses (*concerns*) que incluem “propriedades de agência”, tais como autonomia, adaptação e interação [Lob04]. Por último, em [Amo09] há uma grande simplificação no conjunto de locais na execução do agente onde podem ser executados os aspectos (ou, de acordo com a terminologia da programação orientada a aspectos, o conjunto de *join points*), visto que só são interceptados o envio e o recebimento de mensagens.

2.3. Considerações sobre o capítulo

As seções anteriores apresentaram um apanhado geral na literatura buscando contextualizar o uso da noção de contexto em sistemas computacionais adaptativos. Inicialmente, algumas definições para contexto foram apresentadas, esclarecendo as diversas interpretações dadas para o termo. Depois disso, discorreu-se sobre conceitos de adaptação e de sistemas adaptativos ao contexto. Sempre que considerado necessário, foram enfatizadas as definições/classificações aceitas pelos autores deste trabalho.

Em relação à adaptação, apenas uma breve contextualização foi apresentada. Como já foi dito, adaptação é um tema estudado há muitos anos em várias áreas de conhecimento e, sendo assim, possui uma vasta gama de trabalhos relacionados. Em [Eli06] e [Eli08] podem ser encontradas mais informações sobre a origem do termo adaptação e toda a epistemologia associada a ele.

Depois de apresentados os principais conceitos referentes à pesquisa, foram descritos alguns trabalhos relacionados. A Tabela 2.1 sumariza as características dos trabalhos relacionados citados nesse capítulo (as características cujos tratamentos não foram indicados explicitamente nos artigos foram marcadas com o símbolo “-”). Como se pode observar na tabela, diferentes tipos de soluções são propostas na literatura e a maioria dos trabalhos foca em aprendizado individual. Também, ainda são poucos os trabalhos que aplicam esforços para a verificação das adaptações realizadas (no sentido de avaliar o nível de satisfação alcançado). A definição de um metamodelo para classificar as informações contextuais também é uma questão que merece maior atenção da comunidade científica, visto que apenas em [Ran03] é detalhado o modelo de contexto utilizado.

Como pode ser visto na coluna “Escopo de adaptação”, o comportamento adaptativo de grande parte dos estudos analisados é realizado no nível de seleção (e conseqüentemente ativação) de entidades definidas em tempo de projeto (sejam elas funções, componentes, ou ações). De acordo com González e co-autores [Gon07], a utilização das tecnologias de programação disponíveis faz com que a adaptabilidade em tempo de execução seja, muitas vezes, um aspecto de projeto derivado da arquitetura do software. Isto porque, em todas as técnicas de programação existentes, os pontos de variação são fixados durante o projeto e pouco progresso tem sido obtido com respeito à influência mútua entre consciência do contexto e variabilidade dinâmica de sistemas. Por último, apenas Amor e Fuentes [Amo09] disponibilizam uma ferramenta para uso.

A última linha da Tabela 2.1 já antecipa as características apresentadas pela arquitetura proposta neste trabalho. Como pode ser visto, desenvolveu-se uma *arquitetura* para a criação de agentes de software adaptativos ao contexto, sendo que o aprendizado das melhores políticas de adaptação pode ser *individual* ou *multiagentes* (mediante compartilhamento de adaptadores). Outra característica da arquitetura é a *verificação da satisfação* atingida com a adaptação realizada. Também, a arquitetura poderá ser utilizada no desenvolvimento de sistemas multiagentes de diferentes domínios e de agentes conscientes de *diferentes tipos de informações contextuais*. O escopo de adaptação compreende, principalmente, os *planos de ação* e as *ações* dos agentes. No entanto, podem ser modificadas *outras estruturas* presentes na arquitetura interna dos agentes. A arquitetura desenvolvida ficará disponível para *download* mediante obtenção de licença.

Com base na análise das descrições dos trabalhos relacionados também foi possível identificar certos requisitos que deveriam ser contemplados por uma arquitetura para o desenvolvimento de aplicações adaptativas ao contexto. Alguns requisitos estavam explicitamente citados nos artigos, enquanto outros foram deduzidos a partir de citações dos autores. Os requisitos identificados estão listados a seguir.

Para o gerenciamento do contexto, a arquitetura deve:

1. Apoiar a coleta de informações contextuais de diferentes sensores e a entrega dessas informações para os agentes [Ran03].
2. Permitir a implantação e o uso de novos sensores durante a execução do sistema (baseado em [Ran03]).
3. Utilizar um metamodelo de contexto que permita, além da instanciação das informações contextuais de domínio, a visualização dos tipos de informações contextuais, suas propriedades (como precisão e dinamicidade) e origens (baseado em [Ran03, Abo00]).
4. Utilizar uma forma de representação para informações contextuais que seja robusta e de rápida atualização (baseado em [Gon07]).
5. Dar suporte ao processamento das informações contextuais de forma a obter informações em um nível de abstração maior do que o provido pelos sensores [Ran03].
6. Permitir interoperabilidade sintática e semântica entre diferentes agentes (através do uso de ontologias) (afirmado por [Ran03] e corroborado em [Gon07]).

Tabela 2.1 – Sumarização das características dos trabalhos relacionados.

	Fonte	Tipo de solução	Tipo de aprendizado	Avaliação da adaptação	Solução genérica	Contexto	Escopo da adaptação	Ferramenta disponível
Abordagem parametrizada	[Ler03]	Mecanismo / algoritmo	Individual	-	-	Memórias passadas	Execução de ações	Não
	[Sim08]	Linguagem de programação	Individual	Sim	Sim	Eventos ocorridos durante a execução	Execução de ações (de forma a aumentar o número de "prêmios" recebidos)	Não
	[Wey08]	Comparação de duas abordagens para a atribuição dinâmica de tarefas	Multiagentes	-	Não	Informações do ambiente físico	Execução de ações, como as direções a seguir.	Não
	[Ran03]	Middleware	Individual	Sim (humana)	Sim	Diverso (contexto físico, do sistema, do dispositivo, entre outros)	Execução de ações (métodos)	-
Abordagem composicional	[Gun08a]	Arquitetura	Individual e multiagentes (há interação entre os agentes)	-	Sim	-	Adição e remoção de capabilities de acordo com os comandos do itinerário / Migração de agentes	Não
	[Han07]	Abordagem (processo de adaptação + plataforma em desenvolvimento)	-	Sim (humana)	Sim	-	Adição e remoção de componentes	Não
	[Amo09]	Arquitetura	Individual	-	Sim	Componentes e aspectos ativos no agente	Mecanismos de codificação de mensagens (aspecto de comunicação dos agentes)	Sim
	Solução proposta	Arquitetura	Individual e multiagentes	Sim	Sim	Diverso (contexto físico, do sistema, do dispositivo, do usuário, entre outros)	Plano de ação, ações e outras estruturas necessárias	Sim

7. Definir meios para o armazenamento e tratamento de informações contextuais históricas (baseado em [Cha06, Mik06]).

Já para a adaptação ao contexto, é necessário:

8. Possibilitar a realização de adaptações em diferentes estruturas da arquitetura interna dos agentes (segundo Gunasekera e co-autores [Gun08b], agentes de software adaptativos são ainda limitados por causa das possibilidades de adaptação disponíveis).
9. Verificar dependências entre o componente de software que está sendo modificado e os demais componentes do agente antes da realização de qualquer adaptação (baseado em [Han07]).
10. Permitir aos agentes utilizar diferentes tipos de mecanismos para raciocínio ou aprendizado [Ran03].
11. Disponibilizar um repositório de adaptações (variações inseridas na arquitetura) que possa ser expandido pelo agente por meio de aprendizado ou por colaboração com outros agentes (baseado em [Gun08b]).
12. Permitir que o agente avalie as adaptações realizadas, de forma a atender as expectativas dos usuários e melhorar seu desempenho (baseado em [Sit07]).
13. Realizar as modificações no agente sem que os atrasos adicionados pela adaptação possam ser percebidos pelo usuário (baseado em [Aye08]).

Para o desenvolvimento da arquitetura **K2**, foram observados os requisitos acima listados, conforme discutido no próximo capítulo (que apresenta o modelo conceitual da arquitetura).

3. MODELO DE ARQUITETURA PARA A ADAPTAÇÃO DE AGENTES AO CONTEXTO

É sabido que “sistemas de software modernos devem ser flexíveis e adaptáveis para lidar com ambientes dinâmicos” [Han07]. Então, considerando-se a importância da utilização de informações contextuais para a criação de aplicações que consigam perceber o ambiente em que estão inseridas e agir de acordo com essas percepções, neste trabalho está sendo proposta uma arquitetura para o desenvolvimento de agentes de software adaptativos ao contexto. De fato, o objetivo geral da arquitetura é permitir a modificação de partes de elementos estruturais de um agente de software, adaptando o seu comportamento e estrutura de acordo com as mudanças percebidas no contexto em que o agente está inserido. A adaptabilidade permitida pela arquitetura possibilita, entre outras coisas, a modificação dos planos de ações e das próprias ações dos agentes.

O tipo de adaptação utilizada é a composicional [McK04]. De acordo com Gunasekera e co-autores [Gun09a], um agente de software se adapta de forma composicional quando sua estrutura interna é alterada com a aquisição de novos componentes e funcionalidades. Ainda segundo os autores, a utilização de adaptação composicional é justificada devido à diversidade de tarefas que precisam ser executadas e o número de ambientes que devem ser atendidos pelos agentes, o que torna a adaptação via aprendizado muitas vezes insuficiente.

De forma geral, a realização de adaptação consciente de contexto requer o estudo de várias questões, como: captura, modelagem e interpretação do contexto, disseminação das informações contextuais e adaptação da aplicação, o que compreende a definição do que poderá ser adaptado e dos tipos de adaptações que poderão ser realizadas. Para a discussão de todas essas questões, a arquitetura proposta será apresentada em dois diferentes níveis. No primeiro nível (apresentado neste capítulo), serão enfatizados o modelo para representação das informações contextuais, os locais na execução e na estrutura de um agente onde poderão ser feitas as adaptações e os tipos de adaptações passíveis de realização sempre que o contexto atingir determinado estado ou ocorrer determinado evento na arquitetura interna de um agente. Também, nesse nível é descrito o modelo conceitual da arquitetura.

No segundo nível (apresentado no Capítulo 4), é fornecida uma descrição completa, em termos de implementação, da arquitetura proposta.

3.1. Representação das informações contextuais

Sistemas computacionais são basicamente representacionais [Dou04], o que nos faz procurar formas para codificar e representar o contexto. Até o momento, não há um consenso na literatura sobre qual o melhor modelo ou linguagem para a representação de informações contextuais. Acredita-se, no entanto, que a criação de representações mais sofisticadas tende a aumentar a capacidade das aplicações conscientes do contexto, além de fornecer uma correta separação entre o que é sentido (informações capturadas do ambiente) e quais as atitudes que devem ser tomadas a partir de cada percepção (a adaptação necessária para o contexto sentido).

Assim sendo, foi desenvolvido um modelo de referência para a classificação de informações contextuais. O modelo possibilita, entre outras coisas, visualizar os tipos de informações contextuais, as suas origens (de onde uma informação foi captada) e como diferentes informações colaboram para descrever o contexto em que uma entidade está inserida. Como poderá ser observado durante a descrição do modelo, vários de seus conceitos e relações já foram citados ou sugeridos por outros autores na literatura. Isso ocorre porque um dos objetivos da criação do modelo compreendia, justamente, a identificação e a compilação de características comuns presentes nos diferentes modelos para representação de informações contextuais disponíveis.

Antes de apresentar o modelo desenvolvido, serão apresentadas algumas características e propriedades das informações contextuais, as quais ajudam a entender as relações e conceitos do modelo proposto. Também, discorrer-se-á sobre as formas de representação de contexto, salientando-se os prós e contras de cada alternativa.

3.1.1. Características das informações contextuais

De acordo com Henricksen *et al*. [Hen02], as informações contextuais podem ser caracterizadas em relação ao tempo como estáticas ou dinâmicas. Informações de contexto estáticas descrevem os aspectos de sistemas que não são alterados no decorrer do tempo enquanto as informações de contexto dinâmicas compreendem o conjunto de informações que variam constantemente. Por exemplo, a data do aniversário é considerada uma informação estática de um indivíduo, já a sua localização é uma informação dinâmica, pois varia constantemente ao longo do tempo. Ainda segundo os autores, existem três tipos de informações dinâmicas [Hen02]:

- Sentidas (do inglês *sensed*): são aquelas capturadas por meio de sensores lógicos e físicos. Como exemplo, cita-se o nível de umidade de um ambiente (que pode ser adquirido através de um sensor de umidade).
- Informadas: são aquelas fornecidas explicitamente pelo usuário, como sua senha de acesso ou itens de sua agenda particular.
- Interpretadas: são as informações obtidas através da análise de outras informações contextuais. A presença de dispositivos perto de uma pessoa é um exemplo desse tipo de informação, pois pode ser deduzida a partir da localização atual de cada objeto.

Outras características de informações contextuais são:

- Precisão: informações contextuais geralmente são imperfeitas [Hen02, Has05]. A imperfeição, neste caso, pode se referir a uma interpretação errônea do estado atual do mundo, a existência de informações contraditórias ou ao não conhecimento de algum aspecto referente à própria informação contextual (o que pode prejudicar a sua captura e análise). Essas imperfeições representam um problema especialmente quando se considera a veracidade das informações interpretadas que, além de serem criadas através de métodos e regras muitas vezes imprecisos, podem ser baseadas em dados que não refletem a situação atual da entidade em questão.
- Forte relacionamento entre diferentes informações contextuais: Henricksen *et al*. [Hen02] afirmam que as informações contextuais são altamente interconectadas. Segundo os autores, há relações óbvias entre os indivíduos, seus dispositivos e canais de comunicação.
- Estabilidade das informações contextuais: Dourish, em [Dou04], defende que contexto não é estável (pode variar de aplicação para aplicação e também de atividades ou eventos de uma mesma aplicação). Segundo o autor, o contexto não pode ser estável e nem pode ser apenas uma descrição externa da cena em que uma atividade ocorre. Ao invés disto, ele surge da atividade e é contínuo através dela.
- Dinamicidade das informações: muitos autores sustentam a idéia de que o conjunto de informações contextuais relevantes é dinâmico (não pode ser delimitado e definido *a priori*). De acordo com Mani e Sundaram [Man07], os

atributos específicos de contexto são sempre dependentes de aplicação, direcionados as suas reais necessidades.

3.1.2. Formas de representar o contexto

De acordo com Henricksen *et al.* [Hen02], uma das alternativas conhecidas para a modelagem de informações contextuais é a utilização de técnicas de modelagem de dados vindas da área de Sistemas de Informação, onde as informações são normalmente armazenadas e manipuladas com o apoio de um sistema para gerenciamento de bases de dados (SGBD). Outra alternativa é a utilização de técnicas para a modelagem de sistemas orientados a objeto (OO), como a UML [Pen03].

A UML pode ser utilizada para construir um modelo de informações contextuais e também para auxiliar no mapeamento desse modelo em uma linguagem de programação OO. Rocha e Endler [Roc05] afirmam que utilizar uma abordagem baseada em técnicas de modelagem para sistemas OO é simples, fácil de programar e eficiente, o que justifica o fato de muitos pesquisadores adotarem esse tipo de técnica. Porém, ainda segundo Rocha e Endler, o uso de técnicas baseadas em OO requer que sejam desenvolvidas infra-estruturas para suportar todas as operações sobre as informações contextuais (como armazenamento e consultas).

Já segundo Henricksen *et al.* [Hen02], técnicas de modelagem OO não são eficientes para a modelagem de informações contextuais devido às dificuldades encontradas para: (i) distinguir os diferentes tipos de informações contextuais (para expressar, por exemplo, a diferença entre as informações estáticas e dinâmicas, e entre as informações sentidas e as fornecidas pelo usuário); (ii) representar as características temporais das informações; e (iii) para expressar relações como dependência. Como alternativa para a representação das informações contextuais, os autores sugerem o uso de construtores especiais desenvolvidos com base nas características do contexto que deverá ser modelado.

O uso de ontologias para representar informações contextuais é citado em vários trabalhos, como [Man07, Che03, Gan04, Str04, Roc05, Hef03]. De acordo com Rocha e Endler [Roc05], ontologias constituem um poderoso paradigma para modelagem de contexto devido a sua rica expressividade e por prover suporte aos aspectos de evolução das informações contextuais.

Na análise apresentada por Strang e Linnhoff-Popien [Str04], embora seja preconizado o uso de ontologias, são citadas as seguintes abordagens para modelagem de contexto encontradas na literatura: modelagem *key-value*, *markup scheme models*, modelagem orientada a objetos, modelagem gráfica, modelagem baseada em lógica e, por último, modelagem baseada em ontologias. A Tabela 3.1 apresenta uma síntese das características de cada uma das abordagens citadas.

O modelo de referência para classificação de informações contextuais definido neste trabalho (que é apresentado na próxima subseção) é representado com o uso de ontologias e apresenta, portanto, os aspectos positivos e negativos de tal forma de representação (conforme última linha da Tabela 3.1).

3.1.3. O modelo de referência para classificação de informações contextuais proposto

Considerando-se a importância de um modelo para melhor estruturar as informações contextuais, foi desenvolvido um novo modelo de referência para a representação e classificação dessas informações. O modelo desenvolvido é, na verdade, uma sumarização de outros modelos para representação de informações contextuais disponíveis na literatura. Como será visto a seguir, o modelo incorporou os tipos de informações contextuais propostos por Henricksen e co-autores em [Hen02], agregando a eles uma série de atributos e relações (que não tinham sido definidos pelos autores no artigo citado).

Como forma de representação, optou-se pelo uso de ontologias. De maneira simplificada, o modelo de referência pode ser considerado como uma ontologia geral que pode ser estendida por ontologias de domínio (ou ontologias mais específicas). A utilização de ontologias para representar informações contextuais, embora ainda não seja um consenso na literatura, tende a facilitar o compartilhamento de conhecimento e a interpretação por agentes de software, visto que as ontologias estabelecem uma terminologia comum para o entendimento dos conceitos de um domínio.

Os conceitos do modelo de referência proposto são apresentados graficamente na Figura 3.1 através de um diagrama de classes UML estereotipado, conforme sugerido pelo OMG (*Object Management Group*) em [OMG00]. O Apêndice B apresenta o código OWL [W3C11] da ontologia gerada.

Tabela 3.1 – Abordagens para a modelagem de contexto encontradas na literatura.

Abordagem	Aspectos positivos	Aspectos negativos	Recuperação das Informações	Exemplos de Propostas
Key-value models	• Fácil de gerenciar [Str04]. • Boa aplicabilidade em ambientes existentes [Str04].	• Não é uma estrutura sofisticada que permite o uso de algoritmos de recuperação de contexto eficientes [Str04]. • Difícil de validar [Str04]	Busca linear	Context Toolkit
Markup Scheme Models	• Fácil validação [Str04]. • Boa aplicabilidade em ambientes ubíquos cuja infra-estrutura seja centrada em marcação [Str04].	• A ambigüidade e as informações incompletas são tratadas só em nível da aplicação [Str04].	*Markup Query Language*	*Context Fabric*
Modelos gráficos	• Úteis para estruturar as informações [Str04]. • Aplicáveis na derivação de um modelo ER [Str04].	• Não são utilizados no nível de instâncias [Str04]. • Dificuldades para composições distribuídas [Str04]. • Baixo formalismo [Str04].	Transformação	
Modelos orientados a objetos	• Encapsulamento e reusabilidade [Str04] • São simples, fáceis de programar e eficientes [Roc05].	• Dificuldade de representar aspectos dinâmicos do contexto [Str04]. • O acesso a informações contextuais só é possível através de interfaces específicas [Str04]. • Necessita de infra-estruturas para suportar todas as operações sobre as informações contextuais [Roc05]. • Dificuldades para representar as características temporais das informações e expressar relações como dependência [Hen02]. • A invisibilidade (em conseqüência do encapsulamento) prejudica a formalização [Str04]. • Requerem um alto nível de concordância (*agreement*) entre as aplicações para poder ter interoperabilidade [Kru07].	*Inferencing*	
Modelos baseados em lógica	• Alto grau de formalismo [Str04].	• Dispositivos computacionais ubíquos geralmente não apresentam um *reasoner* lógico completo [Str04].	Algoritmo	
Modelos baseados em ontologias	• Rica expressividade e suporte aos aspectos de evolução das informações contextuais [Roc05] • Permitem a realização de inferência sobre as informações contextuais disponíveis [Hef03, Kru07]. • Permitem que entidades não projetadas para trabalhar em conjunto cooperem [Hef03].	• Restrito a ambientes capazes de trabalhar com OWL para a representação de conhecimento [Str04]. • Uso inapropriado ou modelagem errônea podem limitar o impacto e o uso de ontologias para modelar o contexto [Kru07].	*Reasoning*	*Gaia* *Cobra*

A classe `Context`, como o próprio nome sugere, representa a situação ou contexto de uma entidade. A definição do conceito contexto seguida neste trabalho está apresentada na Seção 2.1.1. O contexto é formado por uma série de informações contextuais (classe `ContextualInformation`). Cada informação contextual possui um valor (atributo `lastUpdatedValue`). A auto-associação `isCreatedBasedOn` (na classe `Context`) indica que o contexto atual de uma entidade pode ser gerado com base em seus contextos anteriores.

Uma entidade (classe `Entity`) representa qualquer coisa que se possa falar sobre. De acordo com Dey e Abowd [Dey99], uma entidade pode ser uma pessoa, um lugar ou um objeto. Para os autores deste trabalho, entidades podem representar também dispositivos, agentes de software e outros. Entidades podem se relacionar com outras entidades: elas podem trabalhar de forma cooperativa, podem prestar serviços umas para as outras, podem estar em localizações próximas, podem conter outras entidades, entre outros. A possibilidade de relacionamento entre as entidades é prevista pelo relacionamento `hasRelationshipWith` do modelo proposto.

No modelo, assim como em [Hen02], há dois tipos principais de informações: as informações atemporais (representadas pela classe `NonTemporalInformation`) e as informações temporais (classe `TemporalInformation`). Optou-se pela nomenclatura “temporal” e “atemporal” para enfatizar questões referentes ao tempo de vida das informações (em [Hen02] são utilizadas as nomenclaturas “estática” e “dinâmica”). O contexto atual de uma entidade pode ser formado de informações temporais e atemporais.

As informações temporais possuem três atributos: fator de confiança, precisão e indicador de hora de geração (*timestamp*). Os atributos fator de confiança e precisão devem ser medidos durante a aquisição ou geração da informação, uma vez que as informações contextuais podem apresentar falhas por razões como: conhecimento parcial do contexto, atraso entre a geração da informação e o seu uso, problemas nos sensores ou erros nos algoritmos e regras para deduzir novas informações. Tais atributos das informações contextuais temporais não são citados por Henricksen e co-autores em [Hen02].

Segundo Henricksen *et al.* [Hen02] (e corroborado por Cortese e co-autores [Cor05]), há três tipos de informações dinâmicas ou temporais: as informações sentidas (que no modelo proposto são produzidas pelos monitores de fontes de informação –

classe `InformationSourceMonitor`), as explicitadas (informadas explicitamente pelas entidades) e as inferidas (criadas a partir de interpretações sobre o conjunto de informações contextuais disponíveis). Os tipos de informações dinâmicas podem ser visualizados nas especializações da classe `TemporalInformation`.

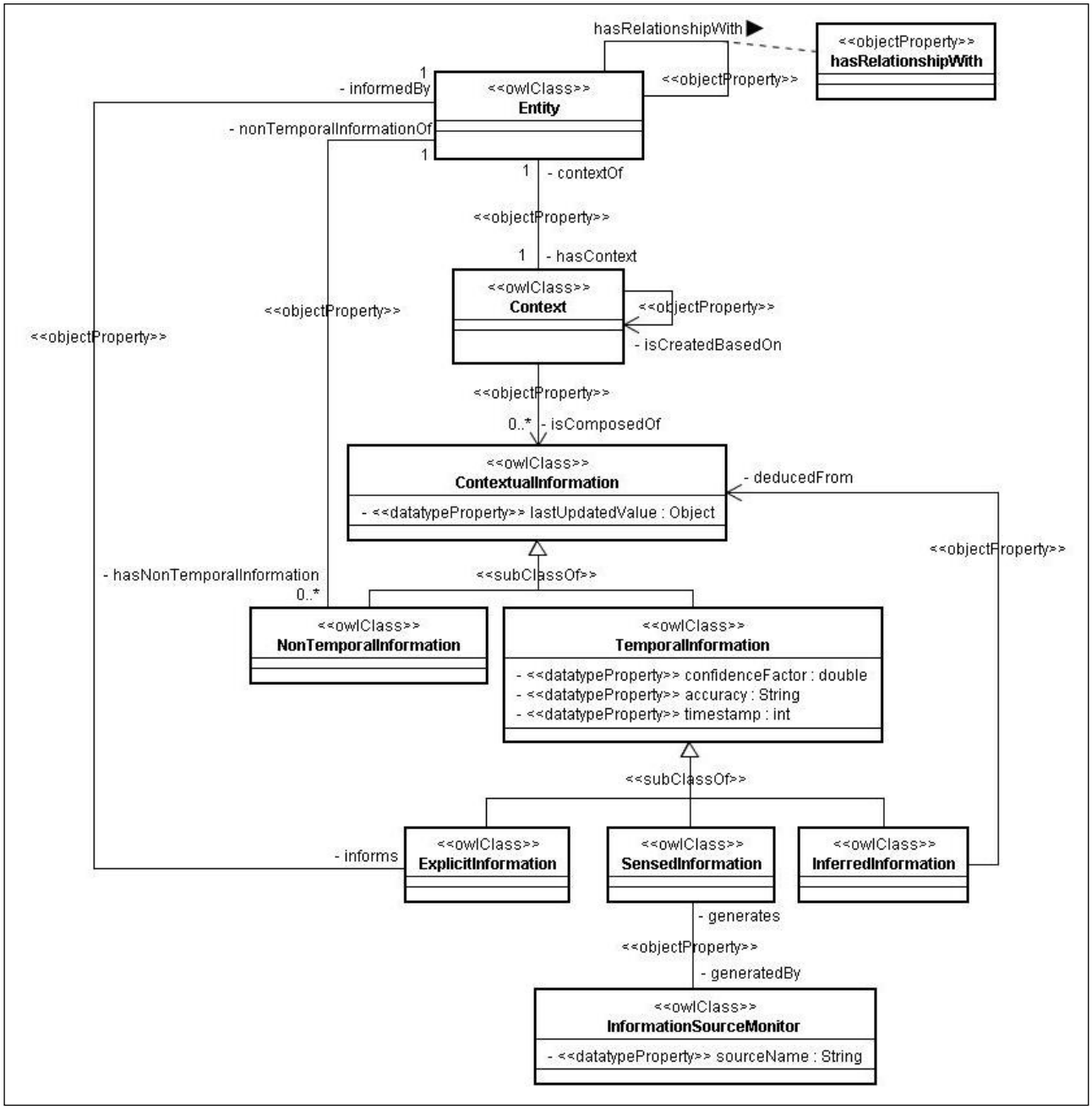

Figura 3.1 – Modelo de referência para a representação de informações contextuais proposto.

Na próxima subseção será apresentado um exemplo teórico de uso do modelo proposto. O exemplo mostra uma releitura do modelo apresentado por Li em [Li06] (Figura 3.2), que traz exemplos de informações contextuais para aplicações centradas em pessoas. Utilizou-se um modelo encontrado na literatura para mostrar que com o modelo de referência proposto é possível descrever outros modelos.

3.1.3.1. Exemplo de uso

A Figura 3.3 apresenta uma releitura do modelo de Li baseada no modelo de referência proposto. A maioria dos conceitos-chave propostos por Li (classes `Place`, `Person` e `Device`), foram mapeados como especializações da classe `Entity`. O conceito `Service` foi mapeado como um tipo de informação temporal da entidade `Device`, uma vez que se considerou que cada dispositivo oferece uma série de serviços já conhecidos *a priori*. Os demais atributos da classe `Device` foram transformados em informações atemporais da nova entidade `Device`. O mesmo aconteceu com o atributo da classe `Place` (`PlaceName` foi transformado em uma informação atemporal da nova entidade `Place`). Já os atributos da classe `Person` foram mapeados para dois diferentes tipos de informações da nova entidade `Person`. Os atributos `Name`, `Tel` e `Address` foram considerados informações atemporais. Já o atributo `Activity` foi considerado uma informação explícita, visto que a atividade precisa ser indicada pelo indivíduo durante a execução da aplicação.

A escolha do tipo de informação para representar a atividade de uma pessoa é uma questão bastante subjetiva e representa uma decisão de projeto. Neste caso, optou-se por classificar a atividade como uma informação explícita porque não é citado no modelo nenhum elemento que possibilite que essa informação seja sentida e também não há outras informações atemporais ou temporais a partir das quais a atividade de uma pessoa possa ser deduzida.

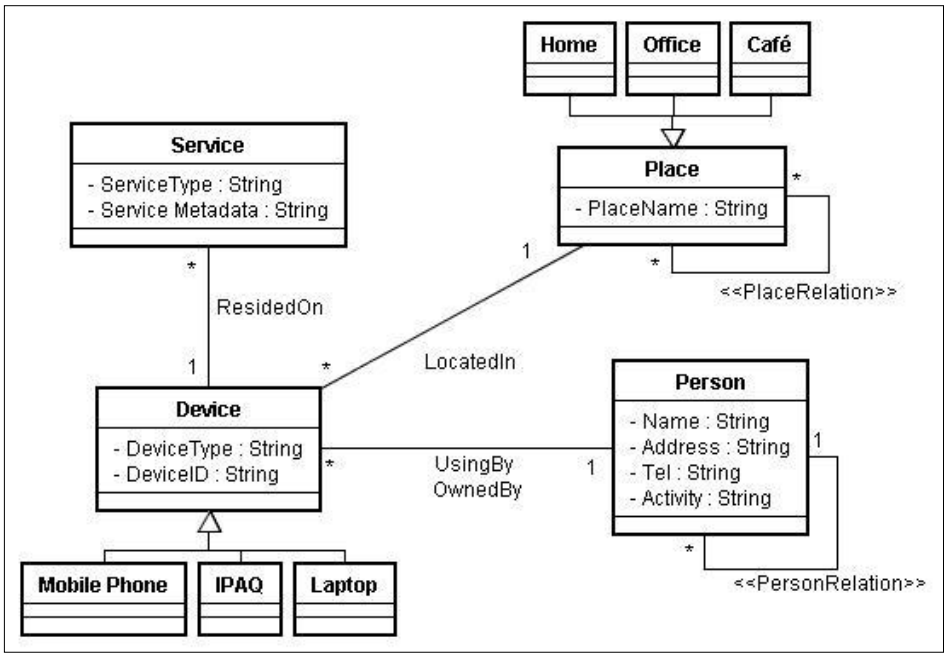

Figura 3.2 – Modelo de contexto proposto por Li em [Li06].

Os relacionamentos entre as entidades foram mantidos conforme indicado por Li. A existência de relacionamentos entre as entidades é prevista no modelo proposto pela auto-associação `hasRelationshipWith` da classe `Entity`. Assim, as relações entre as classes `Place`, `Device` e `Person` estão representadas na Figura 3.3 como sub-propriedades de `hasRelationshipWith` (relacionamento permitido na linguagem OWL). As especializações das classes `Place` e `Device` não estão representadas na figura, a fim de melhorar sua visibilidade.

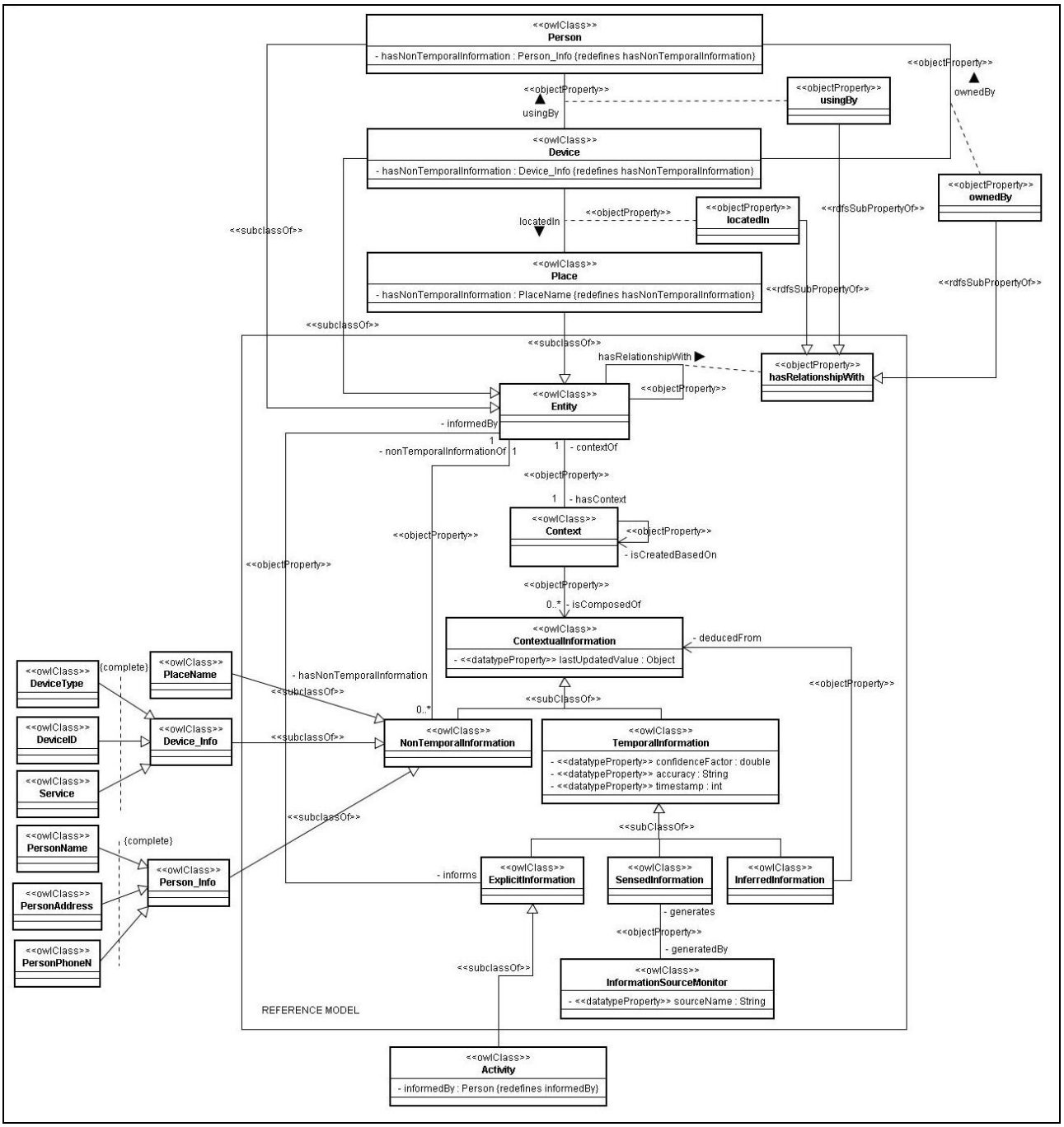

Figura 3.3 – Releitura do modelo de Li utilizando-se o modelo de referência proposto.

3.2. Definição do escopo de adaptação em arquiteturas de agentes

Para identificar as operações de adaptação[11] a serem realizadas na arquitetura de um agente e também os locais passíveis de adaptação em sua execução e estrutura, foi utilizado um metamodelo genérico para o desenvolvimento de SMAs chamado FAML [Bey09]. Esse metamodelo foi escolhido principalmente por não ser restrito a nenhum modelo arquitetural de agente ou tecnologia disponível para o desenvolvimento de SMAs (ele foi validado diante de outros metamodelos e metodologias para o desenvolvimento de SMAs, mas é independente deles).

No metamodelo FAML, os conceitos são distribuídos em dois conjuntos: conceitos em tempo de projeto e conceitos em tempo de execução (*runtime*). Segundo os autores, essa diferenciação permite indicar em que estágio do ciclo de vida do desenvolvimento um conceito particular será útil [Bey09]. Dentro de cada conjunto, os conceitos são divididos mais uma vez em dois escopos: o escopo de conceitos internos ao agente e o de conceitos externos ao agente. O Anexo A apresenta todos os diagramas e conceitos definidos pelo metamodelo FAML.

Neste estudo, como se pretendia adaptar o comportamento dos agentes de software ao contexto, a parte mais significativa do metamodelo FAML é a que compreende os conceitos presentes em tempo de execução na arquitetura interna dos agentes. Porém, muitos dos conceitos presentes em tempo de execução são gerados a partir de especificações definidas em tempo de projeto. Logo, foi necessário analisar tanto os conceitos definidos em tempo de projeto quanto os definidos em tempo de execução. Também, como se pretendia trabalhar com o conceito de informações contextuais, foi necessário estender o metamodelo, adicionando conceitos externos à arquitetura do agente.

A análise dos conceitos definidos pelo FAML, que é discutida na Seção 3.2.1, permitiu identificar as estruturas constituintes de um agente e seus relacionamentos e, a partir disto, puderam ser definidos os tipos de adaptação possíveis em um agente. A Seção 3.2.1 também descreve brevemente[12] as alterações realizadas no metamodelo FAML. Já na Seção 3.2.2 é apresentada a unificação das operações de adaptação de acordo com seus significados, o que permitiu definir 12 primitivas de caráter genérico. Por

[11] Para os autores deste trabalho, operações de adaptação são todas as operações que podem ser realizadas em um agente de forma a adaptar sua estrutura ou comportamento.
[12] A descrição e justificativa de todas as alterações realizadas no metamodelo FAML estão no Apêndice C.

fim, na Seção 3.2.3 são indicados os locais na execução do agente onde poderá haver adaptação.

3.2.1. Definição das operações de adaptação possíveis em agentes

Tendo como ponto de partida os conceitos definidos pelo metamodelo FAML na parte interna dos agentes tanto em tempo de projeto quanto em tempo de execução, foram identificadas cinco estruturas que podem ser alvo de adaptação em um agente, que são:

- O **estado mental** do agente;
- Os **objetivos** presentes no estado mental do agente;
- Os **planos de ação** utilizados para atingir um ou mais objetivos;
- As **ações** que constituem os planos de ação;
- Os **papéis** desempenhados pelo agente.

Os conceitos de estado mental e papel estão diretamente relacionados à classe `Agent` no diagrama de classes que descreve os conceitos internos de um agente em tempo de execução definido pelo FAML[13] (como mostra a Figura A.4 do anexo A). No diagrama, há uma associação do tipo agregação entre as classes `Agent` e `MentalState` e outra associação simples entre as classes `Agent` e `Role` (indicando que um agente executa um ou mais papéis). Se um agente tem um relacionamento direto com essas estruturas, elas devem ser consideradas em seu processo de adaptação.

O estado mental de um agente é composto por objetivos e crenças (a classe `MentalState` tem relacionamentos de agregação com as classes `AgentGoal` e `Belief`). Então, adaptar o estado mental de um agente significa adaptar o seu conjunto de objetivos e/ou crenças. Adaptações sobre tais conjuntos incluem, basicamente, a adição e remoção desses elementos do estado mental do agente. Em relação aos papéis, as adaptações incluem o comprometimento com o desempenho de um novo papel na sociedade ou o cancelamento da execução de um papel (com o qual o agente tinha se comprometido anteriormente).

Os objetivos presentes no estado mental de um agente, por sua vez, estão relacionados a um conjunto de planos (há uma associação entre as classes `AgentGoal` e

[13] No referido diagrama, há também um relacionamento de agregação entre as classes `Agent` e `Obligation`. Esse relacionamento, de acordo como os próprios autores do metamodelo, foi removido depois de revisões (mas a figura não foi atualizada de acordo com o texto). Considerando a menção direta que os autores fazem no artigo sobre a remoção de tal relacionamento, ele não foi considerado.

`Plan`). Além disso, os objetivos possuem alguns atributos, como o atributo `commited`, que indica se o agente está tentando atingir o objetivo ou não. Há adaptações que podem ser feitas no contexto de um objetivo, como é o caso da adição ou remoção de um plano da lista de planos aplicáveis e a interrupção ou cancelamento de seu alcance (quando esse estiver sendo perseguido pelo agente). Cabe salientar que, nas adaptações que dizem respeito ao estado mental do agente, é possível adicionar ou remover objetivos. Já nas adaptações onde a estrutura base é o próprio objetivo, podem ser alterados ou modificados os elementos ou informações que o compõem (ou que são referenciados por ele), como é o caso da lista de planos aplicáveis.

Já os planos são compostos por ações (a classe `Plan` tem um relacionamento de agregação com a classe `Action`) e podem utilizar recursos (o que é evidenciado pela associação entre as classes `PlanSpecification` e `PlanResourceSpecification` em tempo de projeto). Assim, é possível, por exemplo, adaptar um plano adicionando ou removendo ações ou ainda alterando o fluxo de execução dessas ações. Também, é possível iniciar a execução de uma ação no contexto de um plano. Já nas ações é possível alterar as pré e pós-condições (observe, na Figura A.2 – Anexo A, que, em tempo de projeto, as ações possuem os atributos `precondition` e `postcondition`).

Para cada uma das cinco estruturas base identificadas, foi definida, inicialmente, uma série de operações de adaptação, conforme indica a Tabela 3.2. Basicamente, para as adaptações estruturais, cuja definição será dada a seguir, verificou-se a necessidade de realização de três operações sobre os elementos constituintes de cada estrutura – as operações criar, atualizar e remover. Essas operações, acrescidas da operação consultar, são normalmente utilizadas para a persistência de dados (a operação consultar não foi utilizada porque sua execução não acarreta modificações na estrutura do agente). Além dessas, foram definidas as operações ditas comportamentais, que possibilitam a inicialização, o cancelamento, a interrupção e o recomeço de determinadas execuções realizadas pelos agentes e suas estruturas constituintes.

As operações de adaptação listadas na Tabela 3.2 possuem um tipo, que se refere ao tipo de modificação realizado quando executada a operação. Os tipos são:

- Estrutural: indica que há adição, remoção ou atualização de estruturas ou informações que constituem o agente ou outra estrutura relacionada a ele. Quando há uma adaptação estrutural, o resultado da adaptação pode ou

não ser observado no comportamento do agente no momento em que a adaptação é realizada.

- Comportamental: quando algo é realizado/ativado, mas não há alteração nas estruturas internas do agente (não há adição, remoção ou atualização de uma estrutura). A adaptação reflete-se imediatamente no comportamento do agente.
- Comportamental/Estrutural: quando algo é realizado/ativado e, em decorrência disto, novas estruturas são adicionadas ou removidas da arquitetura interna de um agente. Acontece, tipicamente, quando um agente se compromete a desempenhar um papel ou deixa de desempenhar um papel na sociedade.

Como pode ser visto nas linhas 21 e 22 da Tabela 3.2, neste trabalho os recursos são tratados somente no contexto de um plano. No metamodelo FAML original há certa confusão entre as estruturas que utilizam os recursos. Em tempo de projeto, diz-se que as especificações dos planos utilizam as especificações dos recursos (o conceito `PlanSpecification` tem um relacionamento chamado `Uses` com o conceito `PlanResourceSpecification`, que é uma especialização do conceito `ResourceSpecification`). Já em tempo de execução, não há qualquer relacionamento entre planos e recursos, sendo que os recursos passam a se relacionar diretamente com os agentes (o conceito `Agent` tem o relacionamento `Uses` com o conceito `Resource`). Para resolver essa inconsistência, o metamodelo foi alterado e o relacionamento entre planos e recursos foi mantido em tempo de execução (essa e outras alterações realizadas no metamodelo FAML estão detalhadas no Apêndice C).

No diagrama de classes internas ao agente em tempo de execução definido pelo metamodelo FAML, a classe `Agent` possui também duas associações com a classe `Communication`, cuja função é indicar as mensagens enviadas e recebidas pelo agente (a definição do conceito `Communication` é “composição de mais de uma mensagem”). Então, considerou-se também a possibilidade de adaptar determinados mecanismos e operações utilizados pelos agentes para a comunicação com outros agentes do sistema. São eles:

- Mecanismo para o **envio de mensagens** através do ambiente.
- Mecanismo para o **recebimento de mensagens** através do ambiente.

Tabela 3.2 – Operações de adaptação possíveis sobre as estruturas que compõem um agente.

Id	Estrutura Alvo	Operações possíveis	Tipo	Descrição
1	Estado mental do agente	commitToGoal	Comportamental	Indica que um desejo do agente virou uma intenção e, portanto, será perseguido.
2	Estado mental do agente	interruptGoalAchievement	Comportamental	Interrompe o alcance de um objetivo por tempo indeterminado, mantendo o estado.
3	Estado mental do agente	resumeGoalAchievement	Comportamental	Reinicia o alcance de um objetivo a partir do ponto onde houve a interrupção.
4	Estado mental do agente	abortGoalAchievement	Comportamental	Cancela o alcance do objetivo permanentemente.
5	Estado mental do agente	addAgentGoal	Estrutural	Adiciona um novo objetivo à especificação do agente.
6	Estado mental do agente	removeAgentGoal	Estrutural	Remove um objetivo da especificação do agente.
7	Estado mental do agente	addBelief	Estrutural	Adiciona uma nova crença ao modelo mental do agente.
8	Estado mental do agente	removeBelief	Estrutural	Remove uma crença do estado mental do agente.
9	Objetivo	executePlan	Comportamental	Inicializa a execução de um plano associado a um objetivo.
10	Objetivo	interruptPlanExecution	Comportamental	Interrompe a execução de um plano por tempo indeterminado mantendo o estado de execução do plano.
11	Objetivo	resumePlanExecution	Comportamental	Reinicia a execução de um plano.
12	Objetivo	abortPlanExecution	Comportamental	Cancela permanentemente a execução de um plano vinculado a um objetivo.
13	Objetivo	addPlan	Estrutural	Adiciona um novo plano para ser usado no alcance de um objetivo do agente.
14	Objetivo	removePlan	Estrutural	Remove um dos planos associados ao objetivo do agente.
15	Objetivo	updatePlanDescriptors	Estrutural	Substitui o conjunto de planos apropriados para uso no alcance de um objetivo
16	Objetivo	updatePlanSelectionCriterion	Estrutural	Substitui o conjunto de critérios utilizados para, em tempo de execução, selecionar ou abandonar planos relacionados a um objetivo.
17	Plano	changeFlow	Estrutural	Modifica o fluxo de ações de um plano, adicionando novas ações antes ou depois de outras já adicionadas.
18	Plano	removeAction	Estrutural	Remove uma ação do plano.
19	Plano	executeAction	Comportamental	Inicia a execução de uma ação do plano.
20	Plano	abortActionExecution	Comportamental	Cancela a execução de uma ação do plano.
21	Plano	addResource	Estrutural	Adiciona um novo recurso ao plano.
22	Plano	removeResource	Estrutural	Remove um recurso contido no plano.
23	Plano	updateFailureCondition	Estrutural	Substitui a condição de falha do plano.
24	Plano	updateSuccessCondition	Estrutural	Substitui a condição de sucesso do plano.
25	Plano	updateGoalCondition	Estrutural	Permite atualizar o conjunto de objetivos em que o plano pode ser utilizado.
26	Ação	updatePreCondition	Estrutural	Modifica as pré-condições da ação.
27	Ação	updatePostCondition	Estrutural	Modifica as pós-condições da ação.
28	Papel desempenhado pelo agente	commitToRole	Estrutural / Comportamental	Indica que o agente se comprometeu a desempenhar um novo papel na sociedade.
29	Papel desempenhado pelo agente	cancelRole	Estrutural / Comportamental	Indica que o agente deixou de desempenhar um papel na sociedade.

Para permitir adaptações sobre o mecanismo para recebimento de mensagens através do ambiente, foram inseridos dois novos conceitos ao metamodelo FAML: `Sensor` e `SensorDefinition`. Em [Rib02], é dito que “um sensor pode ser visto como uma ligação entre um agente e o ambiente, que serve para filtrar informações recebidas em busca de algum padrão pré-estabelecido”. Ou seja, agentes possuem sensores para capturar determinados tipos de informações que trafegam pelo ambiente. Assim, um sensor pode ser definido como “elemento utilizado para capturar determinados tipos de informação do ambiente”.

O conceito `SensorDefinition` foi incluído para especificar a estrutura de um dado sensor em tempo de projeto e pode ser definido como “especificação da estrutura de um determinado sensor, o que inclui um e somente um padrão para captura de informações”. Em tempo de projeto, a definição de um agente também possui definições de sensores (o conceito `AgentDefinition` tem um relacionamento do tipo agregação com o conceito `SensorDefinition`). Em tempo de execução, um agente é também uma agregação de zero ou mais sensores (o conceito `Agent` tem um relacionamento do tipo agregação com o conceito `Sensor`).

A Tabela 3.3 apresenta as operações que podem ser realizadas sobre os mecanismos para o envio e recebimento de informações através do ambiente. Como pode ser visto na tabela, foram identificadas 10 operações sobre tais mecanismos, sendo 7 comportamentais e 3 estruturais. As operações do tipo estrutural (com identificadores 33, 34 e 35) permitem realizar as operações criar, atualizar e remover sobre o conjunto de sensores de um agente (cabe lembrar que, com a atualização do metamodelo FAML, a classe `Agent` passou a ser, também, uma agregação de objetos da classe `Sensor`). Além disso, é possível ativar ou desativar um dos sensores do agente (o que é feito pelas operações de identificação 36 e 37). Quando desativado, um sensor não filtra as mensagens recebidas pelo agente.

Já as operações com identificação 38 e 39 permitem bloquear (e posteriormente desbloquear) o recebimento de quaisquer informações através do ambiente. Bloquear a recepção de informações (operação 38) significa desativar todos os sensores disponíveis na arquitetura interna do agente, deixando-o incomunicável com o mundo exterior. O bloqueio de tal mecanismo pode ser importante quando o agente necessita, por exemplo, reservar todo o seu poder de processamento para realizar determinada tarefa.

As operações com identificação 31 e 32 permitem, respectivamente, bloquear e desbloquear o envio de mensagens através do ambiente. Enquanto tal mecanismo está bloqueado, nenhuma mensagem pode ser enviada a outros agentes do sistema. Já a operação `sendMessage` (de identificação 30 na Tabela 3.3) permite que o agente envie uma mensagem a outros agentes como resultado de sua adaptação.

Considerando-se as operações descritas em ambas as Tabelas 3.2 e 3.3, com base no metamodelo FAML foram identificadas 39 diferentes operações de adaptação sobre estruturas e mecanismos dos agentes. Com elas é possível, por exemplo, adicionar novas ações a um plano, ativar o alcance de um objetivo ou enviar uma mensagem a outro agente do sistema.

Tabela 3.3 – Operações de adaptação possíveis sobre o envio e recebimento de mensagens.

Id	Estrutura Alvo	Operações possíveis	Tipo	Descrição
30	Agente / Envio de mensagens	sendMessage	Comportamental	Permite aos agentes se comunicar através do envio de mensagens.
31	Agente / Envio de mensagens	blockMessageSending	Comportamental	Interrompe o envio de mensagens
32	Agente / Envio de mensagens	unblockMessageSending	Comportamental	Libera o envio de mensagens a outros agentes do sistema.
33	Agente / Recebimento de mensagens	addSensor	Estrutural	Adiciona um novo sensor a estrutura do agente.
34	Agente / Recebimento de mensagens	removeSensor	Estrutural	Remove um sensor da estrutura do agente.
35	Agente / Recebimento de mensagens	updateSensorPattern	Estrutural	Modifica o padrão de mensagens observado por um sensor.
36	Agente / Recebimento de mensagens	blockSensor	Comportamental	Bloqueia o recebimento de informações do ambiente via o sensor especificado.
37	Agente / Recebimento de mensagens	unblockSensor	Comportamental	Desbloqueia o recebimento de informações do ambiente via o sensor especificado.
38	Agente / Recebimento de mensagens	blockInformationReception	Comportamental	Bloqueia o recebimento de quaisquer informações do ambiente.
39	Agente / Recebimento de mensagens	unblockInformationReception	Comportamental	Desbloqueia o recebimento de informações do ambiente.

3.2.2. Definição das primitivas genéricas

Muitas das operações definidas nas Tabelas 3.2 e 3.3 possuem semântica similar. Por exemplo, tanto a operação `addPlan` (operação 13 definida na Tabela 3.2) quanto a operação `addSensor` (operação 33 definida na Tabela 3.3) objetivam adicionar uma nova estrutura a uma estrutura base: a operação `addPlan` objetiva adicionar um plano a um

objetivo (conceitos `Plan` e `AgentGoal`) enquanto a operação `addSensor` objetiva adicionar um novo sensor a um agente (conceitos `Sensor` e `Agent`).

De forma a agrupar as operações com semântica similar, foi feita uma análise nas operações definidas que resultou em um conjunto de 12 primitivas genéricas. São elas: `add`, `remove`, `update`, `changeFlow`, `execute`, `abort`, `interrupt`, `resume`, `commitTo`, `block`, `unblock` e `sendMessage`. A primitiva `replace` é apenas uma primitiva facilitadora e, portanto, não é contabilizada (ela pode ser substituída pelas primitivas `add` e `remove`). As primitivas genéricas estão apresentadas na Tabela 3.4.

As colunas da Tabela 3.4 apresentam o tipo, as variações e os usos possíveis de cada primitiva genérica definida, onde: o *tipo* indica se com o uso da primitiva será feita uma adaptação estrutural, comportamental ou estrutural/comportamental (conforme definições apresentadas anteriormente); as *variações da primitiva* indicam os parâmetros aceitos pela primitiva genérica; e a coluna *usos possíveis da primitiva* indica como a primitiva pode ser utilizada considerando-se as estruturas definidas pelo metamodelo FAML. Na tabela, sempre que era especificado o item `Parameters` nas variações de uma primitiva, foi indicado, dentro de chaves, o tipo de parâmetro esperado em cada uso. Já quando era solicitado um tipo (um item `Type`), foram utilizadas aspas duplas para indicar as palavras-reservadas aplicáveis em cada uso da primitiva. Os usos possíveis das primitivas são, na verdade, uma releitura das Tabelas 3.2 e 3.3, sendo que existem 39 possíveis usos para as primitivas genéricas. Nos parágrafos a seguir, cada primitiva genérica é descrita.

A primitiva genérica **`add`** pode ser utilizada em cinco situações distintas para realizar modificações estruturais na arquitetura de um agente. Ela pode ser utilizada para adicionar objetivos e crenças ao estado mental de um agente e sensores a sua arquitetura interna. Aos planos, a primitiva permite adicionar recursos. Por fim, é possível adicionar novos planos a um objetivo informando os critérios[14] que devem ser utilizados para a sua seleção. A seguir, é mostrado um exemplo de uso da primitiva `add`. Como as primitivas genéricas podem ser aplicadas a diferentes elementos, para melhorar o entendimento, sempre há uma referência a classe do elemento antes de seu identificador. No exemplo, considera-se que um agente de identificação *SellerAgent* deseja vender livros e, para tanto, necessita adicionar o objetivo de identificação *SellBookGoal.*

[14] De acordo com os autores do metamodelo FAML, os critérios são utilizados para selecionar, em tempo de execução, o plano mais apropriado para o alcance de um objetivo [Bey09].

```
add (Agent: SellerAgent, Goal: SellBookGoal)
```

A primitiva genérica **`remove`** pode ser usada em seis situações e também realiza modificações estruturais na arquitetura de um agente. Utilizando-a com planos como estrutura base é possível remover ações e recursos. Já se a estrutura base for um objetivo, é possível remover planos de ação. Dos agentes é possível remover três estruturas: objetivos, crenças e sensores. O exemplo a seguir mostra a remoção do objetivo com identificação *SellBookGoal* do agente chamado *SellerAgent*.

```
remove (Agent: SellerAgent, Goal: SellBookGoal)
```

A primitiva **`update`** permite atualizar o valor de determinadas características ou atributos de uma estrutura base. Para essa primitiva, foram identificados oito possíveis usos. Inicialmente, é possível atualizar as condições de falha e de sucesso de um plano. De acordo com Beydoun e co-autores [Bey09], o plano contém condições de falha e de sucesso tanto em tempo de projeto (na classe `PlanSpecification`) quanto em tempo de execução. Consideram-se as condições de falha e de sucesso como sentenças lógicas onde também é possível indicar os valores `true` (a condição será sempre verdadeira) ou `false` (condição sempre falsa). O exemplo de uso a seguir mostra a atualização da condição de sucesso do plano *SellBookPlan* (a execução do plano citado é considerada um sucesso quando o preço final do livro negociado é maior do que o preço mínimo acrescido de 10%).

```
update (Plan: SellBookPlan, "SuccessCondition", saleValue > mPrice
                                + mPrice*0,1)
```

Com a primitiva `update` também é possível modificar o conjunto de objetivos ao qual o plano é aplicável (as especificações dos planos possuem um atributo chamado `GoalCondition`). Nas ações podem ser atualizadas as pré e pós-condições e nos sensores é possível alterar o padrão de percepção. Já nos objetivos é possível atualizar o atributo `planDescriptors`, que especifica os planos apropriados para uso no alcance do objetivo. Muitos planos podem ser apropriados para uso em um objetivo e a escolha do plano deve ser feita em tempo de execução [Bey09]. Os critérios para seleção de um plano também podem ser atualizados utilizando-se a primitiva genérica `update`.

A primitiva **`changeFlow`** permite adicionar novas ações a um plano e/ou modificar o seu fluxo de ações. No exemplo abaixo, utiliza-se a primitiva `changeFlow` para adicionar a ação `GenerateReceiptAction` ao plano de identificação *SellBookPlan*. A

nova ação será adicionada após outra ação já pertencente ao plano (a saber, a ação `ConcludeSaleAction`).

```
changeFlow (Plan: SellBookPlan, Action: ConcludeSaleAction,
                Action: GenerateReceiptAction)
```

Para ativar a execução de planos e ações, foi definida a primitiva **`execute`**. Na execução de ações pode ser necessário especificar determinados parâmetros, por isto foram definidas duas variações para essa primitiva. Ainda considerando como exemplo o agente de nome *SellerAgent* (que deseja vender livros), abaixo são mostradas duas aplicações da primitiva `execute`. Na primeira delas, a primitiva é utilizada para iniciar a execução do plano *SellBookPlan*, no contexto do objetivo *SellBookGoal*. A execução de tal plano poderia ser iniciada, por exemplo, toda vez que fosse recebida uma mensagem de solicitação de livros. O segundo exemplo mostra como a primitiva pode ser utilizada para iniciar a execução de uma ação que necessita de parâmetros na ativação (para ativar a ação `SellBookAction`, é preciso informar como parâmetro o nome do livro que se deseja vender).

```
(a)  execute (Goal: SellBookGoal, Plan: SellBookPlan)
(b)  execute (Plan: SellBookPlan, Action: SellBookAction,
     {Book: The Lost Symbol})
```

A primitiva **`commitTo`** permite que um agente se comprometa a alcançar um objetivo ou comece a desempenhar um novo papel na sociedade. A primitiva `commitTo` é considerada estrutural e comportamental, pois quando um agente começa a desempenhar um papel, ele passa a ter novos objetivos, o que implica em modificações nos elementos que o constituem. O exemplo a seguir ilustra o momento em que o agente *SellerAgent* passa a desempenhar o papel de *SellerManager*.

```
commitTo (Agent: SellerAgent, Role: SellerManager)
```

O cancelamento da execução de determinadas estruturas é garantido pela primitiva **`abort`**. Com essa primitiva é possível: (1) cancelar a execução de um plano no contexto de um objetivo; (2) cancelar a execução de uma ação no contexto de um plano; (3) desistir de alcançar um objetivo (um agente pode desistir de alcançar um objetivo); e (4) deixar de desempenhar um papel na sociedade (um agente pode deixar de desempenhar um papel). Assim como a primitiva `commitTo`, a primitiva `abort` é do tipo comportamental e estrutural (quando um agente deixa de desempenhar um papel, ele também deixa de ter certos objetivos, o que implica em modificações estruturais em sua

arquitetura). Por exemplo, o cancelamento do alcance do objetivo *SellBookGoal*, do agente *SellerAgent*, é feito da seguinte forma com a primitiva `abort`:

```
abort (Agent: SellerAgent, Goal: SellBookGoal)
```

A interrupção temporária da execução de algumas estruturas é garantida pela primitiva **`interrupt`**. Com ela é possível paralisar temporariamente a execução de um plano ou deixar de tentar alcançar um objetivo. Quando a primitiva `interrupt` é acionada, o estado da estrutura no momento da interrupção é armazenado (o que não acontece quando é utilizada a primitiva `abort`). Para reiniciar as execuções interrompidas, foi definida a primitiva **`resume`**. Com ela um agente pode retomar o desejo de alcançar um objetivo ou reiniciar a execução de um plano.

Os exemplos a seguir mostram a interrupção e, posteriormente, a reativação do objetivo *SellBookGoal*. É importante salientar que, tanto a primitiva `abort` quanto a primitiva `interrupt` tem efeito em cascata quando aplicadas a um objetivo ou plano. Assim, se o alcance de um objetivo é cancelado ou interrompido, o mesmo acontece com o plano que está sendo executado e com as ações do plano. Quando a interrupção ou cancelamento é de um plano, todas as suas ações são também canceladas.

```
interrupt (Agent: SellerAgent, Goal: SellBookGoal)
resume (Agent: SellerAgent, Goal: SellBookGoal)
```

As primitivas **`block`** e **`unblock`** são aplicadas, basicamente, aos mecanismos disponíveis no agente para envio e recebimento de informações através do ambiente. Com a primitiva `block` é possível bloquear o envio de quaisquer informações através do ambiente ou bloquear apenas o envio de determinados tipos de mensagens (o tipo de mensagem é representado pelo conceito `MessageSchema` definido no metamodelo FAML). Pode-se fazer o mesmo com os mecanismos para recebimento de informações: bloquear o recebimento de quaisquer informações ou apenas de informações capturadas por determinado sensor. Para cada tipo de bloqueio possível, há um desbloqueio correspondente. Nos exemplos a seguir, primeiro é bloqueado o envio de mensagens cujo assunto contenha a *string* “*Sale*” (o que pode acontecer quando não há mais produtos em estoque) e depois o envio de mensagens com o mesmo assunto é liberado.

```
block (Agent: SellerAgent, “MessageSending”, {message.subject = “Sale”})

unblock (Agent: SellerAgent, “MessageSending”, {message.subject =
“Sale”})
```

Tabela 3.4 – Primitivas genéricas para a adaptação de agentes.

Primitiva Genérica	Tipo	Variações da Primitiva	Descrição	Usos possíveis da Primitiva considerando o metamodelo FAML
add	Estrutural	add(BaseStructure, Structure)	Adiciona uma nova estrutura a estrutura base.	`add(Agent, Goal)`
				`add(Agent, Belief)`
				`add(Agent, Sensor)`
				`add(Plan, Resource)`
		add(BaseStructure, Structure, Parameters)	Adiciona uma nova estrutura a estrutura base informando uma lista de parâmetros.	`add(Goal, Plan, {SelectionCriterion})`
remove	Estrutural	remove(BaseStructure, Structure)	Remove permanentemente uma estrutura da estrutura base.	`remove(Plan, Action)`
				`remove(Plan, Resource)`
				`remove(Agent, Goal)`
				`remove(Goal, Plan)`
				`remove(Agent, Belief)`
				`remove(Agent, Sensor)`
update	Estrutural	update(BaseStructure, AttributeType, Value)	Atualiza o valor de determinado atributo da estrutura base.	`update(Plan, "FailureCondition", value)`
				`update(Plan, "SuccessCondition", value)`
				`update(Plan, "GoalCondition", value)`
				`update(Action, "Precondition", value)`
				`update(Action, "PostCondition", value)`
				`update(Sensor, "Pattern" value)`
				`update(Goal, "PlanDescriptors", value)`
				`update(Goal, "PlanSelectionCriterion", value)`
changeFlow	Estrutural	changeFlow(Plan, Action, NextAction)	Modifica o fluxo de ações de um plano. Se alguma das ações indicadas não estiver no plano, ela será adicionada. Caso nenhuma das ações estiver no plano, a ação informada em `Action` será colocada no início do fluxo. O parâmetro `NextAction` pode ser vazio (no caso de plano com apenas uma ação).	`changeFlow(Plan, Action, NextAction)`
execute	Comportamental	execute(BaseStructure, Structure)	Inicializa a execução de uma estrutura vinculada a uma estrutura base.	`execute(Goal, Plan)`
		execute(BaseStructure, Structure, Parameters)	Inicializa a execução de uma estrutura vinculada a uma estrutura base informando parâmetros.	`execute(Plan, Action, {Parameters})`

Primitiva Genérica	Tipo	Variações da Primitiva	Descrição	Usos possíveis da Primitiva considerando o metamodelo FAML
abort	Comportamental / Estrutural	abort(BaseStructure, Structure)	Cancela ou anula a execução/alcance de uma estrutura no contexto da estrutura base. Não mantém o estado de execução.	`abort(Goal, Plan)`
				`abort(Agent, Goal)`
				`abort(Agent, Role)`
interrupt	Comportamental	interrupt(BaseStructure, Structure)	Interrompe a execução de uma estrutura por tempo indeterminado mantendo o estado da estrutura no momento da interrupção.	`interrupt(Agent, Goal)`
				`interrupt(Goal, Plan)`
resume	Comportamental	resume(BaseStructure, Structure)	Reinicia a execução/alcance de uma estrutura vinculada a uma estrutura base após sua interrupção.	`resume(Agent, Goal)`
				`resume(Goal, Plan)`
commitTo	Comportamental / Estrutural	commitTo(BaseStructure, Structure)	Indica que uma estrutura base se comprometeu a exercer/alcançar determinada estrutura.	`commitTo(Agent, Role)`
				`commitTo(Agent, Goal)`
block	Comportamental	block(BaseStructure, BlockType, Parameter)	Bloqueia determinado mecanismo da estrutura base por período indeterminado (o mecanismo é indicado por BlockType). Quando indicado algum parâmetro, o mecanismo é bloqueado parcialmente.	`block(Agent, "MessageSending", {Pattern})`
				`block(Agent, "InformationReception",{Sensor})`
unblock	Comportamental	unblock(BaseStructure, BlockType)	Desbloqueia determinado mecanismo da estrutura base (o mecanismo é indicado por BlockType). Quando indicado algum parâmetro, o mecanismo é desbloqueado parcialmente.	`unblock(Agent, "MessageSending", {Pattern})`
				`unblock(Agent,"InformationReception", {Sensor})`
sendMessage	Comportamental	sendMessage(Sender, Recipients, MessageSchema, Parameters)	Executa um tipo especial de ação que permite aos agentes se comunicar através do envio de mensagens. É necessário informar o remetente, os destinatários, o *template* (*MessageSchema*) que será utilizado para envio da mensagem e os parâmetros da mensagem.	`sendMessage(Sender, Recipients, MessageSchema, {Parameters})`
replace	Estrutural	replace(BaseStructure, Structure, NewStructure)	Substitui permanentemente uma estrutura já contida na estrutura base por outra estrutura.	`replace(Plan, Action, NewAction)`
				`replace(Goal, Plan, NewPlan)`
				`replace(Agent, Belief, NewBelief)`
				`replace(Plan, Resource, NewResource)`

A primitiva **`sendMessage`** permite aos agentes se comunicar através do envio de mensagens. Para enviar uma mensagem, é necessário informar o remetente, os destinatários, o *template* (`MessageSchema`) que será utilizado para envio da mensagem e os parâmetros da mensagem (de acordo com o *template* selecionado). Para enviar uma mensagem de oferta de livros entre os agentes *SellerAgent* e *CustomerAgent*, a primitiva `sendMessage` poderia ser utilizada da seguinte forma:

```
sendMessage (Agent: SellerAgent, Agent: CustomerAgent, SOAPMessage,
      {subject = "Sale"; content = "Valentine´s Day Sale. Enjoy!"})
```

Por fim, a primitiva genérica **replace** permite substituir ações e recursos de um plano, planos de um objetivo e crenças do estado mental de um agente. A seguir, é ilustrada a substituição do plano de identificação *SellBookPlan* pelo plano de identificação *SellBookSalePlan* no contexto do objetivo *SellBookGoal*.

```
replace(Goal: SellBookGoal, Plan: SellBookPlan, Plan: SellBookSalePlan)
```

Como poderá ser visto na próxima subseção, as primitivas genéricas definidas podem ser utilizadas em duas situações: (i) para indicar a adaptação que deverá ser introduzida quando o contexto atingir determinado estado ou ocorrer determinado evento interno na arquitetura do agente; (ii) para referenciar locais passíveis de adaptação na arquitetura de um agente, visto que os eventos internos são gerados a partir da execução das primitivas (em outras palavras, a execução ou chamada de uma primitiva corresponde a um ponto na execução ou estrutura de um agente em que pode haver interceptação e inserção de comportamento adaptado).

3.2.3. Definição dos locais passíveis de adaptação

No modelo proposto, um agente pode iniciar seu processo de adaptação quando for executada qualquer uma das primitivas definidas. Assim, pode-se iniciar uma adaptação quando um novo plano é adicionado a um objetivo, quando o agente se compromete a alcançar um objetivo ou a desempenhar um novo papel, quando uma crença é adicionada ou atualizada no estado mental do agente, entre outros. Deste modo, além das informações contextuais (que também são partes constituintes das crenças do agente) pode-se utilizar qualquer outra informação relacionada à estrutura ou à execução do agente para ativar o seu processo de adaptação.

Para indicar a execução ou chamada de uma primitiva, foi definido o conceito de evento interno no metamodelo FAML (o metamodelo não previa o gerenciamento de

eventos internos ao agente, apenas eventos externos). Cada execução de primitiva resulta na criação de um evento interno, sendo que foi definido um tipo de evento interno para cada primitiva. São esses eventos que disparam o processo de adaptação do agente. Os eventos, além de um tipo, trazem consigo outras informações, como as estruturas alteradas ou adicionadas.

Por padrão, os eventos internos são gerados ao final da execução de uma primitiva. Assim, quando o evento é gerado, o efeito da primitiva já pôde ser percebido no agente. Quando a execução das primitivas é praticamente instantânea, como nos casos das primitivas estruturais e algumas comportamentais, o tempo discorrido entre a execução da primitiva e a geração do evento é desprezível. Porém, há casos onde é necessário indicar o início e o término de uma execução.

As primitivas `execute` e `commitTo` são os principais casos a serem considerados devido ao tempo em que podem permanecer em execução na arquitetura de um agente. Nestes dois casos não bastaria gerar um evento ao final da execução ou comprometimento com o alcance de algo: é preciso saber também quando o agente começou a executar o plano/ação ou a desempenhar o papel. Para resolver esse problema, foram criados três novos tipos de eventos internos: `executionStart`, `executionEnd` e `goalAchieved`.

Quando é executada a primitiva `execute` (qualquer uma das variações), um evento do tipo `executionStart` é criado. Esse evento permanece ativo até que um evento do tipo `executionEnd` seja gerado para indicar o término da execução do plano ou ação. O evento `executionEnd` armazena, além da estrutura base, da estrutura e dos parâmetros utilizados (quando aplicável), o resultado obtido na execução (se satisfatória ou não). Com a criação dos eventos `executionStart` e `executionEnd` pode-se indicar que determinada adaptação deve ser realizada enquanto algo estiver sendo executado ao apenas ao fim da execução. Devido a esses dois tipos de eventos, não são criados eventos do tipo `execute` no modelo proposto.

O evento `goalAchieved` foi definido especialmente para indicar o alcance de um objetivo. Quando um agente indica que deseja alcançar um objetivo, é criado um evento do tipo `commitTo` (conforme primitiva genérica definida) e quando ele enfim o alcança é gerado um evento do tipo `goalAchieved`. O evento `goalAchieved` também registra o grau de satisfação atingido durante o alcance do objetivo. Eventos do tipo `goalAchieved` não são gerados quando a primitiva `commitTo` é utilizada para indicar

que um agente passou a desempenhar um papel (a saber "`commitTo (Agent, Role)`"), porque um agente não "atinge" um papel, apenas o desempenha ou deixa de desempenhar. Para indicar que um agente deixou de desempenhar um papel, são gerados eventos do tipo `abort`.

Além dos eventos criados para sinalizar o início e o término de determinadas operações, outros dois tipos de eventos relacionados exclusivamente a agentes adaptativos ao contexto foram definidos. São eles: o `contextUpdate` e o `adaptationDone`.

O evento `contextUpdate` foi definido para auxiliar na especificação do contexto relevante para determinada adaptação. Embora as informações do contexto corrente do agente fiquem armazenadas juntamente com suas crenças (cujas alterações são indicadas por eventos do tipo `add`), considerou-se pertinente definir um novo tipo de evento para evidenciar aspectos relacionados ao contexto. Assim, há uma clara distinção entre as informações de contexto e as informações regulares utilizadas para a tomada de decisão nos planos e ações. O evento `adaptationDone` foi definido para sinalizar quando um agente é alvo de adaptações. Esse evento armazena informações sobre o agente que sofreu adaptação e também sobre a estrutura que foi ativada para a realização da adaptação (como será debatido na próxima seção, a estrutura ao qual o evento se refere é chamada de adaptador).

A Tabela 3.5 sumariza os tipos de eventos definidos no modelo e as informações armazenadas por cada um deles (os tipos de eventos que não tem mapeamento direto com as primitivas genéricas definidas estão com preenchimento). Abaixo, há alguns exemplos de eventos internos considerando-se o agente de nome *SellerAgent* e a utilização das primitivas citadas ao longo do texto.

```
(a) add (Agent: SellerAgent, Goal: SellBookGoal)
(b) executionStart (Goal: SellBookGoal, Plan: SellBookPlan)
(c) goalAchieved (Agent: SellerAgent, Goal: SellBookGoal, "Good")
(d) contextUpdate (Interpreter:Sales, =, "growing up")
(e) adaptationDone (Agent: SellerAgent, Adaptor: SalesContextAdaptor)
(f) remove (Agent: SellerAgent, Goal: SellBookGoal)
```

O exemplo (a) apresenta o evento que é gerado quando executada a primitiva `add`. No exemplo, é adicionado o objetivo *SellBookGoal* ao agente *SellerAgent*. Já o

exemplo (b) mostra o evento gerado ao início da execução de um plano (no exemplo, é iniciada a execução do plano *SellBookPlan* no contexto do objetivo *SellBookGoal*). O exemplo (c) apresenta o evento gerado quando um agente alcança um objetivo (no terceiro elemento da n-tupla está o grau de satisfação atingido). Os exemplos (d) e (e) mostram a utilização dos eventos definidos exclusivamente para agentes adaptativos ao contexto. O exemplo (d) indica que, em determinado momento, o interpretador de nome *Sales* possui como valor a *string* "*growing up*". Já o evento (e), indica que o agente *SellerAgent* sofreu adaptação quando o adaptador de identificação *SalesContextAdaptor* foi ativado. Por fim, o evento (f) mostra a remoção de um objetivo do estado mental de um agente. Em todos os exemplos, foi omitido o valor do *timestamp*.

Por fim, cabe salientar que os eventos internos não são nem consumidos nem descartados depois do uso, ao invés disto, são armazenados em uma estrutura chamada `AgentHistory`. Na verdade, os eventos internos ficam ativos durante determinado tempo (tempo de vida) e, depois de transcorrido esse tempo, são considerados eventos "antigos", e então passam a fazer parte do histórico de eventos do agente. Os eventos do tipo `contextUpdate`, `executionStart` e `commitTo` são uma exceção a essa regra, uma vez que permanecem ativos até que, respectivamente, o contexto seja alterado, a execução de um plano ou ação seja finalizada ou o agente alcance um objetivo (ou deixe de desempenhar um papel). Com o armazenamento do histórico de eventos é possível tentar prever situações no futuro ou até mesmo modificar o mecanismo de adaptação, tornando-o mais preciso. Considera-se que o próprio agente terá um mecanismo para gerenciar os eventos internos e que esse mecanismo (bem como os próprios eventos internos) não será alvo de adaptação.

3.3. Visão geral dos conceitos e mecanismos da arquitetura K2

Antes de descrever o modelo conceitual da arquitetura desenvolvida, é preciso dar uma idéia geral das estruturas e mecanismos por ela utilizados. A Figura 3.4 ilustra as principais estruturas definidas na arquitetura **K2**. Para que os agentes tivessem ciência do contexto no qual estão inseridos, foi necessário fornecer uma infra-estrutura para o gerenciamento das informações contextuais. Na arquitetura definida, três estruturas são responsáveis pela coleta, processamento e distribuição das informações contextuais, conforme descrito a seguir.

Tabela 3.5 – Tipos de eventos internos que podem ocorrer na arquitetura de um agente adaptativo ao contexto.

Tipo de Evento	Primitiva Relacionada (com variações)	Momento de criação	Descrição
add	add (BaseStructure, Structure) add (BaseStructure, Structure, Parameters)	Ao término da operação de adição	Indica que uma nova estrutura foi adicionada a uma estrutura base.
remove	remove (BaseStructure, Structure)	Ao término da operação de remoção	Indica que uma estrutura foi removida de uma estrutura base.
update	update (BaseStructure, AttributeType, Value)	Ao término da operação de atualização	Indica que um atributo de uma estrutura base foi atualizado.
changeFlow	changeFlow(Plan, Action, NextAction)	Ao término da operação de mudança de fluxo	Indica que o fluxo de ações de um plano foi alterado.
executionStart	execute (BaseStructure, Structure) execute (BaseStructure, Structure, Parameters)	No início da operação de execução	Indica que uma estrutura começou a ser executada.
executionEnd	execute (BaseStructure, Structure) execute (BaseStructure, Structure, Parameters)	Ao término da operação de execução	Indica que terminou a execução de uma estrutura que estava sendo executada.
abort	abort (BaseStructure, Structure)	Ao término da operação de cancelamento	Indica que uma estrutura teve sua execução cancelada.
interrupt	interrupt (BaseStructure, Structure)	Ao término da operação de interrupção	Indica que uma estrutura teve sua execução interrompida.
resume	resume (BaseStructure, Structure)	Ao término da operação de reinicio	Indica que uma estrutura teve sua execução reiniciada.
commitTo	commitTo (BaseStructure, Structure)	Ao término da operação de comprometimento	Indica que o agente começou a desempenhar um papel ou está em busca de um objetivo.
block	block (BaseStructure, BlockType, Parameter)	Ao término da operação de bloqueamento	Indica que um mecanismo do agente foi bloqueado.
unblock	unblock (BaseStructure, BlockType)	Ao término da operação de desbloqueamento	Indica que um mecanismo do agente foi desbloqueado.
sendMessage	sendMessage (Sender, Recipients, MessageSchema, Parameters)	Ao término da operação de envio de mensagens	Indica que uma mensagem foi enviada.
replace	replace (BaseStructure, Structure, NewStructure)	Ao término da operação de substituição	Indica que uma estrutura da estrutura base foi substituída por outra estrutura.
goalAchieved	*commitTo (Agent, Goal)	Ao término do alcance do objetivo	Indica que o agente alcançou um objetivo.
contextUpdate	*`add (Agent, Belief)`	Ao término da operação de adição de contexto	Indica que houve uma atualização em alguma das informações contextuais do contexto corrente do agente.
adaptationDone	-	Ao término da execução de uma adaptação	Indica que foi realizada uma adaptação no agente.

* Indica que o evento está relacionado a apenas um dos possíveis usos da primitiva genérica.

Os *monitores de informações contextuais* são responsáveis por capturar informações contextuais de diferentes fontes de informação (tanto de hardware quanto de software). Como exemplos de informações contextuais citam-se: a temperatura de uma sala, a posição de uma pessoa, a variação do câmbio do dólar, entre outros. Há um monitor associado a cada fonte de informação contextual. As variações no contexto percebidas pelos monitores são sinalizadas ao gerente de contexto.

Os *gerentes de contexto* centralizam o recebimento das atualizações no contexto e as redirecionam para os *interpretadores de contexto*, que são responsáveis por interpretar os dados coletados a partir dos monitores (o que geralmente resulta em informações em um nível mais alto de abstração). Os gerentes de contexto também são responsáveis pela manutenção da ontologia de contexto da aplicação. Na ontologia, todas as informações contextuais disponibilizadas por determinado gerente de contexto são descritas. Uma aplicação composta por agentes adaptativos ao contexto deve ter ao menos um gerente de contexto, que pode estar vinculado a vários monitores e interpretadores. As informações interpretadas são utilizadas pelos gerentes de contexto para construir o contexto atual de um *agente adaptativo ao contexto*. É esse agente quem possui um contexto e é alvo de adaptações.

Todas as ações realizadas por/em um agente adaptativo ao contexto são registradas na forma de *eventos internos*. Observe, na Figura 3.4, que há um repositório de eventos internos na arquitetura do agente adaptativo. Como exemplos de eventos internos citam-se: a adição de uma nova crença, a remoção de um objetivo, a inicialização ou a finalização da execução de um plano, a atualização das pré-condições de uma ação, o comprometimento com o alcance de um objetivo, entre outros. Com os eventos internos, os agentes passam a ter consciência deles próprios (em termos de suas estruturas e comportamentos ativos). Em [Kle08] é dito que consciência é um pré-requisito para a adaptação automática de sistemas e, para se adaptar, um sistema precisa ter consciência de si próprio (autoconsciência) e consciência do ambiente em que está inserido.

Os *adaptadores* são os elementos responsáveis por observar o histórico de eventos do agente e, quando apropriado, realizar adaptações em sua arquitetura. Os adaptadores são constituídos, basicamente, por um ponto de adaptação e por uma variante. O ponto de adaptação é composto por uma série de eventos que, quando gerados, iniciam o processo de adaptação na arquitetura do agente. Já a variante contém todas as instruções para a realização da adaptação (ela é formada por uma seqüência de primitivas). Os agentes adaptativos ao contexto possuem um repositório de adaptadores.

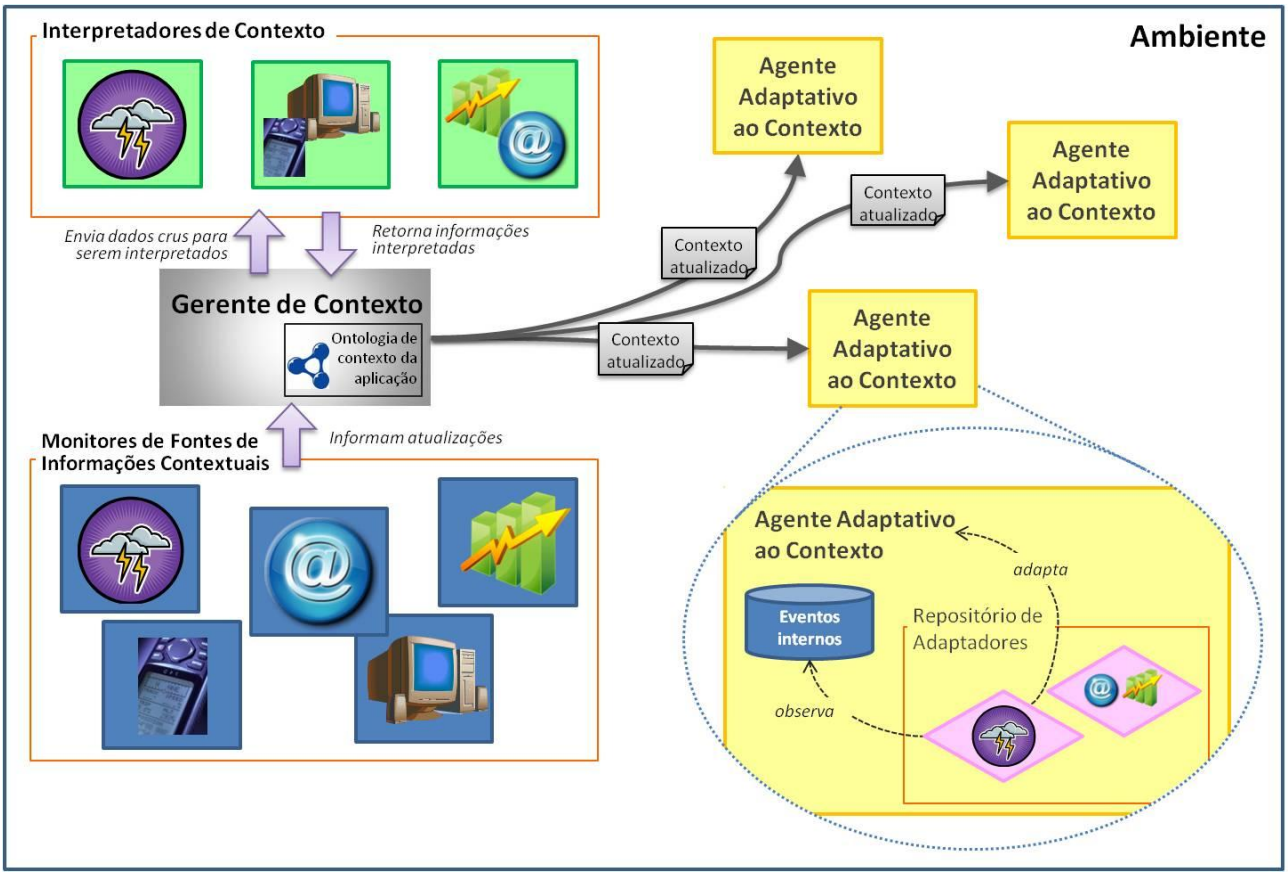

Figura 3.4 – Visão geral das principais estruturas da arquitetura **K2**.

O repositório de adaptadores de um agente pode ser expandido durante a sua execução de maneira manual ou automática (baseada em aprendizado multiagentes). A expansão de maneira manual requer a interação com usuários ou desenvolvedores para o cadastramento de novos adaptadores em tempo de execução. Já para expandir o repositório de adaptadores de maneira automática, foram definidos alguns mecanismos que permitem a busca e o compartilhamento de adaptadores entre os agentes.

O mecanismo de *aquisição de adaptadores* pode ser inicializado por dois motivos: (i) para substituir adaptadores obsoletos ou insuficientes (com base em avaliações de desempenho do agente); ou (ii) para buscar adaptadores para atingir um objetivo de forma mais satisfatória em determinado contexto. Na verdade, em ambos os casos o agente precisa procurar por novos adaptadores para melhor atingir os seus objetivos. A Figura 3.5 apresenta, de forma esquemática, alguns aspectos da execução do mecanismo citado. Como pode ser observado na figura, dentro do mecanismo de aquisição de adaptadores são executadas duas atividades. Inicialmente, as informações sobre a necessidade de adaptadores são encaminhadas à atividade “capturar”, que então as encapsula em uma mensagem para envio aos outros agentes adaptativos ao contexto do sistema. Depois de certo tempo, são verificadas as respostas recebidas para a solicitação, a fim de identificar os adaptadores compartilhados pelos outros agentes. Tal

verificação é feita na atividade “aplicar”. Depois de selecionado e aplicado um adaptador, esse é armazenado no repositório de adaptadores do agente.

Quando um agente adaptativo ao contexto envia uma solicitação de adaptadores, todos os agentes que recebem tal solicitação verificam seus repositórios em busca de adaptadores compatíveis com a necessidade descrita. Se existir algum adaptador compatível, é feito seu envio ao agente solicitante. A verificação dos adaptadores compatíveis e seu envio são tarefas realizadas pelo *mecanismo de distribuição de adaptadores*, cujo funcionamento é ilustrado na Figura 3.6. Conforme indicado na figura, sempre que é recebida uma mensagem de solicitação de adaptadores, é iniciada a execução da atividade “distribuir”. Se forem encontrados adaptadores compatíveis (como é o caso do cenário ilustrado), é enviada uma mensagem de “entrega” de adaptadores ao agente solicitante.

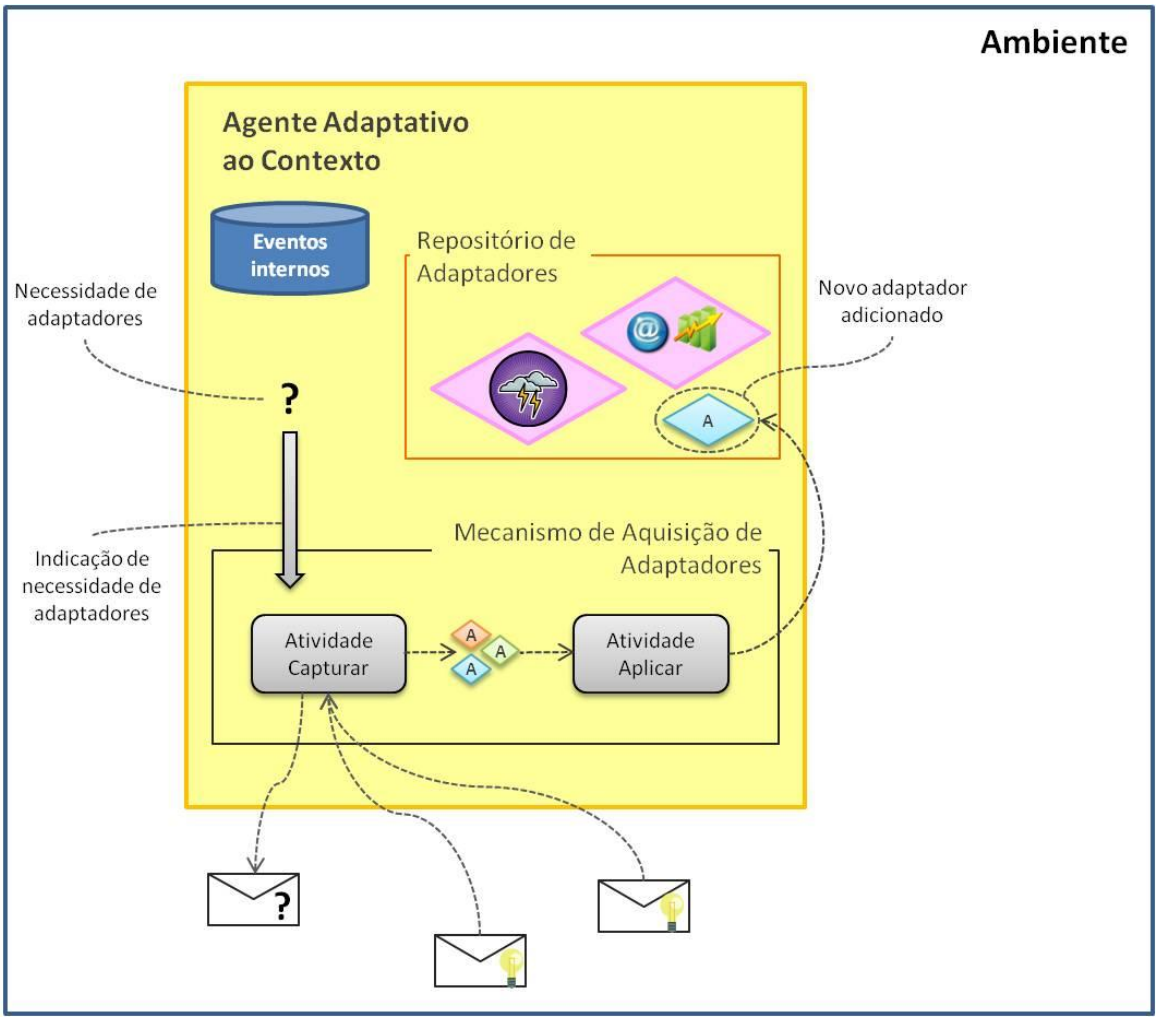

Figura 3.5 – Modelo esquemático do processo de aquisição de novos adaptadores.

Por fim, um agente adaptativo ao contexto é capaz de avaliar o seu próprio desempenho (avaliar os resultados atingidos depois do alcance de um objetivo). Assim, ele pode identificar quando os adaptadores disponíveis não estão contribuindo para o

alcance de bons resultados, adquirindo novos adaptadores para a sua substituição. Esta estratégia é baseada no comportamento humano: quando não estamos obtendo bons resultados trabalhando de uma determinada maneira, buscamos melhorar ou mudar nossa maneira de trabalhar a fim de alcançar melhores resultados.

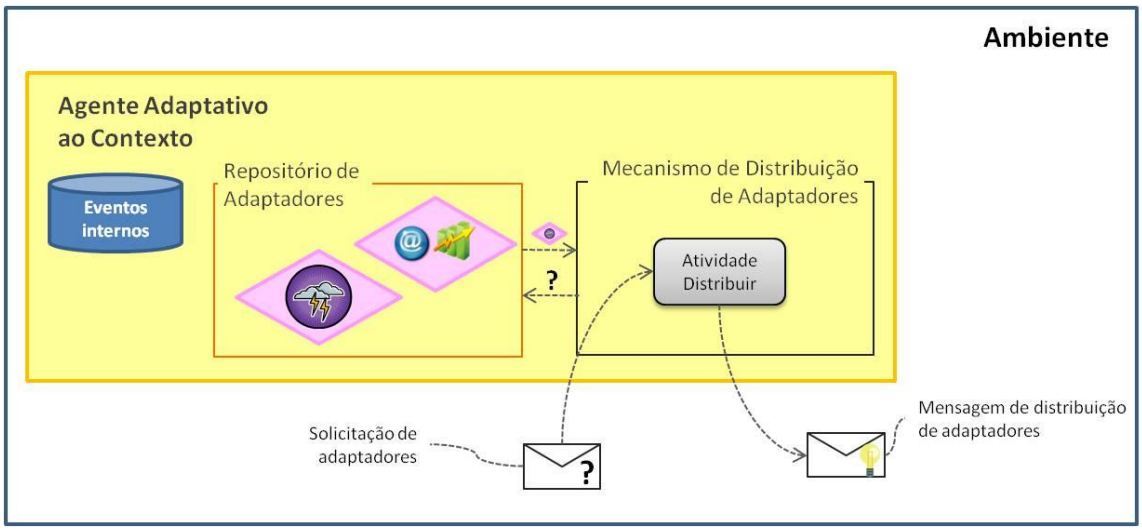

Figura 3.6 – Modelo esquemático do processo de distribuição de adaptadores.

3.4. Descrição do Modelo Conceitual da Arquitetura K2

A Figura 3.7 apresenta o modelo conceitual da arquitetura proposta. Observando-se a figura, podem ser destacados dois aspectos. Primeiro, a arquitetura desenvolvida deverá ser integrada a alguma plataforma para o desenvolvimento de agentes para que sejam criados agentes adaptativos ao contexto. De fato, a proposta é viabilizar a adaptação de diferentes tipos de agentes ao contexto (por isto a utilização de um metamodelo para especificar as adaptações possíveis e os locais na arquitetura de um agente onde podem haver adaptações) e não criar uma nova plataforma de agentes. Para que haja a integração entre a arquitetura e uma plataforma para o desenvolvimento de agentes, a classe `ContextAdaptiveAgent` deverá ser estendida, como será discutido em detalhes no Capítulo 4.

O segundo aspecto refere-se ao gerenciamento de contexto. Como pode ser visto na figura, vários conceitos apresentados não são relacionados à adaptação propriamente dita, mas sim ao gerenciamento de contexto. A inclusão de uma infra-estrutura para o gerenciamento de contexto foi necessária, porque a maioria das plataformas para o desenvolvimento de SMAs disponíveis não abrange tal infra-estrutura. Também, a infra-estrutura é externa aos agentes, pois vários autores sugerem o uso de mecanismos externos às aplicações para o gerenciamento de contexto (como [Hu08]).

3.4.1. Classes e relações do modelo conceitual

O pacote `contextManagement` contém todas as classes relacionadas ao gerenciamento das informações contextuais. A classe abstrata `InformationSourceMonitor` é responsável por capturar as informações contextuais de diferentes sensores, sejam eles sensores de hardware ou software (definição similar a "*context provider*" dada em [Ran03]).

Os atributos da classe `InformationSourceMonitor` são: `id`, que representa um identificador único; `informationID`, que indica o nome da informação contextual coletada como, por exemplo, "Temperatura" ou "Cotação do Dólar"; `sourceName`, que armazena o local onde a informação contextual está sendo adquirida (seguindo o exemplo anterior, poderiam ser nomes de fontes, respectivamente, "Sensor de Temperatura da Sala de Reuniões" e "http://br.advfn.com/p.php"; `sleepTime`, que representa o tempo de espera decorrido entre as leituras na fonte; `sensedInformation`, que armazena a última informação capturada; `sensedInformationTimestamp`, que indica o horário em que foi capturada a informação mais atualizada; e `log`, que é uma tabela utilizada para armazenar o histórico de atualizações da informação contextual (diferentemente da solução proposta em [Ran03], onde o histórico de contexto é armazenado por outro serviço do *middleware*). A utilização de um *timestamp* é justificada pela natureza temporal das entidades e recursos em ambientes pervasivos - usuários, dispositivos e serviços podem aparecer ou desaparecer repetidas vezes em um intervalo de tempo [Men06]. Também, segundo Ranganathan e Campbell, "o armazenamento de contextos passados possibilita o uso de algoritmos de mineração de dados para aprender e descobrir padrões no comportamento do usuário, nas atividades de uma sala ou em outros contextos" [Ran03].

Depois de capturados os dados brutos dos sensores, eles são processados pelos interpretadores de contexto (classe `ContextInterpreter`). A classe `ContextInterpreter` tem os seguintes atributos: `id` (identificador único do interpretador); `lastValue` (último valor interpretado); e `updateTime` (horário em que foi gerado o último valor interpretado). Há ainda uma associação entre as classes `ContextInterpreter` e `InformationSourceMonitor` para indicar a lista de monitores cujas informações coletadas são processadas por um interpretador. A idéia de utilizar interpretadores de contexto (ou sintetizadores [Ran03]) não é nova na literatura.

Dey e co-autores, em [Dey01], já indicam que retirando a interpretação do contexto das aplicações é possível reutilizar os interpretadores em múltiplas situações.

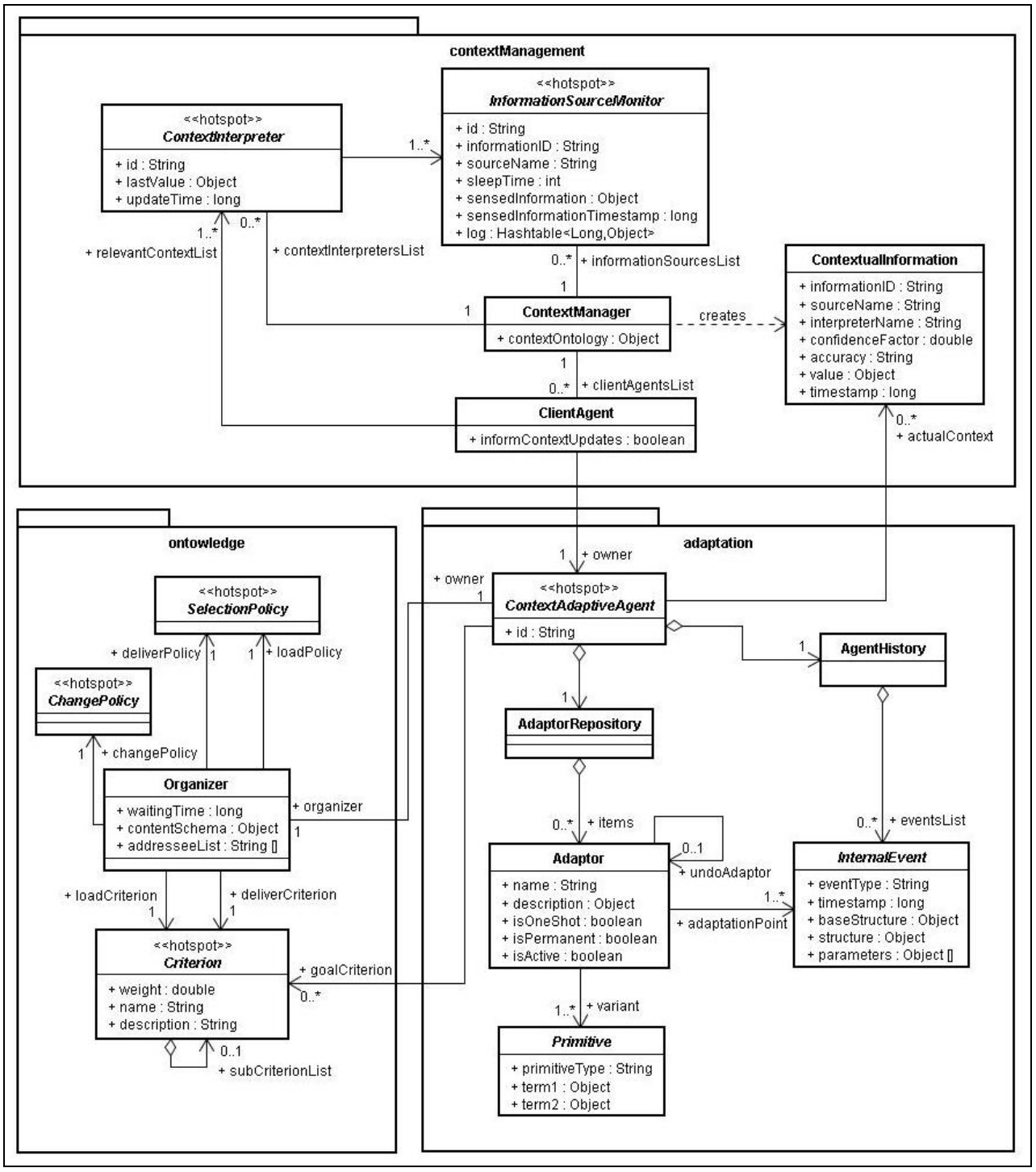

Figura 3.7 – Modelo conceitual da arquitetura **K2.**

De maneira geral, na literatura são citados três tipos de processamento ou interpretação de contexto[15]: (1) transformação das informações sentidas a partir de sensores no formato de representação utilizado no modelo de contexto; (2) dedução de

[15] Conclusão obtida durante a execução da revisão na literatura.

novas informações com base nas informações já conhecidas; e (3) filtragem das informações, sendo que somente devem ser mantidas as informações relevantes para a aplicação. Na arquitetura proposta neste trabalho são fornecidos meios para a realização dos três tipos de processamento citados. A realização do processamento do tipo 1 é nativa na arquitetura, visto que é utilizado o modelo de referência de contexto detalhado na Seção 3.1.3. Para a realização do processamento do tipo 2, são utilizados os interpretadores. Já para o processamento do tipo 3, os próprios agentes indicam o seu conjunto de informações contextuais de interesse e só são notificados de mudanças nessas informações (funcionalidade garantida pela classe `ContextManager`). A filtragem de informações contextuais relevantes, aspecto não muito citado nos estudos relacionados, é defendida por Mohomed e co-autores que indicam que “nem todas as informações contextuais são igualmente importantes para determinar os requisitos da adaptação” [Moh06].

É a classe `ContextManager` quem centraliza o recebimento das atualizações dos monitores de fontes de informações e as redireciona para os interpretadores. Assim, ela constitui um importante elo para a troca de informação na arquitetura proposta. Os gerentes de contexto (objetos da classe `ContextManager`) também são responsáveis pela manutenção da ontologia de contexto da aplicação. A ontologia de contexto da aplicação é uma extensão do modelo de referência apresentado na Seção 3.1.3. A manutenção de uma ontologia de contexto permite que seja criado um vocabulário comum para a troca de informação entre os agentes, reduzindo a ambigüidade na identificação e nos tipos de informações contextuais disponíveis. A ontologia deve ser atualizada sempre que um monitor de informação contextual é adicionado, um novo agente cliente é criado (ou um dos agentes antigos modifica seu conjunto de informações contextuais relevantes), entre outros.

As informações interpretadas são utilizadas para construir o contexto atual de um agente adaptativo ao contexto. Quando um agente adaptativo ao contexto (representado pela classe `ContextAdaptiveAgent`) se registra no `ContextManager` para receber informações de contexto relevantes, um novo objeto da classe `ClientAgent` é criado. Esse objeto é o responsável por armazenar os dados do agente que solicitou as informações contextuais (vide associação entre `ClientAgent` e `ContextAdaptiveAgent`) e por armazenar as preferências desse agente. As preferências de um agente cliente estão representadas no modelo pela associação entre as classes `ClientAgent` e `ContextInterpreter` (o que resulta em uma lista de

interpretadores “relevantes” – `relevantContextList` – na classe `ClientAgent`). O atributo `informContextUpdates` indica se o agente quer ser informado de mudanças ocorridas em seu contexto (valor `true`) ou se é ele quem vai solicitar as informações quando achar conveniente (valor `false`).

O pacote `adaptation` engloba as classes relacionadas ao agente de domínio e seus mecanismos e também as classes relacionadas à adaptação propriamente dita. A classe `ContextAdaptiveAgent`, como já mencionado, representa o agente de domínio, que é quem possui um contexto e sofre as adaptações. Ela é abstrata e deve ser estendida pelos agentes da plataforma ao qual a arquitetura estiver integrada. Para que possam ser realizadas adaptações em agentes de diferentes plataformas, a classe `ContextAdaptiveAgent` contém métodos que permitem executar todas as primitivas genéricas definidas. As partes constituintes de um agente, de acordo com o metamodelo FAML, não estão apresentadas na Figura 3.7, porque podem variar de acordo com a arquitetura interna dos agentes da plataforma ao qual a arquitetura **K2** estiver integrada.

O contexto atual de um agente possui uma lista de objetos da classe `ContextualInformation` (que devem ser transformados em instâncias da ontologia de contexto da aplicação no momento da transmissão). A classe `ContextualInformation` possui os seguintes atributos: `informationID` (identificador da informação contextual coletada); `sourceName` (nome da fonte de informação); `interpreterName` (nome do interpretador utilizado para atribuir o valor da informação); `confidenceFactor` (valor do fator de confiança da informação); `accuracy` (valor da precisão da informação); `value` (valor atual da informação contextual); e `timestamp` (horário em que o valor da informação foi atualizado pela última vez).

Todo agente adaptativo ao contexto contém um repositório de adaptadores (classe `AdaptorRepository`). O repositório de adaptadores, como o próprio nome sugere, nada mais é do que uma agregação de adaptadores (classe `Adaptor`). Os adaptadores são as estruturas que encapsulam o gatilho (eventos internos que, quando gerados, iniciam o processo de adaptação) e o efeito (conjunto de primitivas) de uma adaptação. Com a utilização de adaptadores, o comportamento adaptativo fica separado do comportamento padrão do agente. Assim, o código da adaptação não fica espalhado ao longo de toda a aplicação, o que facilita a inclusão de novos adaptadores em tempo de execução e também a manutenção dos adaptadores disponíveis.

Os eventos internos, representados pela classe `InternalEvent`, possuem como atributos: o tipo do evento (`eventType`), que pode ser qualquer um dos 17 tipos de eventos definidos na Seção 3.2.3; o momento em que o evento foi gerado (`timestamp`), a estrutura base que sofreu a alteração (`baseStructure`); a estrutura que foi adiciona/removida/alterada/executada na estrutura base (`structure`); e uma lista de parâmetros (`parameters`), visto que algumas operações contêm, além da estrutura, uma série de parâmetros. Os eventos internos são armazenados no histórico do agente (classe `AgentHistory`).

A expressão lógica criada com o conjunto de eventos internos utilizados para ativar um adaptador é chamada de *ponto de adaptação* (observe na Figura 3.7 que o papel da classe `InternalEvent` na associação entre `Adaptor` e `InternalEvent` é chamado de `adaptationPoint`). A escolha da nomenclatura “ponto de adaptação” foi motivada pelos pontos de atuação (*pointcuts*) da teoria de aspectos e já foi utilizada anteriormente em [Vay05] e [Eli08]. Em [Vay05], tal nomenclatura é utilizada na integração de um *middleware* em diferentes aplicações. Já em [Eli08], embora a nomenclatura seja utilizada para a adaptação de agentes, há divergência no significado (em [Eli08] os pontos de adaptação tem a função de representar as partes do agente que sofreram adaptação).

Para compor o ponto de adaptação, a estrutura base, a estrutura e os parâmetros de qualquer tipo de evento podem ser substituídos por elementos genéricos. Assim, é possível indicar que um adaptador deve ser inicializado quando, por exemplo, um plano qualquer estiver sendo executado (ou seja, um mesmo adaptador pode impactar em diferentes pontos ou momentos da execução do agente). É utilizado o caractere “?” para indicar que um elemento é genérico. Abaixo, há dois exemplos de utilização de elementos genéricos. Se tais eventos fossem utilizados para compor o ponto de adaptação de um adaptador qualquer, na situação (1) seria inicializado o processo de adaptação quando estivesse sendo executado qualquer plano para atingir o objetivo de nome “*SellBookGoal*” e na situação (2) bastaria estar sendo executado um plano no contexto de qualquer objetivo para inicializar a adaptação.

```
(1) executionStart (Goal:SellBookGoal, ?Plan)
(2) executionStart (?Goal, ?Plan)
```

O exemplo a seguir apresenta o ponto de adaptação de um adaptador chamado *SalesContextAdaptor*. Tal adaptador é inicializado quando o agente está executando o

plano de nome *SellBookPlan* no contexto do objetivo *SellBookGoal* (primeira cláusula da expressão) e quando o valor da informação contextual com identificador *Sales* é verdadeiro (segunda cláusula da expressão).

```
executionStart (Goal:SellBookGoal, Plan:SellBookPlan) &
contextUpdate (Interpreter:Sales, =, value: "true")
```

Além do operador booleano AND (representado pelo caractere "&" no exemplo acima), podem ser utilizados os operadores OR (caractere "|") e NOT (caractere "!") para criar a expressão do ponto de adaptação.

A classe `Primitive` foi inserida para representar as primitivas genéricas definidas na Seção 3.2.2. Na verdade, cada uma das primitivas genéricas definidas é uma especialização da classe `Primitive`. A classe `Primitive` tem como atributos: `primitiveType`, que representa o tipo da primitiva; e os termos `term1` e `term2`, que representam as estruturas e valores utilizados para a execução da primitiva.

O conjunto de primitivas utilizado para a realização de uma adaptação em um agente é chamado de *variante* (observe na Figura 3.7 que o papel da classe `Primitive` na associação entre `Adaptor` e `Primitive` é chamado de `variant`). No contexto deste trabalho, uma variante pode ser definida como as primitivas e/ou trechos de código que definem a adaptação a ser realizada quando um determinado ponto de adaptação é atingido. O conjunto de primitivas descrito pela variante é executado de forma seqüencial. A possibilidade de inclusão de trechos de código nas variantes é justificada pela possível necessidade de se realizar outras operações, como cálculos ou acesso a bancos de dados, além das já realizadas pelas primitivas definidas. A motivação para o termo "variante" surgiu a partir do trabalho apresentado em [Kha07], que usa o termo *application variant* no desenvolvimento de aplicações adaptativas baseadas em componentes (no trabalho citado, as variantes representam diferentes configurações de componentes, sendo que cada uma é mais apropriada para determinado contexto).

Considerando o adaptador de nome *SalesContextAdaptor* (cujo ponto de adaptação foi definido na página anterior), a seguir há um exemplo de variante. O exemplo indica que, sempre que a expressão lógica formada pelo ponto de adaptação do adaptador citado for verdadeira, será cancelada a execução do plano *SellBookPlan*, do objetivo *SellBookGoal*, e um novo plano - o *SellBookSalePlan* - será adicionado ao objetivo. Depois disto, é inicializada a execução do plano adicionado. Assim, o agente de

identificação *SellerAgent* muda sua estratégia de venda quando o contexto indica que é época de promoções.

```
abort (Goal: SellBookGoal, Plan: SellBookPlan)
add (Goal: SellBookGoal, Plan: SellBookSalePlan, {})
execute (Goal: SellBookGoal, Plan: SellBookSalePlan)
```

Além do ponto de adaptação e da variante, os adaptadores possuem uma série de atributos. São eles:

- `name`: compreende a identificação do adaptador e deve ser único;
- `description`: compreende uma descrição com os principais conceitos envolvidos no adaptador. Essa descrição é utilizada para selecionar adaptadores para compartilhamento com outros agentes.
- `isOneShot`: indica se o adaptador permanecerá ativo depois de sua primeira execução (valor `true`) ou será desativado (valor `false`).
- `isPermanent`: se verdadeiro, indica que as adaptações realizadas pelo adaptador serão permanentes. Se falso, indica que as adaptações serão provisórias, sendo que será necessário "desfazer" a adaptação em algum momento. Quando é necessário desfazer a adaptação de determinado adaptador, outro adaptador deve ser criado e associado a ele (veja auto-associação com papel `undoAdaptor` na classe `Adaptor`). A persistência ou não das adaptações só é pertinente quando se trata de adaptações estruturais, como adição de uma nova ação ou objetivo.
- `isActive`: indica se o adaptador está ativo e pode, portanto, ser aplicado na adaptação de um agente quando seu ponto de adaptação for verdadeiro, ou se ele não está ativo (o que significa que seu ponto de adaptação não está sendo sequer avaliado). A possibilidade de ativar ou desativar um adaptador é particularmente relevante quando há mais de um adaptador que pode ser aplicado na mesma situação.

A Tabela 3.6 sumariza as informações do adaptador de nome *SalesContextAdaptor*, que vem sendo citado como exemplo ao longo do texto. Na tabela, além do ponto de adaptação e da variante, pode-se ver o nome do adaptador, sua descrição e o valor de seus atributos `isOneShot`, `isPermanent` e `isActive`. Como já comentado, tal adaptador é executado quando o contexto indica período de promoções

(*Interpreter:Sales, =, Value: "true"*) e o agente está executando o plano "*SellBookPlan*" para atingir o objetivo *SellBookGoal* (*executionStart (Goal:SellBookGoal, Plan:SellBookPlan)*). Como resultado da adaptação, o agente deixa de executar o plano *SellBookPlan* (é utilizada a primitiva `abort` para indicar tal cancelamento) e começa a executar o plano *SellBookSalePlan* (o que é possível graças a primitiva `execute`). Observe que antes da execução do plano *SellBookSalePlan*, ele é adicionado a lista de planos aplicáveis do objetivo *SellBookGoal* (usando a primitiva `add`).

Tabela 3.6 – O adaptador *SalesContextAdaptor*.

Nome: SalesContextAdaptor	
One shot: ( ) Sim (X) Não	**Ativo**: (X) Sim ( ) Não
Adaptação permanente: (X) Sim ( ) Não **Ativar adaptador para anular:** -	
Descrição: Vendas, Promoção.	
Ponto de adaptação: executionStart (Goal:SellBookGoal, Plan:SellBookPlan) **&** contextUpdate (Interpreter:Sales, =, Value: "true")	
Variante: abort (Goal: SellBookGoal, Plan: SellBookPlan) add (Goal: SellBookGoal, Plan: SellBookSalePlan, {}) execute (Goal: SellBookGoal, Plan: SellBookSalePlan)	

O adaptador *SalesContextAdaptor* não é do tipo *one shot*, porque é necessário mudar a estratégia de venda sempre que o agente estiver executando o plano *SellBookPlan* no período de promoções. Quanto à permanência ou não da adaptação, é dito que o adaptador é permanente (ou seja, não é necessário indicar um adaptador para desfazer seu efeito), uma vez que a adaptação realizada é basicamente comportamental (a adição de um plano, embora seja uma adaptação estrutural, não afeta o comportamento futuro do agente quando não são indicados os critérios de seleção que possibilitam a escolha do plano para uso no alcance do objetivo).

Para que a arquitetura proposta apresentasse as funcionalidades de avaliação das adaptações realizadas e aprendizado multiagentes, foram incorporados a ela alguns conceitos e mecanismos do *framework* Ontowledge (desenvolvido anteriormente pelos autores deste trabalho) [Lem07a, Lem09]. O *framework* Ontowledge foi desenvolvido para dar suporte a agentes que não possuem conhecimento (ou não possuem o conhecimento adequado) para atingir determinado objetivo. Sempre que um agente com o Ontowledge integrado não possui o conhecimento necessário para atingir um objetivo, um objeto de conhecimento é recuperado e o seu conteúdo (o conhecimento encapsulado no objeto) é automaticamente adicionado a arquitetura do agente.

Na arquitetura **K2**, os mecanismos para compartilhamento de conhecimento entre agentes providos pelo Ontowledge foram customizados para permitir o compartilhamento de adaptadores ao invés de objetos de conhecimento. Também, como os agentes adaptativos ao contexto já possuem um repositório de adaptadores e há eventos para armazenar os resultados obtidos na execução de planos e no alcance de objetivos, não foram utilizados os conceitos `KnowledgeBase` e `ExecutionHistory` definidos no Ontowledge. A realização de modificações no Ontowledge já era prevista, uma vez que ele não foi projetado especificamente para o fim no qual está sendo utilizado.

Os conceitos vindos da arquitetura do Ontowledge, que estão brevemente descritos nos parágrafos a seguir, estão agrupados no pacote `ontowledge` no diagrama apresentado na Figura 3.7. O relacionamento entre a arquitetura proposta e os conceitos vindos do Ontowledge é evidenciado pela associação entre as classes `ContextAdaptiveAgent` e `Organizer`. É na classe `Organizer` que estão os principais métodos que permitem o compartilhamento e a aplicação de adaptadores. Cada objeto da classe `Organizer` possui os atributos:

- `owner` (proprietário): referencia o agente no qual o *framework* está integrado.
- `waitingTime` (tempo de espera): representa o tempo decorrido entre o envio de uma requisição de adaptadores e o início da análise das respostas recebidas para a solicitação.
- `contentSchema` (formato do conteúdo das mensagens): estrutura ontológica para o envio de solicitações de adaptadores e de respostas a solicitações. Esse formato permite que as solicitações de adaptadores e os próprios adaptadores sejam encapsulados, podendo ser enviados e interpretados pelos agentes do sistema. Cada mensagem contém um tipo ("*requestAdaptor*" ou "*deliverAdaptor*"), um identificador (toda mensagem de solicitação tem um identificador único) e o adaptador requisitado ou compartilhado.
- `addresseeList` (lista de destinatários): lista com os possíveis destinatários para as mensagens de requisição de adaptadores. Essa lista pode ser preenchida com os valores: "*all*", que indica o envio de mensagens para todos os agentes que habitam o mesmo ambiente do agente solicitante; ou com a identificação de qualquer outro agente (nome do agente). Caso a lista

esteja vazia, considera-se, como padrão, o envio para todos os agentes do ambiente ao qual o agente solicitante está inserido.

Com respeito ao atributo `contentSchema`, o *framework* Ontowledge objetiva tratar aplicações que trabalham com conceitos de *Web* Semântica e ontologias, então tanto o formato para representação do conteúdo de mensagens quanto o formato para a representação do conhecimento compartilhado remetem ao uso de ontologias. Segundo Weber e Kaplan (2003), o uso de ontologias pode auxiliar na resolução de dois desafios encontrados no desenvolvimento de sistemas baseados em conhecimento: o alto custo de aquisição de conhecimento e a falta de conhecimento de senso comum.

O uso de ontologias para representar as mensagens de requisição e compartilhamento de conhecimento não acrescenta qualquer dificuldade na utilização do Ontowledge neste trabalho, ao contrário, o seu uso tende a enriquecer a forma de encontrar o adaptador mais relevante para determinada necessidade. A descrição dos adaptadores (atributo `description`) é utilizada para verificar a compatibilidade entre uma solicitação e os adaptadores disponíveis no agente. Essa descrição, devido ao tipo do atributo (classe `Object`) pode assumir diferentes formas, podendo ser desde uma palavra ou sentença até uma ontologia.

Critérios de avaliação (classe `Criterion`) são utilizados para verificar a similaridade entre os adaptadores disponíveis e uma determinada solicitação/necessidade. Os critérios de avaliação são um ponto de flexibilidade da arquitetura proposta (no Ontowledge, os critérios também são pontos de flexibilidade). Cada critério, além de um nome, possui os atributos descrição e peso (que representa a importância do critério na avaliação do adaptador) e pode estar associado a uma lista de critérios (auto-associação `subCriterionList`), formando uma hierarquia de critérios.

A classe `Organizer` possui duas associações com a classe `Criterion`. O critério indexado em `deliverCriterion` é usado para avaliar a similaridade entre os adaptadores armazenados no repositório de adaptadores e uma solicitação de adaptador recebida. Já o critério contido em `loadCriterion` é usado para avaliar os adaptadores recebidos depois de finalizado o tempo de espera de uma determinada solicitação.

Os critérios utilizados para avaliar os adaptadores recebidos devem, além de verificar a relevância de cada adaptador em relação à necessidade que gerou a solicitação, verificar a aplicabilidade do adaptador no agente destino. Por avaliar a aplicabilidade, entende-se verificar se tanto o ponto de adaptação quanto as variantes são

aplicáveis ao agente de destino. Ou seja, é necessário verificar se o agente destino possui as estruturas para executar as primitivas contidas na variante e também se ele tem os elementos sobre os quais poderão ser gerados os eventos responsáveis por inicializar a execução do adaptador. Por exemplo, um adaptador que contenha o evento "`executionStart (Goal: SellBookGoal, Plan: SellBookPlan)`" no ponto de adaptação não é diretamente aplicável em um agente que não possua um objetivo com identificação "*SellBookGoal*" e um plano com identificação "*SellBookPlan*" (o adaptador pode até ser adicionado e ativado, mas ele nunca será executado). Fala-se que o adaptador não é "diretamente" aplicável, porque se pode implementar políticas que permitam ao agente solicitar as estruturas faltantes ao agente que compartilhou o adaptador, ou o adaptador ainda pode ser modificado/atualizado de forma semi-automática.

Para selecionar um ou mais adaptadores para distribuição ou aplicação devem ser implementadas políticas de seleção (`SelectionPolicy`). Novamente, a classe `Organizer` tem duas associações com a classe `SelectionPolicy`: `deliverPolicy` e `loadPolicy`. Em `deliverPolicy` é indexada a política de distribuição. Como exemplos de política de distribuição citam-se: somente o adaptador com maior similaridade será enviado; todos os adaptadores com alguma similaridade serão enviados, etc. Já em `loadPolicy` está indexada a política de seleção de adaptadores para aplicação (aplicar um adaptador significa ativá-lo e adicioná-lo ao repositório de adaptadores).

Para transmitir um adaptador de um agente para outro, é criada uma ontologia com os valores dos atributos do adaptador, assim o adaptador pode ser recriado no agente destino e adicionado ao seu repositório de adaptadores. A possibilidade dos agentes compartilharem conhecimento sobre adaptação (na forma de adaptadores) evidencia uma das características da arquitetura proposta, que é o aprendizado multiagentes.

Outra funcionalidade definida no *framework* Ontowledge é a verificação da satisfação alcançada com o uso de um objeto de conhecimento. No implementação original do *framework*, cada vez que um objeto de conhecimento é utilizado para atingir um objetivo, um registro de execução é criado para armazenar os resultados alcançados com o seu uso. O resultado da execução indica se o objetivo foi alcançado de maneira muito satisfatória, satisfatória, de maneira regular, insatisfatória ou não foi atingido. Os registros de execução permitem que o *framework* verifique o quão satisfatório está sendo

o uso de um objeto de conhecimento para atingir um objetivo e que, quando necessário, seja inicializado o processo de aquisição de novos conhecimentos.

Na arquitetura **K2**, o resultado obtido durante o alcance de um objetivo é armazenado em eventos internos do tipo `goalAchieved`, então não se faz necessário o uso de registros de execução (os mecanismos do *framework* Ontowledge foram modificados para permitir tal substituição). Para atribuir uma nota ao alcance de um objetivo, também são utilizados critérios. A arquitetura proposta se beneficiou da classe `Criterion`, já definida no *framework* Ontowledge, para avaliar o nível de satisfação obtido no alcance dos objetivos do agente (observe a associação entre as classes `ContextAdaptiveAgent` e `Criterion`). Cada agente pode ter nenhum ou vários critérios para avaliar o alcance de seus objetivos.

Os resultados obtidos no alcance de um objetivo, somados aos adaptadores executados enquanto o agente desejava alcançar tal objetivo, permitem deduzir, de forma empírica, o sucesso ou não do processo de adaptação. A possibilidade de avaliar as adaptações realizadas é um dos diferenciais da arquitetura proposta, visto que ainda são poucos os trabalhos que aplicam esforços em tal funcionalidade (no sentido de avaliar o nível de satisfação alcançado).

Além de criticar os resultados de seus processos de adaptação, os agentes adaptativos ao contexto desenvolvidos na arquitetura **K2** podem tentar melhorar seus desempenhos procurando novos adaptadores ou desativando/ativando os adaptadores disponíveis de forma automática. A busca por conhecimento novo sempre que um conhecimento se torna inapropriado ou obsoleto já era prevista na arquitetura original do Ontowledge.

A política de troca (`ChangePolicy`) é a responsável por avaliar o histórico de eventos do agente para verificar a necessidade de substituição de adaptadores. A política de troca está altamente relacionada ao tipo de dado utilizado para classificar a satisfação atingida no alcance dos objetivos. Se o resultado da execução for classificado em uma escala numérica (números de 1 a 5), por exemplo, pode-se implementar uma política que indique a necessidade de novos adaptadores quando um objetivo for alcançado 5 vezes com classificação menor do que 3 (a nota três, representa, neste caso, uma classificação pouco satisfatória). Cada vez que um objetivo é alcançado, deve ser verificada a política de troca.

3.5. Considerações sobre o capítulo

Esse capítulo apresentou os aspectos gerais do modelo de arquitetura para criação de agentes adaptativos ao contexto desenvolvido e aplicado nesta tese. Primeiramente, foi apresentado o modelo para a representação e categorização de informações contextuais proposto. Acredita-se que com o uso do modelo poderão ser desenvolvidas formas genéricas para o processamento de informações contextuais do mesmo tipo ou categoria. Além disso, a criação de um vocabulário comum, mesmo sendo em um alto nível de abstração, ajuda diferentes agentes a compartilhar o conhecimento de contexto disponível. O modelo de contexto proposto foi inicialmente relatado em [Lem07b] e posteriormente publicado em [Lem07c].

Depois disto, discorreu-se sobre os tipos de adaptações possíveis em arquiteturas de agentes. O metamodelo FAML foi utilizado para fazer o levantamento dos tipos de adaptações permitidos, sendo que foram identificadas 12 primitivas genéricas cujas aplicações contabilizam 39 diferentes usos. Então, é possível adaptar os agentes (comportamento e/ou estrutura) de 39 maneiras diferentes na arquitetura **K2**. Também, discorreu-se sobre os locais passíveis de adaptação na estrutura e execução dos agentes. Eventos internos, gerados a partir da execução das primitivas, foram utilizados para criar expressões lógicas que indicam o momento que deve ser iniciado o processo de adaptação.

Por fim, foi apresentado o modelo conceitual da arquitetura proposta. Durante a descrição do modelo conceitual, foram evidenciadas as principais características da arquitetura proposta. São elas: desenvolvimento de uma solução genérica; aprendizado individual e multiagentes; utilização de um modelo de contexto também genérico; adaptação de diferentes estruturas e do próprio comportamento dos agentes; e possibilidade de avaliação das adaptações realizadas. As características relacionadas ao aprendizado multiagentes e a avaliação das adaptações realizadas são garantidas devido à utilização de alguns dos mecanismos definidos no *framework* Ontowledge.

No próximo capítulo serão detalhados os aspectos referentes à implementação da arquitetura **K2**. Além da implementação da arquitetura proposta, será descrita a sua integração a uma plataforma simples para o desenvolvimento de agentes (cujas arquiteturas internas são baseadas no metamodelo FAML).

4. PROTÓTIPO E IMPLEMENTAÇÃO

4.1. Introdução

Neste capítulo, a arquitetura **K2** será detalhada em termos de sua implementação. Para o detalhamento, procurou-se seguir a mesma estrutura de apresentação do modelo conceitual (Capítulo 3), visando assim uma melhor identificação entre a parte conceitual e a implementação.

A implementação da arquitetura foi feita utilizando-se a linguagem de programação Java [Ora11], sua extensão orientada a aspectos AspectJ [Ecl11b] e o IDE Eclipse [Ecl11a]. Aspectos foram criados para a implantação dos adaptadores, o que permite que as variantes sejam introduzidas nas estruturas dos agentes em tempo de compilação, carga ou execução. Antes de se decidir pelo uso da programação orientada a aspectos na arquitetura proposta, foi necessário fazer um estudo de viabilidade técnica. Os resultados desse estudo foram inicialmente descritos em [Lem10a] e posteriormente publicados em [Lem10b].

Durante a implementação da arquitetura, procurou-se observar todos os aspectos com constantes modificações ou com implementações variáveis em diferentes plataformas para que eles pudessem ser implementados em estruturas flexíveis e expansíveis. Também, foram herdados alguns pontos de flexibilidade definidos no *framework* Ontowledge. Por exemplo, o Ontowledge trabalha com o conceito de critérios de verificação, sendo que esses podem ser utilizados tanto para avaliar a similaridade entre adaptadores quanto para avaliar a satisfação atingida após o alcance de um objetivo. As primitivas e eventos internos são representados por classes abstratas no modelo, mas não são considerados pontos de flexibilidade da arquitetura (suas implementações não são dependentes da aplicação nem da plataforma ao qual a arquitetura está integrada, como será visto no decorrer do capítulo).

Quanto ao escopo de implementação, neste primeiro momento foram implementados os aspectos referentes ao gerenciamento do contexto e à adaptação dos agentes utilizando adaptadores. As funcionalidades de aprendizado multiagentes e avaliação das adaptações realizadas, embora em andamento, ainda não foram finalizadas e testadas. No momento, é possível apenas fazer uma avaliação manual (humana) dos resultados obtidos pelo agente durante sua execução. Estas e outras limitações serão

melhor discutidas no Capítulo 6, que apresenta as considerações finais e os trabalhos futuros.

Por último, acredita-se que algumas reformulações na versão inicial da arquitetura serão necessárias para o aperfeiçoamento tanto do desempenho quanto de outras questões identificadas a partir da primeira versão. É improvável que se possa chegar a uma arquitetura ideal em flexibilidade e desempenho que combine várias tecnologias na sua primeira versão.

4.2. A arquitetura K2

A arquitetura **K2** apresenta uma série de funcionalidades que a distinguem das demais arquiteturas disponíveis para o desenvolvimento de aplicações adaptativas ao contexto. São características da arquitetura proposta: a utilização de um modelo de referência para a categorização das informações contextuais; a utilização de interpretadores de informações contextuais; o uso de adaptadores para separar o comportamento adaptativo do comportamento padrão do agente; e a possibilidade de adição de novos adaptadores durante a execução dos agentes. Como poderá ser observado no decorrer deste capítulo, a arquitetura **K2** satisfaz boa parte dos requisitos listados ao final do Capítulo 2.

Para um melhor entendimento das características da arquitetura desenvolvida, primeiro serão apresentados benefícios e limitações da programação orientada a aspectos no que tange ao desenvolvimento de aplicações adaptativas ao contexto (Seção 4.2.1). Durante a Seção 4.2.1 também são justificadas algumas decisões de projeto, visto que o conceito de adaptador está fortemente ligado ao conceito de aspecto. Além disso, são apresentados dois trabalhos encontrados na literatura que utilizam essa tecnologia para o desenvolvimento de aplicações adaptativas ao contexto. Depois, na Seção 4.2.2, são descritas as classes e relações da arquitetura desenvolvida e, na seqüência, fala-se sobre a integração da arquitetura em uma plataforma para o desenvolvimento de agentes (Seção 4.2.3).

4.2.1. Por que utilizar adaptadores e programação orientada a aspectos

Dey e colaboradores, em [Dey01], indicam que a separação de interesses, uma das bases da programação orientada a aspectos, é um dos requisitos que um *framework*

para lidar com contexto deve contemplar para auxiliar os desenvolvedores no gerenciamento de contexto. A separação de interesses mencionada pelos autores refere-se à aquisição e ao uso do contexto. De acordo com eles, através da separação da aquisição do contexto de seu uso, as aplicações podem usar informações contextuais sem se preocupar, por exemplo, com detalhes de implementação de um sensor, ou com as interfaces utilizadas para obter informações a partir dele. Carton *et al.* [Car07] citam que um dos requisitos para uma aplicação pervasiva é ser capaz de se adaptar dinamicamente com base em diversos fatores, incluindo o estado do ambiente, limitações do dispositivo, mobilidade do usuário e conectividade. Esses fatores podem impactar ao longo de toda a aplicação, por isto que se pode falar que eles são *transversais* [Car07].

Na visão de autores cujas pesquisas são mais voltadas para consciência de contexto em aplicações (sem se preocupar com o padrão arquitetural) parece ser claro quais são os interesses transversais. Já na literatura de agentes, ainda não há uma definição geral de quais são os interesses transversais que devem ser modelados como aspectos. Segundo Amor e Fuentes [Amo09], "(…) *identifying crosscutting concerns specific to agents is still an open issue* (…)". Para eles, as propriedades de comunicação tipicamente "cortam" diferentes módulos da arquitetura dos agentes e, portanto, englobam os interesses transversais utilizados pelos autores. Já Garcia e Lucena [Gar08] consideram como interesses transversais características de agência, como interação, adaptação, autonomia, colaboração, aprendizagem e mobilidade.

Amor e Fuentes [Amo09] indicam que o primeiro passo para a aplicação efetiva da tecnologia de desenvolvimento de software orientada a aspectos é a identificação dos interesses transversais na arquitetura interna de um agente individual. Neste trabalho, como se adapta os agentes em decorrência de mudanças em seus contextos e eventos ocorridos em suas arquiteturas internas, os interesses transversais são relacionados a ambos contexto e eventos ocorridos no agente. Na verdade, o processo de adaptação, por si só, é um interesse que corta muitos (senão todos) componentes do agente. Sendo assim, para evitar espalhamento e embaralhamento do código (*scattering* e *tangling*), é interessante separar o código relacionado à adaptação dos demais módulos ou componentes do agente, como é feito com o uso de adaptadores na arquitetura **K2**. A natureza "transversal" de um adaptador está na possibilidade de utilizar elementos genéricos na composição dos pontos de adaptação. Com os elementos genéricos, um adaptador pode cortar a aplicação em diferentes pontos.

Na literatura foram encontrados dois[16] trabalhos que utilizam programação orientada a aspectos para o desenvolvimento de aplicações conscientes de contexto. Em [Sin08], Sindico e co-autores apresentam a linguagem JCOOL (*COntext Oriented Language*), desenvolvida para lidar com consciência de contexto em aplicações Java. De acordo com os autores, a *detecção das mudanças* no contexto e as *adaptações relacionadas a essas mudanças* podem ser considerados dois interesses transversais distintos com respeito à lógica de negócio da aplicação.

A Figura 4.1 apresenta um trecho de código-fonte em linguagem JCOOL (extraído de [Sin08]), onde são interceptadas mensagens entre *peers* de acordo com o contexto. Como ilustrado na figura, a linguagem proposta define construtores para o monitoramento de contexto (construtor *Context*) e para a adaptação ao contexto (construtor *Adaptor*). No exemplo, o estado do contexto refere-se à segurança do protocolo de transporte disponível, sendo os valores possíveis “alto” ou “baixo” (conforme indicação *a* na figura). A adaptação compreende o cancelamento do envio de mensagens que requerem alto nível de privacidade quando a segurança do protocolo é baixa (indicação *b* na figura).

Em [Gra07] há aspectos executando o papel de “Monitores de contexto” e as modificações nas informações contextuais são verificadas por um contêiner chamado *Monitor*. Uma vez recuperadas, as informações contextuais são armazenadas pela própria aplicação. Há outro contêiner chamado *Adapter*, cujo papel é aplicar os mecanismos de adaptação de acordo com as indicações recebidas dos monitores de contexto. As políticas de adaptação são descritas em diagramas de atividade UML e no próprio código-fonte da linguagem AspectJ.

a

```
Context MediumReliability involve SmilePeer{

  default;
  low(instance):-instance.currentBinding.
                        equals(SipBinding);
  high(instance):-instance.currentBinding.
                   equals(XMPPBinding) &&
        ((XMPPBinding)instance.currentBinding).
        getTransport().
        equals(XMPPBinding.HTTPS));

}
```

b

```
Adaptor PrivacyAdaptor {

MediumReliability.low(instance) {
  in : {
      System.out.println("Warning unsecured
                          medium ");
  }
  out : { //No action }
  //layer definition
  Public void instance.send(Message msg){
   if((msg.getOverallPrivacyLevel()==HIGH){
      msg.getSender().printMsg("Message
      privacy level not compliant with the
      current medium");
   } else { proceed(); }
  }

}
```

Figura 4.1 – Exemplo do código-fonte da linguagem JCOOL [Sin08].

[16] O trabalho apresentado em [Amo09], como discutido na Seção 2.2.2, também utiliza aspectos para realizar a adaptação do sistema, no entanto, como os pontos de atuação são muito limitados (são interceptados apenas o envio e o recebimento de mensagens), ele não será considerado como exemplo nessa seção.

A Figura 4.2 mostra um exemplo de código em AspectJ (retirado de [Gra07]) de uma aplicação no domínio de serviços para usuários móveis em um aeroporto. As informações contextuais utilizadas no exemplo são a realização do *check-in* e o tempo faltante para o embarque (que deve ser menor que 20 minutos). O efeito obtido com a adaptação é a apresentação de alerta de tempo insuficiente para almoço quando ativada a função “Selecionar Restaurante” (o usuário pode optar por continuar ou abortar).

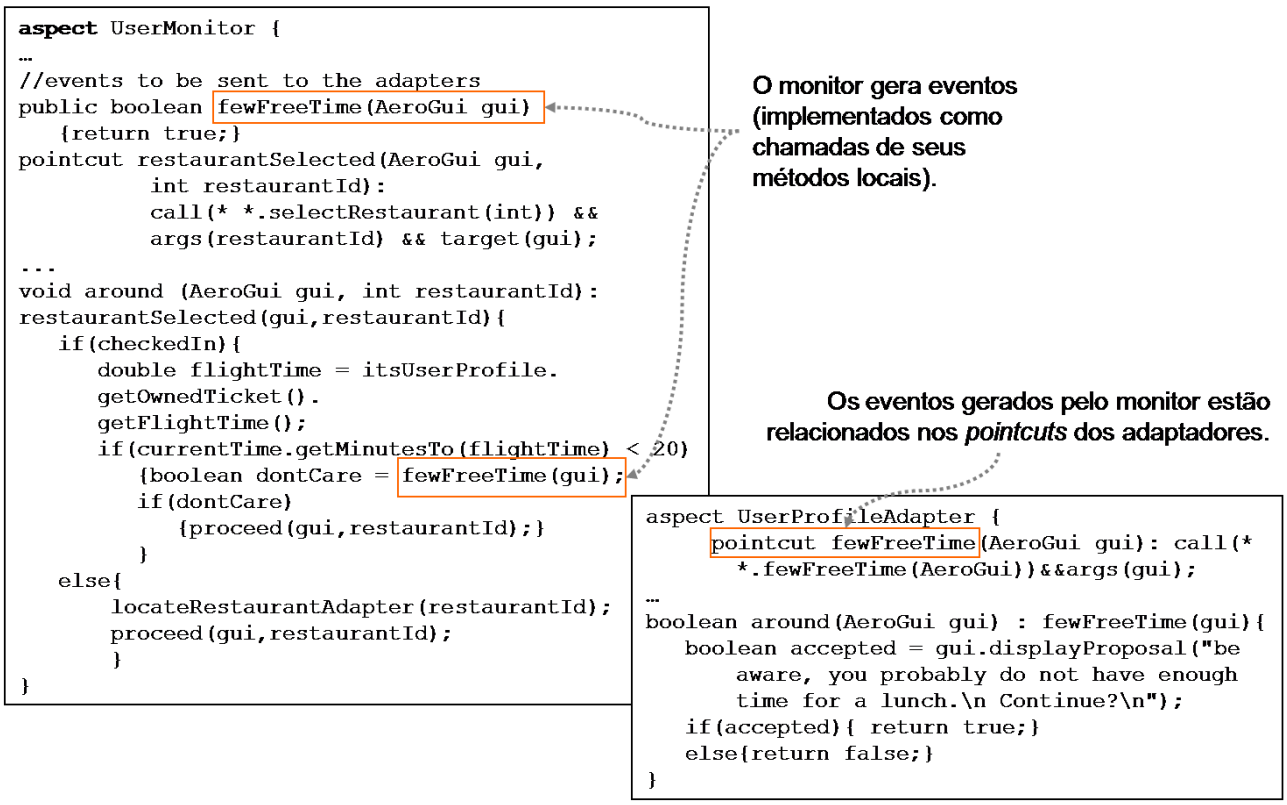

Figura 4.2 – Exemplo do código-fonte em AspectJ da solução proposta em [Gra07].

De maneira geral, a programação orientada a aspectos parece ser apropriada para o desenvolvimento de aplicações adaptativas ao contexto. Alguns dos benefícios obtidos com o uso da tecnologia são a modularização da solução e a separação dos interesses de adaptação ao contexto do restante do comportamento funcional da aplicação, simplificando seu desenvolvimento e manutenção. Embora a utilização de tal tecnologia traga benefícios para o desenvolvimento de aplicações adaptativas ao contexto, há algumas questões que ainda limitam o seu uso. No entanto, como será descrito a seguir, várias dessas limitações são superadas quando se trabalha com um nível de abstração maior do que o próprio aspecto, como é proposto neste trabalho.

De acordo com Grassi e Sindico [Gra07], abordagens orientadas a aspectos propõem soluções para lidar com consciência de contexto no nível de código-fonte, sem considerar o nível de projeto mais abstrato. Já em [Sin08] é criticado o restrito conjunto de

eventos no fluxo de execução de uma aplicação que são considerados como pontos de junção (locais onde há interceptação e possivelmente inserção de um comportamento adaptativo). De acordo com os autores, os pontos de junção normalmente utilizados na programação orientada a aspectos não são expressivos o suficiente para lidar com a complexidade de uma definição de contexto.

As duas limitações citadas pelos autores são superadas quando se usa adaptadores cujos pontos de adaptação e variantes são compostos, respectivamente, de um conjunto abrangente de eventos internos e primitivas. Primeiro, ao definir um adaptador utilizando eventos no ponto de adaptação e primitivas nas variantes, o desenvolvedor trabalha em um nível de abstração maior que o código-fonte, o que permite considerar o nível de projeto mais abstrato. A falta de expressividade dos pontos de junção também é superada com um conjunto abrangente e completo de eventos internos.

Amor e Fuentes [Amo09], com base no estudo de trabalhos disponíveis na literatura, indicam três limitações relacionadas à aplicação de soluções orientadas a aspectos em agentes. A primeira diz respeito à falta de interoperabilidade com plataformas compatíveis com o padrão FIPA, pois o padrão geralmente não é seguido nas soluções propostas na literatura. A segunda limitação refere-se ao tipo de *weaving* de aspectos utilizado. Segundo os autores, a composição dos objetos e aspectos é normalmente executada em tempo de compilação, o que reduz as possibilidades de adaptação dos agentes. A terceira limitação está relacionada ao reuso de aspectos, que é afetado devido à inclusão de nomes de classes, assinaturas de métodos, e outros, dos objetos que serão modificados. Por último, e não menos importante, a programação orientada a aspectos não é tão simples de aprender e, com a falta de ferramentas de suporte, o seu uso pode se tornar realmente complexo [Amo09].

Algumas das limitações indicadas por Amor e Fuentes são superadas devido à forma como os adaptadores são criados e combinados nos componentes do agente. Neste trabalho, os adaptadores são combinados nos componentes do agente em tempo de carga (*load time weaving*), porém, como os pontos de atuação de um aspecto criado para a implantação de um adaptador possui, basicamente, cláusulas com o designador *if*, todas as verificações para a execução de uma adaptação são feitas em tempo de execução. A dificuldade de utilizar programação orientada a aspectos é minimizada pelo fato do desenvolvedor definir os pontos de adaptação e variantes utilizando um conjunto de eventos e primitivas pré-definidos. A falta de interoperabilidade com o padrão FIPA não

é uma preocupação do trabalho proposto, visto que tal compatibilidade é de responsabilidade da plataforma na qual a arquitetura **K2** estiver integrada. Já a questão relacionada ao reuso de aspectos continua sendo uma limitação.

4.2.2. Classes e relações da arquitetura proposta

Para descrever os aspectos relacionados à implementação da arquitetura **K2**, na Figura 4.3 está ilustrado o seu diagrama de classes de projeto. Por questões referentes ao tamanho do diagrama, não estão representados os métodos concretos do tipo *get*, *set*, *add* e *remove* na figura.

Assim como no modelo conceitual, as classes do diagrama estão divididas em três pacotes, que são: `contextManagement`, `adaptation` e `ontowledge`. Naturalmente, o pacote `contextManagement` encapsula as classes responsáveis pela aquisição e distribuição das informações contextuais aos agentes adaptativos ao contexto. Para que os monitores de informações contextuais possam capturar as informações em diferentes fontes, na classe abstrata `InformationSourceMonitor` há o método abstrato `getUpdatedInformation`, cujo valor de retorno é um `Object`. Também, há o método abstrato `equal`, cuja função é verificar se houve atualização no contexto com base na última leitura realizada. Quando há variações, é feita uma sinalização ao gerente de contexto (objeto da classe `ContextManager`) associado ao monitor de informação contextual. Os monitores de informações contextuais possuem seus próprios processos de execução (são especializações da classe `Thread`).

Quando um gerente de contexto é informado que houve atualização em algum de seus monitores (através do método `contextUpdateOn`), ele identifica os interpretadores apropriados e solicita a eles que interpretem as atualizações recebidas. O método abstrato `getInterpretedInformation`, da classe `ContextInterpreter`, é o responsável por interpretar as informações adquiridas pelos monitores, gerando informação contextual em um nível mais alto de abstração. O método `getInterpretedInformation` é abstrato para permitir que as informações coletadas pelos monitores sejam processadas de diferentes formas.

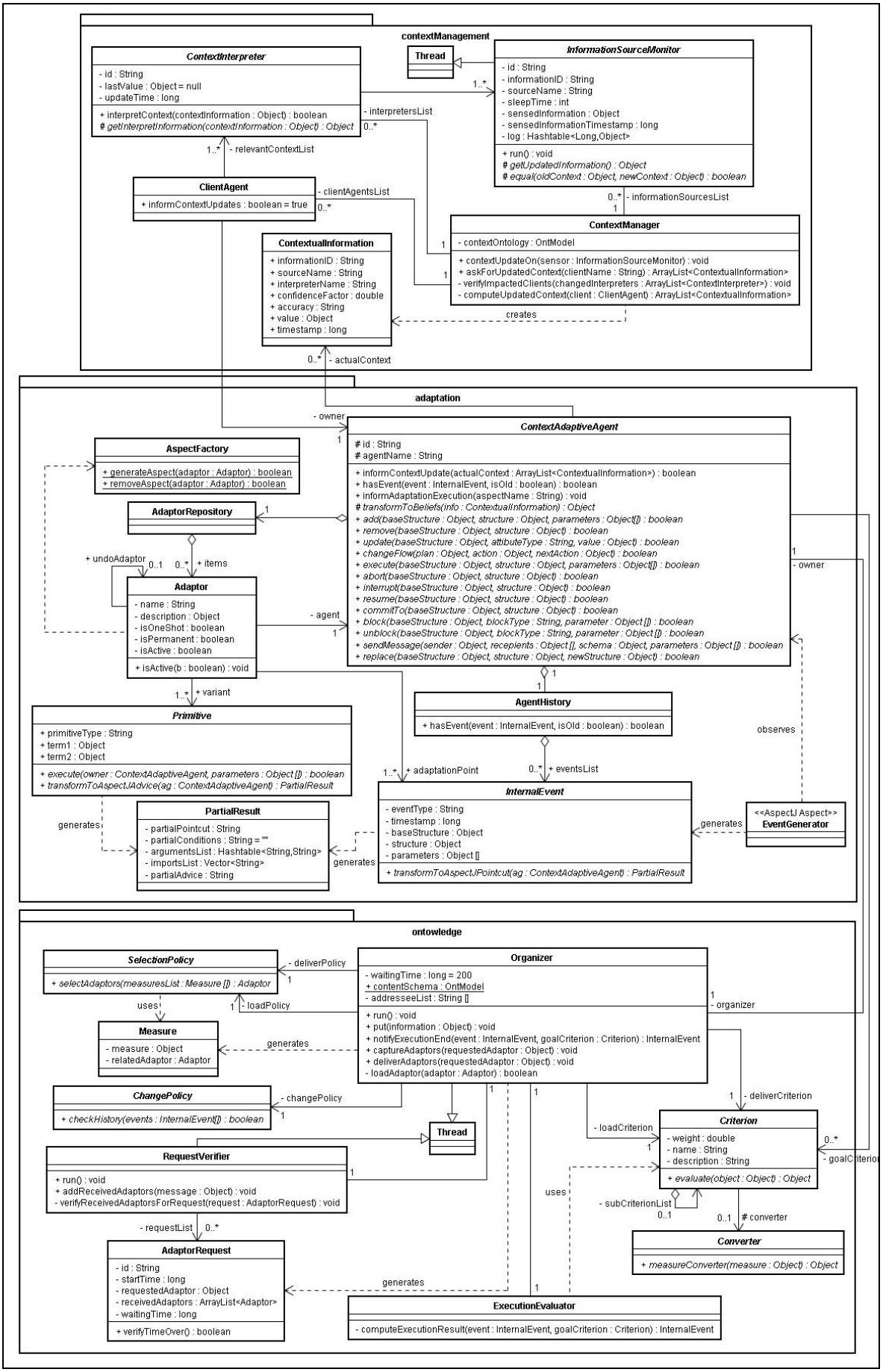

Figura 4.3 – Implementação da arquitetura **K2**.

Depois de receber as informações interpretadas, o gerente de contexto verifica quais os agentes clientes que tiveram seu contexto alterado (invocando o método `verifyImpactedClients`) e, caso se confirme a alteração do contexto de algum agente, o método `computeUpdatedContext` é invocado. Esse método é responsável por gerar o contexto atual de um agente e notificá-lo das atualizações. Para receber as notificações de atualização no contexto, um agente cliente precisa ter o valor `true` no atributo `informContextUpdates` (atributo da classe `ClientAgent`). Caso um agente não deseje receber as atualizações de contexto de forma automática, ele pode acessar seu contexto invocando o método `askForUpdatedContext` da classe `ContextManager`.

As notificações de atualização no contexto são informadas aos agentes adaptativos ao contexto (classe `ContextAdaptiveAgent`) através da invocação do método `informContextUpdate`, cujo parâmetro é uma lista de objetos da classe `ContextualInformation`. A classe `ContextAdaptiveAgent` é uma das classes principais da arquitetura proposta. É ela que deve ser estendida pelos agentes de domínio desenvolvidos em diferentes plataformas para que eles se tornem adaptativos ao contexto.

Inicialmente, as informações contextuais utilizadas para a adaptação são armazenadas e processadas na própria classe `ContextAdaptiveAgent`. No entanto, caso seja de interesse, essas informações podem ser transformadas em crenças (com a implementação do método abstrato `transformToBeliefs`). Dessa forma, os agentes que estenderem a classe `ContextAdaptiveAgent` podem usufruir das informações contextuais nos seus processos decisórios.

Além do método abstrato `transformToBeliefs`, a classe `ContextAdaptiveAgent` possui mais 13 métodos abstratos, sendo cada um responsável pela execução de uma das primitivas genéricas definidas (são 12 primitivas genéricas mais a primitiva facilitadora `replace`). Ou seja, o efeito da execução de uma primitiva se dá através da invocação de um desses 13 métodos. Assim, se for executado um adaptador que contenha a primitiva `add` na variante, será invocado o método `add` da classe `ContextAdaptiveAgent` para que a adaptação seja de fato realizada.

Os métodos que representam a execução das primitivas genéricas são considerados pontos de flexibilidade da arquitetura proposta, uma vez que suas implementações são dependentes da arquitetura interna dos agentes no qual a

arquitetura **K2** estiver integrada. Também, como há diferentes usos possíveis para cada primitiva, os parâmetros dos métodos são, em sua maioria, objetos da classe `Object`.

A classe `ContextAdaptiveAgent` ainda possui outros dois métodos concretos. O método `hasEvent` é invocado de tempos em tempos (a cada *tick* gerado pela arquitetura) para verificar se os eventos contidos nos pontos de adaptação dos adaptadores do agente já foram gerados (o que indica que deve ser iniciado o processo de adaptação) ou não. O método `hasEvent` possui dois parâmetros. O parâmetro do tipo `InternalEvent` representa o evento que se pretende verificar a criação enquanto o parâmetro do tipo `boolean` (`isOld`) está relacionado com o tempo de vida do evento.

Na arquitetura **K2**, um evento é considerado ativo por um pequeno período de tempo. Esse período de tempo é igual ao período do *tick* definido pela arquitetura. Como as avaliações das expressões dos pontos de adaptação são realizadas a cada *tick* gerado, se o evento fosse instantâneo, correr-se-ia o risco de uma adaptação não ser realizada devido à falta de sincronismo entre a geração dos eventos e a avaliação das expressões dos pontos de adaptação. Por outro lado, se o evento tivesse um tempo de vida muito grande, correr-se-ia o risco de iniciar o processo de adaptação muito tarde, sendo que as adaptações poderiam até já ser irrelevantes (devido ao estágio avançado de execução do agente no momento da realização da adaptação).

Depois de transcorrido o tempo de um *tick*, o evento já é considerado antigo (mas permanece registrado no histórico de eventos internos do agente). A finalidade do parâmetro `isOld` do método `hasEvent` é justamente indicar se o evento que se pretende verificar a existência está ativo (ou seja, se ele recém foi criado) ou se ele já está no histórico do agente há mais tempo. Utilizando o parâmetro `isOld` é possível indicar a seqüência[17] de acontecimento dos eventos. Por exemplo, considere um adaptador qualquer cujo ponto de adaptação seja constituído por dois eventos, o evento `eventA` e o evento `eventB`. O adaptador deve ser ativado assim que o evento `eventA` for gerado, mas desde que o `eventB` já tenha ocorrido no passado. O pseudocódigo a seguir mostra como o método `hasEvent` poderia ser utilizado para verificar a ocorrência dos eventos citados na ordem desejada.

```
se (agent.hasEvent (eventA, false) e agent.hasEvent (eventB, true))
      então [faça a adaptação]
```

[17] Na verdade, é possível indicar o evento que acarreta a execução da adaptação (o gatilho) e os eventos que já devem ter sido gerados no passado (ainda não é possível fazer uma seqüência cronológica de vários eventos do passado).

Como já comentado no Capítulo 3, os eventos do tipo `contextUpdate`, `executionStart` e `commitTo` podem ter um tempo de vida maior que um *tick*, uma vez que permanecem ativos até que, respectivamente, o contexto seja alterado, a execução de um plano ou ação seja finalizada ou o agente alcance um objetivo (ou deixe de desempenhar um papel).

Em relação ao uso de diferentes eventos internos em um mesmo ponto de adaptação, cabe salientar que alguns tipos de eventos, quando utilizados em conjunto, reduzem a praticamente zero a possibilidade de um adaptador ser executado. Embora um agente possa estar realizando várias atividades em paralelo, é difícil dois eventos internos estarem ativos ao mesmo tempo (tendo em vista o pequeno tempo de vida de um evento). Por exemplo, o ponto de adaptação "`add (Agent: Agent1, Goal: Goal1) & add (Agent: Agent1, Goal: Goal2)`" possivelmente nunca será verdadeiro, pois, quando o evento relacionado à adição do objetivo `Goal2` for gerado, o primeiro evento já não estará mais ativo. Uma das formas de resolver este problema é indicar que um dos eventos pode ser antigo (ou seja, que ele deve estar presente no histórico de eventos do agente, mas não precisa estar ativo). Obviamente, eventos do tipo `contextUpdate`, `executionStart` e `commitTo`, devido ao tempo de vida maior, não requerem este tipo de cuidado na hora da construção do ponto de adaptação.

Por fim, o método `informAdaptationExecution` (da classe `ContextAdaptiveAgent`) é utilizado para sinalizar que um adaptador foi executado e, portanto, adaptações comportamentais e/ou estruturais foram realizadas no agente. A invocação desse método acarreta a geração de um evento do tipo `adaptationDone`. O conhecimento das adaptações realizadas é importante por dois motivos: primeiro, existem os adaptadores do tipo `oneShot`, que devem ser desativados depois de sua primeira execução; segundo, para avaliar o impacto das adaptações no desempenho dos agentes, é preciso saber quais adaptações foram realizadas (e também quando elas foram realizadas).

A classe `Adaptor`, além dos métodos para acessar e modificar os valores dos atributos (que foram omitidos da figura), possui o método `isActive`, cuja função vai além de alterar o valor do atributo `isActive`. Quando um adaptador é ativado, seu ponto de adaptação começa a ser verificado para que, no momento oportuno, sua variante seja executada no contexto de um agente. O mecanismo desenvolvido para verificação dos pontos de adaptação e execução das variantes de um adaptador se beneficia dos

mecanismos já presentes na AspectJ para a introdução de aspectos aos componentes do sistema.

Cada vez que um adaptador é ativado, um novo aspecto (`Aspect` da AspectJ) é criado para que seja realizada a implantação do adaptador. Durante a criação do aspecto, o ponto de adaptação do adaptador é transformado em um ponto de atuação (*pointcut*) e sua variante é transformada em código-fonte da sugestão (*advice*). A transformação de um adaptador em um aspecto é feita de forma transparente tanto para o usuário quanto para o desenvolvedor. Uma vez criado o aspecto, a avaliação do ponto de adaptação e a execução da variante (quando validado o ponto de adaptação) passam a ser responsabilidade do *weaver* da AspectJ. Já quando um adaptador é desativado, o aspecto criado para implantá-lo é removido da biblioteca de aspectos do agente.

A criação e remoção de aspectos é realizada por métodos estáticos da classe `AspectFactory`. É o método `generateAspect` o responsável por percorrer todas as cláusulas da expressão do ponto de adaptação de um adaptador (cada cláusula é um evento interno) gerando as partes constituintes do ponto de atuação do aspecto que será criado. Na classe `InternalEvent` há o método `transformToAspectJPointcut`, que é acionado para fazer tal transformação. Já na classe `Primitive` há o método `transformToAspectJAdvice`, que é acionado para transformar cada primitiva da variante em código-fonte na sugestão. O retorno de ambos os métodos `transformToAspectJPointcut` e `transformToAspectJAdvice` é um objeto da classe `PartialResult`. Os objetos da classe `PartialResult` são combinados no método `generateAspect` (da classe `AspectFactory`) para criar tanto o ponto de atuação quanto a sugestão do novo aspecto.

Depois de criar o novo aspecto, é necessário introduzi-lo nos componentes e/ou estruturas do agente. Muito embora o ativamento do processo de adaptação seja baseado na avaliação dos eventos internos ocorridos na arquitetura do agente (o que descartaria a necessidade de *weaver* de aspectos e componentes), há a possibilidade de introduzir outros pontos de atuação, que não os eventos internos, nos pontos de adaptação de um adaptador. Para tanto, é necessário conhecer detalhes técnicos (como a assinatura dos métodos) da plataforma ao qual a arquitetura estiver integrada e, devido a tecnologia utilizada, a plataforma deve ser implementada na linguagem Java. Também, atualmente a arquitetura trabalha com *weaving* em tempo de carga (*load time weaving*), então só são avaliados os pontos de junção de classes ainda não carregadas pela máquina virtual Java.

A utilização de *weaving* em tempo de carga não é um problema para a solução proposta, porque os pontos de adaptação são baseados em eventos cuja verificação é feita em tempo de execução (é usado o designador `if` nos pontos de atuação). No entanto, com o uso de eventos, as adaptações são realizadas apenas depois de ocorrida determinada execução (ou seja, as declarações de sugestões são sempre do tipo `after)`. No futuro, pretende-se utilizar *weaving* em tempo de execução e possibilitar a criação de sugestões para serem aplicadas *antes*, *depois* ou *ao invés* de determinada operação do agente. A compilação do novo aspecto e sua composição nos componentes ou estruturas do agente também são instruções realizadas pelo método `generateAspect` da classe `AspectFactory`.

A classe `Primitive`, além do método `transformToAspectJAdvice`, possui o método `execute`, que também é abstrato. A implementação do método `execute` consiste na chamada de um dos métodos da classe `ContextAdaptiveAgent` (um dos 13 métodos abstratos que representam as primitivas genéricas), para que possa ser realizada, de fato, uma adaptação. Quando um aspecto é executado, é invocado o método `execute` de cada umas das primitivas contidas em sua sugestão.

Para a geração dos eventos internos, foi criado o aspecto `EventGenerator`. É esse aspecto o responsável por observar as operações realizadas na arquitetura interna de um agente e, de acordo com essas operações, gerar os eventos apropriados. O aspecto observa as operações realizadas tanto no escopo da classe `ContextAdaptiveAgent` quanto na classe do agente que a estiver estendendo. Assim, a implementação do aspecto `EventGenerator` também é um ponto de flexibilidade da arquitetura. A utilização da tecnologia de aspectos possibilita que a geração de eventos seja feita de forma não invasiva na plataforma que estiver integrada à arquitetura **K2**.

O pacote `ontowledge` encapsula as classes de projeto oriundas da arquitetura do *framework* Ontowledge. Como pode ser visto na Figura 4.3, a classe `Organizer` encapsula as funções necessárias para a busca, a distribuição e a aplicação de adaptadores. Essas funções são acionadas de acordo com as necessidades do agente adaptativo ao contexto ou com o recebimento de requisições de outros agentes. As necessidades de novos adaptadores são verificadas pelo *framework* após suas sinalizações pelo método `put`. Por exemplo, considere que um agente necessita se adaptar para atingir o objetivo de vender livros de forma mais satisfatória num contexto de crise mundial. Essa necessidade será indicada ao objeto da classe `Organizer` vinculado

ao agente adaptativo ao contexto (veja associação entre as classes `ContextAdaptiveAgent` e `Organizer`), que então executará o método `captureAdaptors` para a busca de adaptadores que tratem da venda de produtos em mercados em crise. Após o recebimento de adaptadores, é a vez de executar o método `loadAdaptor` para selecionar e aplicar o adaptador mais apropriado.

Cada vez que é recebida uma solicitação de novos, é criado um objeto do tipo `AdaptorRequest`. Cada `AdaptorRequest` possui quatro atributos, que são: `id`, que é o identificador gerado para a requisição (deve ser um identificador único); `startTime`, que informa o horário em que foi feita a solicitação (necessário para a verificação do tempo de espera); `requestedAdaptor`, que é uma descrição da situação que requer adaptação – essa descrição deve conter a descrição do objetivo que se quer atingir em determinado contexto; e `receivedAdaptors`, que é uma lista com todas as mensagens de distribuição de adaptadores recebidas para a solicitação.

As solicitações de adaptadores do agente são gerenciadas pelo processo `RequestVerifier` (especialização da classe `Thread`). Enquanto há solicitações de adaptadores pendentes na lista de solicitações (`requestList`) é verificado o tempo decorrido de cada solicitação (através do método `verifyTimeOver` da classe `AdaptorRequest`). Cada vez que é recebida uma mensagem cujo tipo é "*deliverAdaptor*", essa mensagem é encaminhada ao objeto `requestVerifier` através do método `addReceivedAdaptors`. No momento que termina o tempo de espera para uma solicitação, é executado o método `verifyReceivedAdaptorsForRequest`, de `RequestVerifier`. Nesse método são verificados todos os adaptadores recebidos para a solicitação (com o uso dos critérios de aplicação) e é selecionado um adaptador para carregamento, de acordo com a política de aplicação referenciada na classe `Organizer`.

Já as solicitações de adaptadores de outros agentes adaptativos ao contexto são processadas no método `deliverAdaptors` (da classe `Organizer`). Cada vez que é recebida uma solicitação externa, é feita uma verificação de similaridade entre a solicitação e todos os adaptadores disponíveis no repositório de adaptadores do agente. Como resultado dessa verificação, tem-se uma lista de objetos da classe `Measure` (que possui uma referência ao adaptador e outra a "nota" atribuída a ele). Essa lista é então encaminhada ao método `selectAdaptors` da política de seleção para distribuição (`deliverPolicy`) relacionada à classe `Organizer`. A implementação do método `selectAdaptors` é um ponto de flexibilidade da arquitetura.

Para avaliar o sucesso no alcance de um objetivo, foi criada a classe `ExecutionEvaluator` (diz-se “criar”, porque essa classe não estava na arquitetura original do Ontowledge). A classe foi criada para permitir que as avaliações de desempenho dos agentes fossem feitas com base nos eventos internos gerados e não nos objetos de conhecimento utilizados. Na arquitetura **K2**, cada vez que é gerado um evento do tipo `goalAchieved`, ele é enviado para um objeto da classe `Organizer` para que sejam avaliados os resultados obtidos pelo agente (é invocado o método `notifyExecutionEnd` da classe `Organizer`, que então invoca o método `computeExecutionResult` da classe `ExecutionEvaluator`). Quando é invocado o método `notifyExecutionEnd` é necessário informar, além do evento (que contém o objetivo que foi atingido), os critérios que devem ser utilizados para fazer a avaliação (objeto da classe `Criterion`). Considera-se que os objetivos dos agentes contenham as informações necessárias para fazer tal avaliação. O método `notifyExecutionEnd` retorna o evento interno já com os resultados da avaliação.

Para padronizar o valor computado pelos critérios, pode ser usado um objeto da classe `Converter`, que converte uma medida de entrada em algum outro valor (por exemplo, um valor 70, que indica uma porcentagem de 70% de satisfação, pode ser convertido, em uma escala lingüística, para o valor “bom”). Avaliar a satisfação atingida no alcance de um objetivo indica que o sistema deve monitorar o seu próprio desempenho e ser capaz de aprender com isso. A capacidade de monitorar seu desempenho, de acordo com Weber e Wu [Web04], pode garantir longos períodos de execução ao sistema sem a necessidade de manutenção.

Depois de computada a classificação alcançada pelo agente adaptativo ao contexto, é iniciado o processo de avaliação de seu histórico de execução. Nesse processo, todos os eventos internos são enviados para o método `checkHistory` da classe `ChangePolicy`, que também é um ponto de flexibilidade da arquitetura. O método `checkHistory` verifica a necessidade de busca de novos adaptadores, aprimorando de forma automática o mecanismo de adaptação.

4.2.3. Integração da arquitetura em uma plataforma para o desenvolvimento de agentes

Para a criação de agentes adaptativos ao contexto, a arquitetura **K2** precisa ser integrada a uma plataforma para o desenvolvimento de sistemas multiagentes. Na literatura, são disponibilizadas diferentes plataformas com esse intuito, como Jason

[Bor07], SemantiCore [Blo07] e Jade [Til11]. Cada uma dessas plataformas define sua própria arquitetura interna de agentes (ainda que com algumas características comuns).

Embora a arquitetura proposta pudesse ser integrada a qualquer uma das plataformas citadas, optou-se por desenvolver uma plataforma simplificada que funciona como um *driver* de teste [Per00], permitindo a criação de agentes adaptativos ao contexto sem o viés de uma ou outra plataforma. Com isso, espera-se que a aplicação da arquitetura de forma integrada com alguma plataforma ocorra sem maiores problemas no futuro.

A plataforma desenvolvida é, na verdade, uma versão simplificada e executável do metamodelo FAML. Como será visto a seguir, assim como no metamodelo FAML, na plataforma simplificada existem os conceitos *agente*, *objetivo*, *estado mental*, *plano*, *recurso*, *ação* e outros.

A Figura 4.4 apresenta o diagrama de classes em projeto da plataforma desenvolvida. A classe `SimpleAgent`, que é um processo autônomo (implementação da interface `Runnable`), representa os agentes desenvolvidos na plataforma simplificada. A classe `SimpleAgent` é uma especialização da classe `ContextAdaptiveAgent`, (definida pela arquitetura **K2**) e, portanto, implementa os métodos para a execução das 13 primitivas genéricas. Além dos métodos para a execução das primitivas, a classe `SimpleAgent` ainda tem outros dois métodos, que são o `receiveMessage` e o `decide`.

Uma das características de agência é a capacidade de interação dos agentes. Para interagirem, os agentes desenvolvidos na plataforma simplificada trocam mensagens (objetos da classe `Message`). Toda vez que é recebida uma mensagem, são verificados os sensores (classe `Sensor`) ativos no agente. São os sensores os responsáveis por filtrar as mensagens relevantes dentre as mensagens recebidas pelo agente através do ambiente.

As decisões do agente são tomadas no método abstrato `decide`. O processo de tomada de decisão tem como base as mensagens recebidas e o estado mental do agente no momento do recebimento de tais mensagens. O método `decide` é abstrato, porque cada agente tem uma forma diferente de processar as mensagens recebidas e utilizar as informações disponíveis em seu estado mental para tomar decisões.

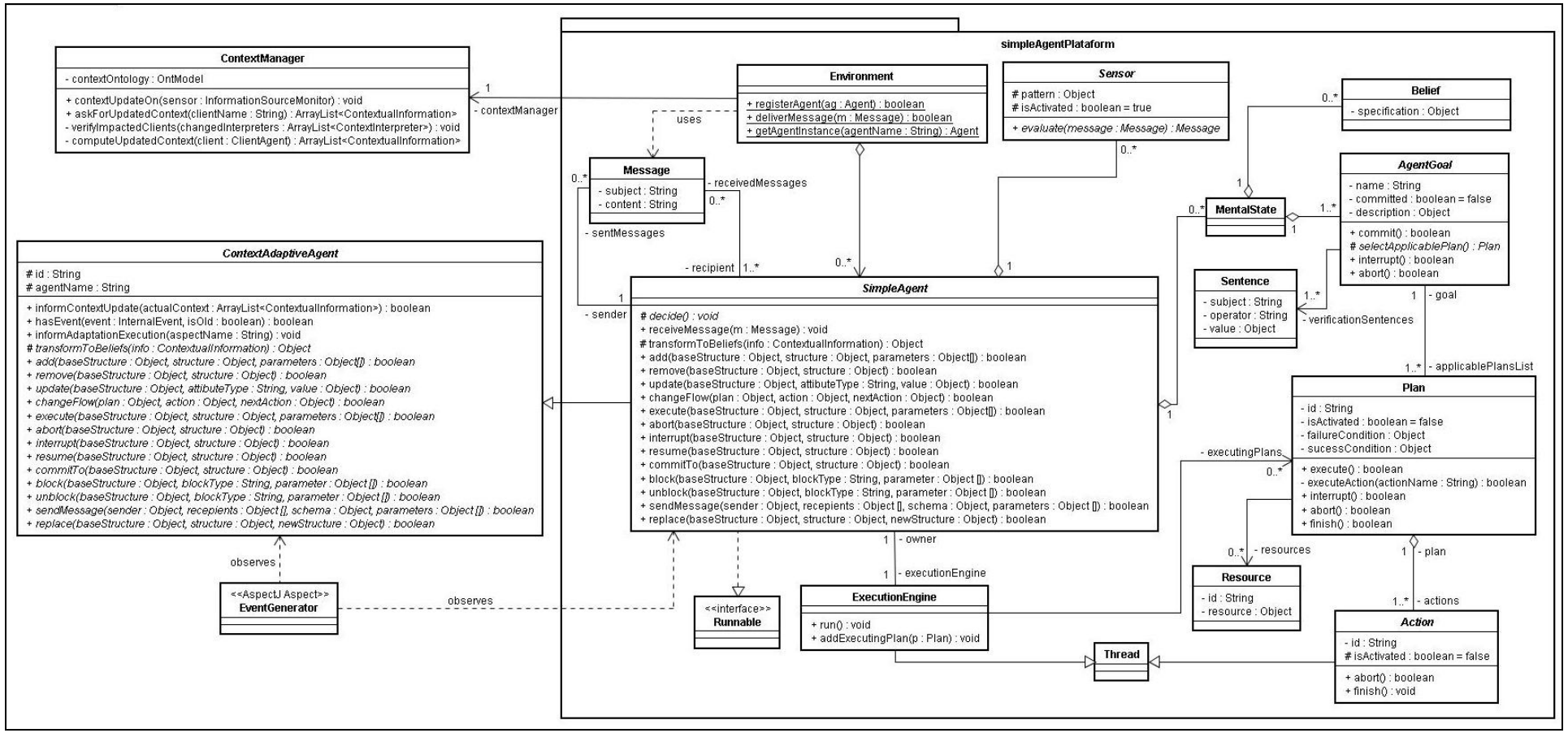

Figura 4.4 – Diagrama de classes de projeto da plataforma para o desenvolvimento de agentes criada.

O estado mental dos agentes desenvolvidos na plataforma (classe `MentalState`), assim como no metamodelo FAML, é o resultado da agregação de suas crenças e objetivos. Toda crença tem uma especificação, que é do tipo `Object`. Já os objetivos (classe `AgentGoal`) possuem três atributos, que são: `name`, que indica o nome ou identificação do objetivo; `committed`, cujo valor booleano indica se o agente está tentando alcançar o objetivo ou não; e `description`, que apresenta uma descrição dos conceitos envolvidos no objetivo - é essa descrição que é utilizada para a procura de adaptadores relevantes.

A classe `AgentGoal` ainda tem associações com outras duas classes, a classe `Sentence` e a classe `Plan`. As sentenças, ou sentenças de verificação, não são definidas no metamodelo FAML original, mas tem um importante objetivo na plataforma desenvolvida. É através das sentenças associadas aos objetivos que é feita a avaliação da satisfação alcançada pelo agente. Para fazer essa avaliação, são utilizados critérios (a Figura 4.3 mostra que a classe `ContextAdaptiveAgent` está associada a um ou mais objetos da classe `Criterion`). A classe `Criterion`, por ser um ponto de flexibilidade, permite que as sentenças sejam avaliadas e comparadas das mais diferentes formas. Cada sentença possui três atributos: (1) *subject*, que representa o nome de uma variável cujo valor deve ser capturado ao final do alcance do objetivo; (2) *operator*, que indica o tipo de operador utilizado (pode ser um operador relacional ou de igualdade); e (3) *value*, que é o valor desejável para a variável.

Cada objetivo possui uma série de planos aplicáveis (vide associação entre `AgentGoal` e `Plan`). Um plano possui uma identificação, um indicador de ativado ou não e condições de falha e de sucesso. Cada vez que um objetivo precisa ser alcançado (é invocado o método `commit` de `AgentGoal`), um dos planos associados a ele deve ser selecionado para execução. A seleção de um plano para execução é feita no método `selectApplicablePlan`, cuja implementação é um ponto de flexibilidade da plataforma proposta.

Objetivos e planos podem ser interrompidos ou cancelados antes do término de suas execuções. A diferença entre as duas operações, conforme descrito na Seção 3.2.2 está no armazenamento do estado de execução. De maneira geral, quando um objetivo ou plano é interrompido, é armazenado o estado da execução. Já quando é utilizado o método `abort`, a execução é completamente cancelada.

Os planos possuem ainda uma série de recursos (vide associação entre `Plan` e `Resource`) e são uma agregação de ações. As ações (objetos da classe `Action`) são abstratas e possuem seus próprios processos de execução (são especializações da classe `Thread`). Cada ação possui dois atributos: um identificador e um indicador de ativada ou não. Além disso, as ações possuem dois métodos, o `abort` e o `finish`. O método `abort` cancela a execução da ação (observe que as ações não possuem um método para interromper a execução, uma vez que não é possível interromper uma `thread` guardando seu estado de execução). O método `finish` é invocado ao final da execução da ação para que sejam coletadas informações para posterior avaliação do desempenho do agente.

Quando um objeto da classe `Plan` é selecionado para uso no alcance de um objetivo, é invocado o seu método `execute`. A implementação desse método compreende a execução, de forma seqüencial, de todas as ações contidas no plano, ou seja, a lista de ações é percorrida e as ações são inicializadas. Como as ações são processos distintos, uma vez inicializadas, elas seguem sua execução de forma autônoma. Então, para que se tivesse controle do término das execuções das ações (o que permite deduzir o término da execução do plano e, conseqüentemente, o alcance de um objetivo), foi criada a classe `ExecutionEngine`.

Cada vez que é iniciada a execução de um plano, é invocado o método `addExecutingPlan` da classe `ExecutionEngine`. Depois disto, no método `run` da classe, são feitas avaliações contínuas do estado das ações do plano (se em execução ou não). Quando todas as ações do plano são finalizadas, é invocado o método `finish` da classe `Plan`. Esse método é utilizado para indicar que a execução do plano foi finalizada (a execução do plano só é finalizada quando todas as suas ações foram executadas).

Por fim, há a classe `Environment`. A classe `Environment` representa o ambiente habitado pelos agentes. Cada vez que um agente é criado, ele deve ser registrado no ambiente através da invocação do método `registerAgent`. Há um e apenas um ambiente para cada sistema multiagentes criado na plataforma. Também, é a classe `Environment` a responsável por distribuir as mensagens aos agentes apropriados. O gerenciamento de contexto é oferecido como um serviço do ambiente. Observe que a classe `Environment` possui uma associação com a classe `ContextManager`. Quando o ambiente de um sistema multiagentes é criado, é criado

também um gerente de contexto para aquele ambiente. Através da associação entre o ambiente e o gerente de contexto, os agentes podem solicitar determinados tipos de informações contextuais e receber notificações de atualizações nessas informações.

4.3. Rumo ao Ambiente de Desenvolvimento K2 (K2 IDE)

Na literatura, vários autores destacam que, para as aplicações conscientes de contexto adaptativas ganharem impulso (sejam elas baseadas em agentes ou não), devem ser disponibilizados métodos de desenvolvimento e ferramentas apropriadas. Então, sabendo-se que de nada adianta desenvolver uma arquitetura robusta sem fornecer ferramentas que auxiliem no seu uso, nesta seção serão apresentadas algumas interfaces gráficas desenvolvidas para, em tempo de execução, visualizar as informações contidas no modelo mental dos agentes, incluir novas informações contextuais no contexto atual de um agente e, obviamente, gerenciar seus adaptadores. Em tempo de projeto, há métodos que permitem a criação e inclusão dos elementos citados. A criação de um *plugin* para o Eclipse já está em desenvolvimento no escopo de um trabalho de conclusão de curso.

Inicialmente, na Figura 4.5 é ilustrada a interface gráfica criada para visualizar as principais informações dos agentes adaptativos ao contexto (informações que variam no decorrer do tempo). As marcações na figura (retângulos tracejados) enfatizam as informações relacionadas à estrutura interna de um agente qualquer e as informações ou estruturas de um agente adaptativo ao contexto. Na parte superior da interface gráfica, há o nome do agente e a identificação do papel que ele está desempenhando no ambiente. Depois disto, há várias listas com elementos que, de acordo com o metamodelo FAML adaptado, são partes constituintes de um agente (ou são elementos manipulados por ele, como é o caso das mensagens). Esses elementos são os objetivos, os sensores, as crenças e as mensagens recebidas e enviadas pelos agentes. As informações contidas nessas quatro listas são apenas informativas, visto que a execução do comportamento padrão do agente é de responsabilidade da plataforma em que a arquitetura **K2** está integrada.

Na parte inferior da interface gráfica estão dispostas as estruturas manipuladas e/ou geradas pelos agentes adaptativos ao contexto criados com a arquitetura **K2.** Inicialmente, há uma lista com a identificação dos adaptadores armazenados no repositório de adaptadores do agente. Para criar um novo adaptador, basta clicar no

botão *Add New...* (ao lado da lista de adaptadores) que é aberta uma nova interface gráfica com campos para preenchimento.

A Figura 4.6 mostra a interface gráfica desenvolvida para a criação de adaptadores em tempo de execução. Na interface, é possível indicar os metadados do novo adaptador e também se pode construir, de forma incremental, tanto o seu ponto de adaptação quanto a sua variante. Os termos a serem utilizados nas cláusulas do ponto de adaptação são sugeridos de acordo com as estruturas correntes do agente. Por exemplo, no momento que é selecionado um evento do tipo `executionEnd`, já são listados para seleção todos os planos e ações do agente. Desta forma, o processo de criação dos pontos de adaptação fica menos propenso a erros.

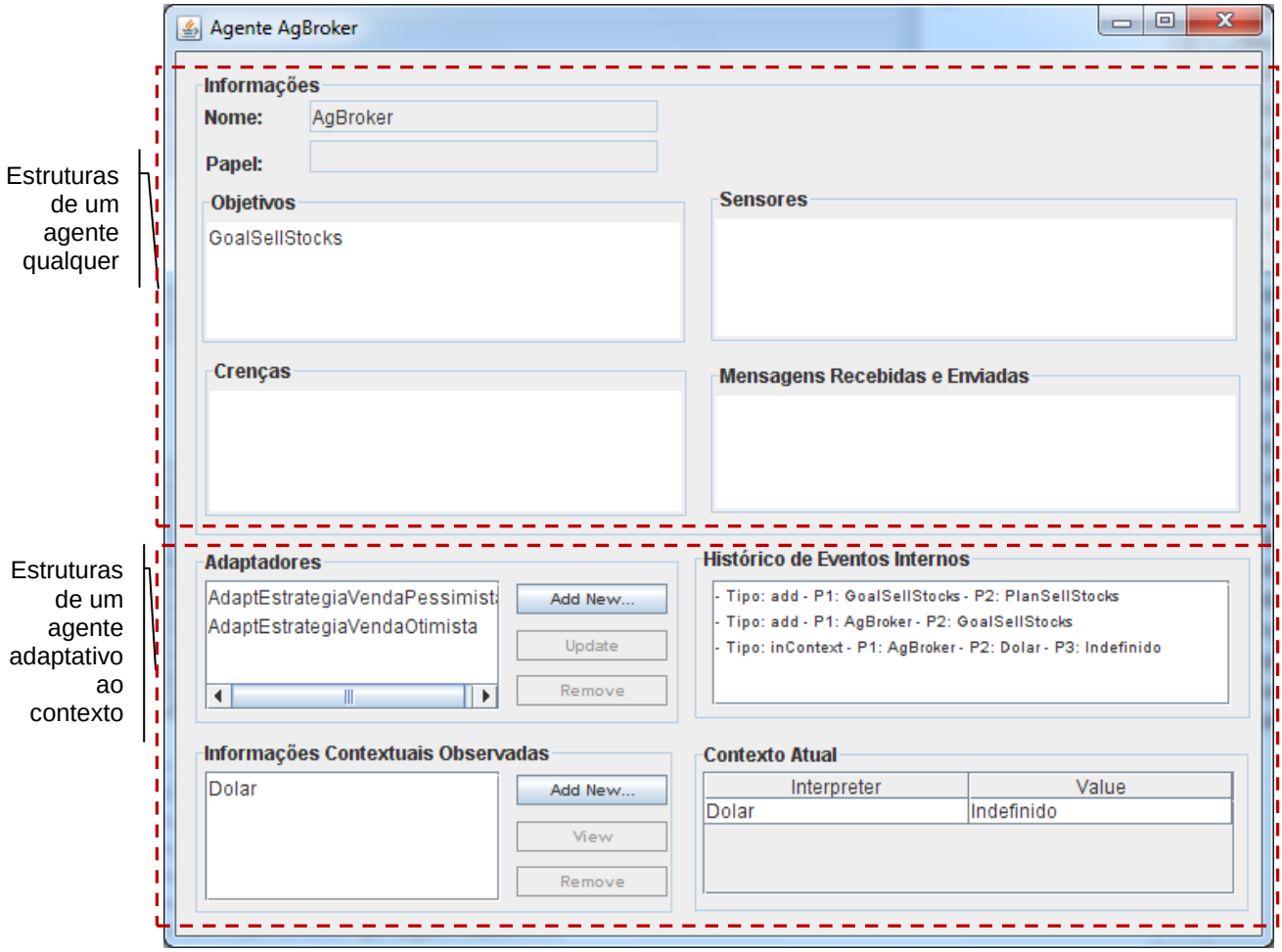

Figura 4.5 – Interface gráfica com as informações de um Agente Adaptativo ao Contexto.

Ainda na Figura 4.5, abaixo da lista de adaptadores está disposta outra lista – a lista de informações contextuais observadas pelo agente adaptativo ao contexto. Como já comentado, cada aplicação pode ter um conjunto diferente de informações contextuais relevantes, visto que a relevância de uma ou outra informação está relacionada com as necessidades da própria aplicação. No caso de aplicações baseadas em agentes, cada

agente pode ter o seu próprio conjunto de informações contextuais relevantes. São essas informações que estão indicadas na lista de informações relevantes da interface gráfica. O agente de identificação *AgBroker*, cujas informações estão listadas na Figura 4.5, tem em seu contexto apenas a informação contextual *Dollar*. Os valores da última atualização de contexto são apresentados ao lado da lista de informações contextuais observadas (há uma tabela com os valores do contexto atual).

Em tempo de execução, é possível modificar a lista de informações contextuais relevantes de um agente (é possível adicionar ou remover informações de seu contexto). A Figura 4.7 mostra a interface gráfica criada para o gerenciamento das informações contextuais de um agente. Nessa interface, são listadas todas as informações contextuais disponíveis no ambiente (resultado de uma consulta ao gerente de contexto), sendo que se pode selecionar ou remover a seleção de determinada informação contextual. Quando se seleciona uma fonte de informação contextual é preciso selecionar, também, qual interpretador que se deseja utilizar.

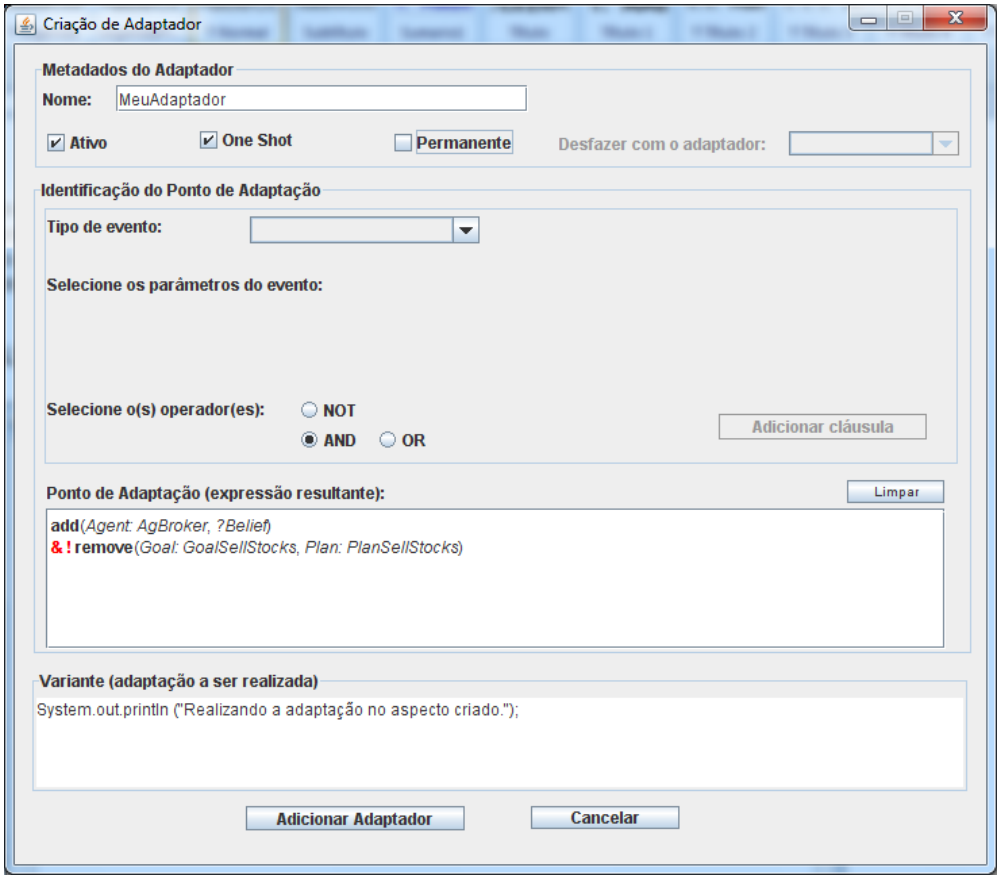

Figura 4.6 – Interface gráfica para a criação de um adaptador em tempo de execução.

Ainda na Figura 4.5, acima da tabela com o contexto atual do agente, há uma lista com o histórico de eventos internos. Essa lista também é apenas informativa, uma vez que os eventos são gerados de forma automática durante o ciclo de vida do agente.

Para finalizar, a Figura 4.8 mostra a estrutura geral de um ambiente multiagentes com gerenciamento de contexto (resultado da integração da arquitetura **K2** na plataforma simplificada para o desenvolvimento de SMAs desenvolvida). Como se pode ver na figura, dentro do ambiente há o serviço de gerenciamento de contexto que, além do próprio gerente de contexto, possui uma série de monitores de informações contextuais e interpretadores. O ambiente também é habitado por uma série de agentes de domínio adaptativos ao contexto.

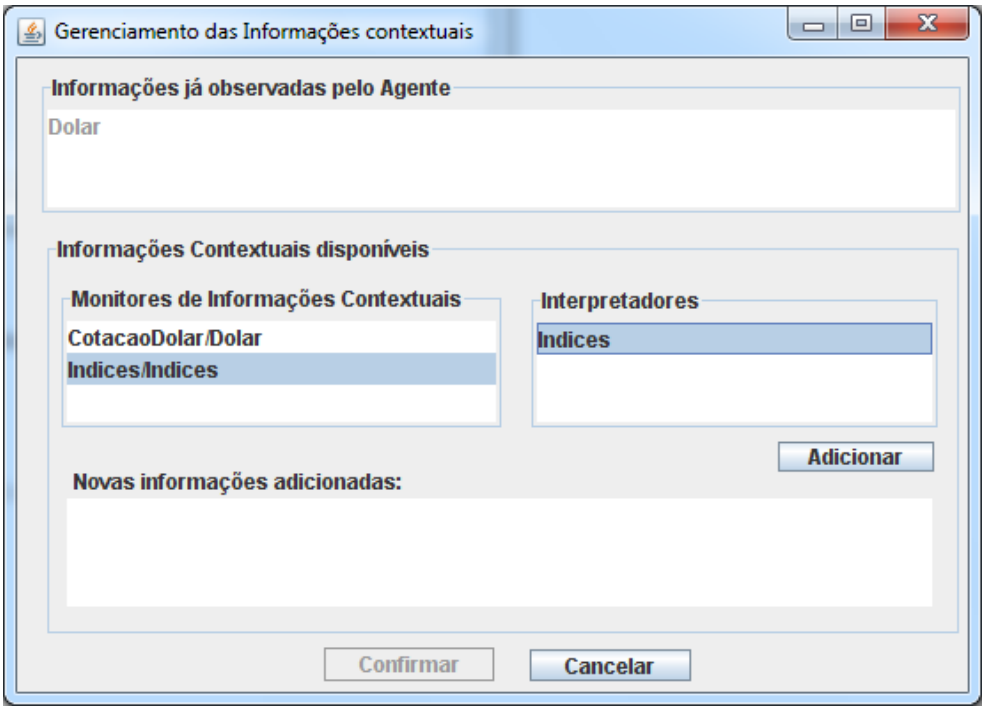

Figura 4.7– Interface gráfica para a inclusão de novas informações contextuais no contexto de um agente.

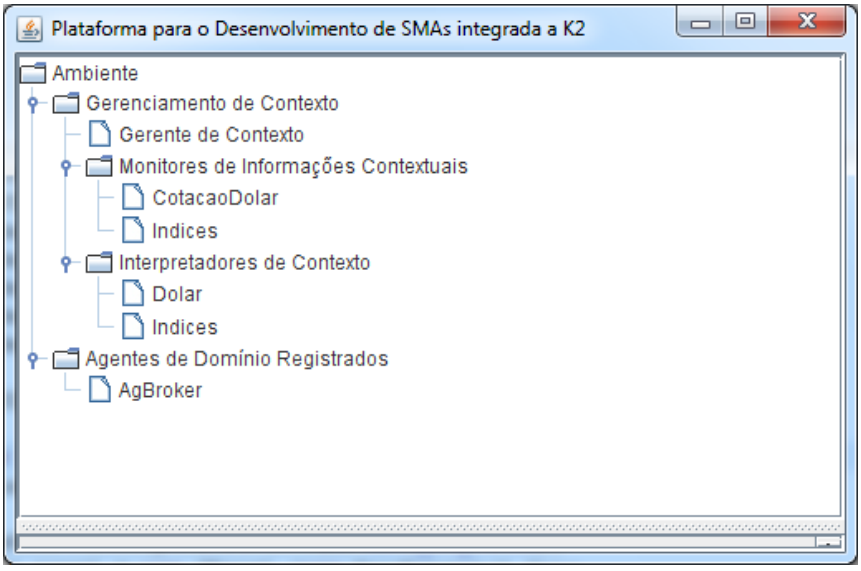

Figura 4.8 – Interface gráfica com a estrutura geral de um ambiente multiagentes com gerenciamento de contexto.

4.4. Guia de uso da Arquitetura K2

Depois de finalizada a apresentação dos aspectos técnicos da arquitetura **K2**, é preciso fornecer um guia com a seqüência de passos que devem ser executados para a sua correta e completa utilização no desenvolvimento de agentes adaptativos ao contexto. Basicamente, depois de se integrar a arquitetura **K2** na plataforma onde se pretende desenvolver os agentes de software, devem ser executados 5 passos, que são: (i) análise do problema; (ii) modelagem e implementação dos agentes; (iii) identificação das informações contextuais relevantes; (iv) implementação dos monitores de informações contextuais e interpretadores; e (v) identificação e criação dos adaptadores.

Os dois primeiros passos são normalmente executados quando se pretende desenvolver uma aplicação baseada em agentes. De fato, primeiro é preciso analisar o problema, decompondo sua solução em uma série de agentes autônomos. Depois, é preciso especificar quais serão as estruturas desses agentes em termos de objetivos, planos, crenças e outros para só então partir para a implementação. A modelagem dos agentes pode ser feita utilizando-se uma linguagem para tal fim, como a MAS-ML [Sil08]. Porém, cabe salientar que a linguagem de modelagem irá auxiliar apenas na análise e projeto das estruturas constituintes de um agente qualquer - a identificação e projeto dos adaptadores serão feitos no quinto passo, conforme será explicado no decorrer do texto.

Depois que o problema for analisado e os agentes constituintes da aplicação já estiverem definidos e implementados, devem ser listadas as informações contextuais relevantes (passo 3). Com base na lista de informações contextuais relevantes é que é executado o quarto passo. Nesse passo deve ser implementado um monitor de informação contextual para cada informação contextual contida na lista. Além disso, deve-se implementar os interpretadores de contexto.

A identificação e criação dos adaptadores é feita no último passo (passo 5). Como os adaptadores interceptam eventos relacionados à estrutura dos agentes e às informações contextuais disponíveis no ambiente, eles só devem ser criados após a criação desses elementos. Durante a execução do quinto passo, devem ser identificados os eventos que servirão de gatilho para os adaptadores e devem ser definidas as adaptações a serem realizadas quando os gatilhos forem ativados. Também, é possível que outras estruturas constituintes dos agentes precisem ser implementadas durante a criação dos adaptadores. Por exemplo, determinado adaptador pode agir sobre um plano de ação, adicionando uma nova ação a ele quando determinado evento for gerado. Nesse caso, além do adaptador, deverá ser implementado o código da ação que será adicionada

ao plano de ação. Os passos 4 e 5 podem ser executados uma ou mais vezes tanto em tempo de projeto quanto em tempo de execução.

Depois de executados os 5 passos, a aplicação composta de agentes adaptativos ao contexto está pronta para a execução.

4.5. Considerações sobre o capítulo

Esse capítulo destacou aspectos referentes à implementação da arquitetura proposta, descrevendo os padrões e as ferramentas utilizadas no seu desenvolvimento bem como o ambiente de implementação e teste. Ao longo do capítulo, foram apresentadas as principais classes do protótipo, com suas características e funcionalidades.

Depois de apresentada a implementação da arquitetura, é possível discorrer sobre sua adequação em relação aos requisitos listados ao final no Capítulo 2. No referido capítulo, foram listados 13 requisitos divididos em dois grupos: os relacionados ao gerenciamento de contexto e os relacionados à adaptação ao contexto. A Tabela 4.1 sumariza os 13 requisitos e a forma como eles estão (ou não) presentes na arquitetura **K2**.

Como pode ser visto na tabela, em relação ao gerenciamento de contexto, a arquitetura desenvolvida contempla, de forma integral, os requisitos 1, 2, 3 e 5. Já os requisitos 4, 6 e 7 são contemplados de forma parcial, conforme justificativas nos parágrafos a seguir.

A arquitetura **K2**, em seu módulo de gerenciamento de contexto, permite a utilização de diferentes fontes de informações contextuais, sendo que os agentes podem se registrar no gerente de contexto para receber notificações automáticas de atualizações nessas informações (em conformidade com o requisito 1). Também, foram desenvolvidas interfaces gráficas que permitem adicionar, em tempo de execução, novos monitores de fontes de informações contextuais ao gerente de contexto (em conformidade com o requisito 2). O requisito 3 refere-se ao uso de um metamodelo de contexto, o que também é contemplado pela arquitetura. Já a definição e criação dos interpretadores garante a conformidade da arquitetura com o requisito 5 (que se refere ao suporte para o processamento das informações contextuais, gerando informações em um nível de abstração maior que o provido pelos monitores).

Tabela 4.1 – Sumário dos requisitos atendidos pela arquitetura **K2**.

	Id	Requisito	A arquitetura K2 contempla
Relacionados ao contexto	1	Apoiar a coleta de informações contextuais de diferentes sensores e a entrega dessas informações para os agentes.	Sim
	2	Permitir a implantação e o uso de novos sensores durante a execução do sistema.	Sim
	3	Utilizar um metamodelo de contexto.	Sim
	4	Utilizar uma forma de representação para informações contextuais que seja robusta e de rápida atualização.	Parcialmente
	5	Dar suporte ao processamento das informações contextuais.	Sim
	6	Permitir interoperabilidade sintática e semântica entre diferentes agentes.	Parcialmente
	7	Definir meios para o armazenamento e tratamento de informações contextuais históricas	Parcialmente
Relacionados à adaptação	8	Possibilitar a realização de adaptações em diferentes estruturas da arquitetura interna dos agentes.	Sim
	9	Verificar dependências entre o componente de software que está sendo modificado e os demais componentes do agente.	Não
	10	Permitir aos agentes utilizar diferentes tipos de mecanismos para raciocínio ou aprendizado.	Parcialmente
	11	Disponibilizar um repositório de adaptações que possa ser expandido pelo agente por meio de aprendizado ou por colaboração com outros agentes.	Parcialmente
	12	Permitir que o agente avalie as adaptações realizadas.	Parcialmente
	13	Realizar as modificações no agente sem que os atrasos adicionados pela adaptação possam ser percebidos pelo usuário.	Não

Quanto à forma de representação e armazenamento das informações contextuais, ainda há algumas questões pendentes (por isto que se considera que os requisitos 4 e 6 estão contemplados de maneira parcial). Como será visto no próximo capítulo, na implementação das aplicações exemplo optou-se por utilizar estruturas de dados ao invés de instâncias na ontologia que representa os tipos de informações contextuais utilizadas pelos agentes do SMA. Embora o uso dessas estruturas garanta uma rápida atualização nos contextos dos agentes, essa não é uma forma robusta de se representar as informações contextuais (então não se tem conformidade completa com o requisito 4). Também, como o contexto atual de um agente não é representado na forma de uma ontologia, não há interoperabilidade semântica entre os contextos de diferentes agentes (então não se pode dizer que a arquitetura contempla completamente o requisito 6). No entanto, é utilizada uma ontologia para descrever os tipos de informações contextuais disponíveis no gerente de contexto do ambiente multiagentes. Logo, no nível de ambiente, há representação semântica dos tipos de informações contextuais disponíveis (por isto que se fala que o requisito 6 é contemplado de forma parcial).

O requisito 7, que trata da definição de meios para o armazenamento e tratamento das informações contextuais históricas, também é atendido de forma parcial, uma vez que as informações são armazenadas, mas elas ainda não são utilizadas para tentar prever situações no futuro (o que corresponderia ao tratamento de tais informações).

Quanto à adaptação ao contexto, a arquitetura desenvolvida contempla, de forma completa, o requisito 8. Já os requisitos 10, 11 e 12 são contemplados de forma parcial. A avaliação e conformidade com os requisitos 9 e 13 ficaram fora do escopo do trabalho, conforme justificativas a seguir.

O requisito 8 refere-se à possibilidade de realizar adaptações em diferentes estruturas da arquitetura interna dos agentes. De fato, considerando todos os possíveis usos das primitivas genéricas definidas, com a arquitetura **K2** é possível realizar 39 tipos de adaptação, o que é um número bastante expressivo. Também, como as primitivas foram definidas com base em um metamodelo para a criação de agentes que, por sua vez, foi criado e refinado a partir de uma análise de várias arquiteturas descritas na literatura, pode-se dizer que o conjunto de primitivas é abrangente o suficiente para mapear os elementos presentes nas diferentes propostas.

O requisito 10 é dito contemplado de forma parcial, pois, embora não se tenha trabalhado diretamente com diferentes mecanismos para raciocínio e aprendizado, é possível customizar alguns mecanismos da arquitetura para que isto seja feito. Já os requisitos 11 e 12 são considerados parcialmente contemplados porque os conceitos e mecanismos oriundos do *framework* Ontowledge não se encontram completamente implementados e operantes na arquitetura **K2** (eles foram definidos no modelo conceitual, mas não foram implementados e verificados). Com a completa implementação de tais conceitos e mecanismos, os agentes adaptativos ao contexto poderão ampliar seu repositório de adaptadores através da colaboração com outros agentes (o que garantirá o aprendizado multiagentes citado no requisito 11). O requisito 12 será contemplado por outra funcionalidade fornecida pelo *framework* Ontowledge, que é a verificação da satisfação obtida pelos agentes no alcance de seus objetivos. Com tal avaliação, será possível deduzir o sucesso ou não das adaptações realizadas.

Por fim, os requisitos 9 (relacionado à verificação de dependências entre os componentes antes de qualquer adaptação) e 13 (relacionado à verificação dos atrasos adicionados com as adaptações) não foram contemplados pela arquitetura. Ambos os

requisitos ficaram fora do escopo do trabalho neste primeiro momento, mas estão listados como trabalhos futuros.

No próximo capítulo será apresentado o desenvolvimento de três aplicações, visando demonstrar a viabilidade e os ganhos alcançados com a arquitetura **K2**.

5. DESENVOLVENDO APLICAÇÕES COM A ARQUITETURA K2

A arquitetura **K2** oferece uma série de facilitadores para a criação de agentes adaptativos ao contexto. Para ilustrar alguns desses facilitadores, este capítulo apresenta o desenvolvimento de três aplicações em diferentes domínios. A primeira aplicação, descrita na Seção 5.1, mostra um agente corretor de ações que adapta sua estratégia de venda de acordo com informações do mercado financeiro. As duas outras aplicações se baseiam em uma das informações contextuais mais utilizadas em aplicações desenvolvidas no contexto da computação pervasiva, que é a localização de dispositivos móveis. A aplicação descrita na Seção 5.2 apresenta um painel de informações cujo conteúdo é adaptado de acordo com as pessoas próximas a ele. Já a aplicação descrita na Seção 5.3 mostra como veículos podem alterar suas rotas de acordo com a percepção de outros veículos na mesma via ou em um cruzamento.

Para a apresentação das aplicações, foram salientados os aspectos funcionais do problema e a definição dos adaptadores. No decorrer do texto, no entanto, são apresentados alguns trechos de código-fonte para auxiliar na ilustração de questões referentes à implementação. A implementação dos exemplos foi feita em linguagem Java (JSDK), no ambiente de desenvolvimento integrado Eclipse [Ecl11a].

5.1. Agentes adaptativos na Bolsa de Valores – a Aplicação *SmartBroker*

A aplicação *SmartBroker* foi desenvolvida para demonstrar a aplicabilidade da arquitetura proposta no domínio da Bolsa de Valores, onde as informações contextuais são adquiridas com base na análise de notícias e cotações. Na aplicação, há agentes que possuem o objetivo de vender ações com o melhor preço possível em um intervalo de tempo pré-estabelecido (as ações devem ser negociadas, necessariamente, antes do fechamento das operações da Bolsa de Valores no dia em que o objetivo for inicializado). Para alcançar o objetivo, os agentes utilizam um plano que é executado sempre que o usuário solicita a venda de determinado conjunto de ações.

A Figura 5.1 mostra um esquema geral da aplicação desenvolvida. Nela, pode-se ver que existe um agente adaptativo ao contexto, o agente *Broker*, que tem como objetivo vender ações. Para alcançar seu objetivo, o agente possui um plano que é formado por três ações executadas na seguinte seqüência: (1) configurar venda de ações; (2) verificar melhor horário para venda (usando a estratégia EMA 6-12, conforme descrição no

decorrer desta seção); e (3) concluir venda. Para se adaptar, o agente possui dois adaptadores, um para o caso de contexto otimista e outro para o caso de contexto pessimista. A classificação de contexto otimista ou pessimista é dada através de uma análise na cotação do dólar e na variação dos principais índices das bolsas mundiais. Para coletar essas informações contextuais, há dois monitores de fontes de informação e dois interpretadores.

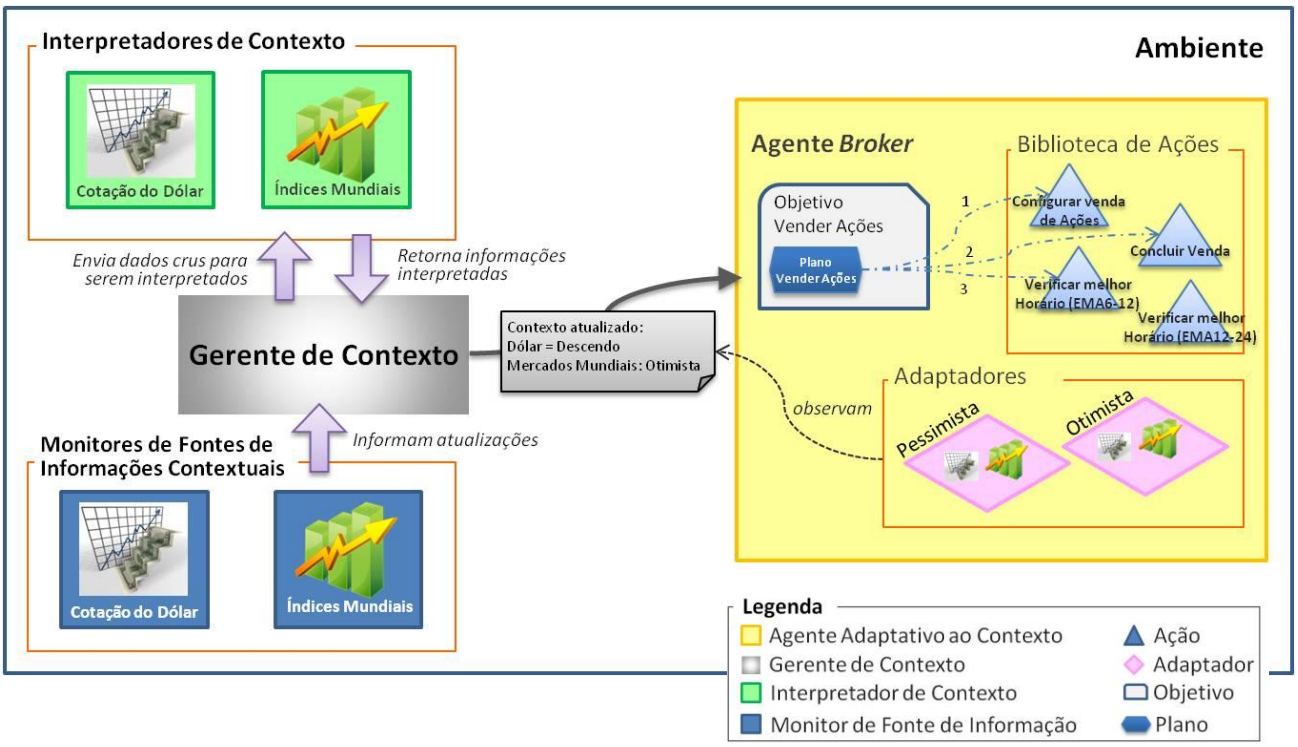

Figura 5.1 – Visão geral dos elementos da aplicação *SmartBroker*.

Para implementar o cenário descrito, várias classes apresentadas nas Figuras 4.3 (que mostra o diagrama de classes em projeto da arquitetura) e 4.4 (que mostra as classes da plataforma simplificada desenvolvida) foram especializadas. A Figura 5.2 apresenta um fragmento do diagrama de classes em projeto da aplicação *SmartBroker* já com essas especializações. A classe `BrokerAgent`, que é uma especialização da classe `SimpleAgent`, representa o agente que possui o objetivo de vender ações (objetivo com identificação *GoalSellStocks*). Para o alcance de tal objetivo, há um plano chamado *SellStocksPlan*. Tanto o objetivo quando o plano citados não aparecem na Figura 5.2, porque são objetos respectivamente, das classes `SimpleAgentGoal` e `Plan`. Ao final da seção, há um diagrama de objetos que ilustra o estado do agente no momento subseqüente a realização da adaptação (Figura 5.7), onde tais objetos podem ser visualizados.

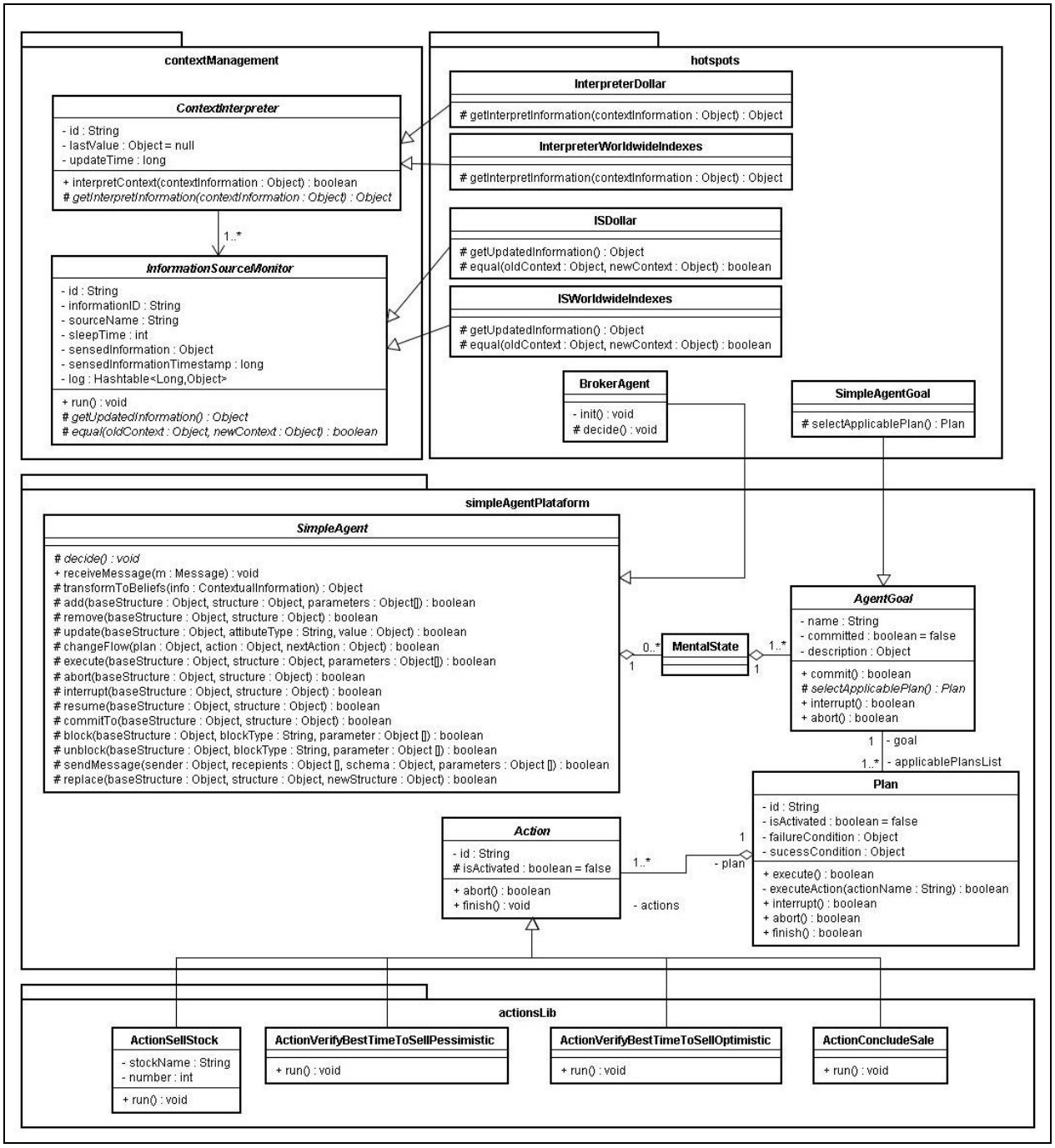

Figura 5.2 – Fragmento do diagrama de classes em projeto da aplicação *SmartBroker*.

Dentro do pacote `actionsLib` há quatro especializações da classe `Action`. A ação `ActionSellStock`, primeira ação do plano a ser executada, é a responsável por colher as informações necessárias para a realização das outras ações. É ela quem pergunta o nome da ação a vender e também a quantidade de ações que devem ser negociadas. Depois disto, é de responsabilidade de uma das ações `ActionVerifyBestTimeToSellPessimistic` ou `ActionVerifyBestTimeToSellOptimist` a análise do melhor momento para a venda das ações, sempre respeitando o horário de fechamento do mercado (são essas duas ações que encapsulam as estratégias de

venda). Por padrão, a análise do melhor momento para venda é feita pela ação `ActionVerifyBestTimeToSellPessimistic`. Por fim, é na ação `ActionConcludeSale` que as ações são de fato vendidas. A seguir, na Figura 5.3, é mostrado o código-fonte da classe `BrokerAgent` (onde é feita a inicialização dos objetivos e planos do agente).

```
public class BrokerAgent extends SimpleAgent
{
 public BrokerAgent (String name)
 {
        super (name);
        init ();
 }

 private void init ()
 {
        Plan p1 = new Plan ("PlanSellStocks");
        p1.addAction("ActionSellStock");
        p1.addAction("VerifyBestTimeToSellPessimistic");
        p1.addAction("ActionConcludeSale");

        this.add(this, new Goal (this, "GoalSellStocks", p1), null);
 }

 protected void decide ()
 {
        // a venda de ações é inicializada pelo usuário
 }
}
```

Figura 5.3 – Código-fonte da classe `BrokerAgent`.

A estratégia utilizada na ação `ActionVerifyBestTimeToSellPessimistic` é baseada no cruzamento de duas médias móveis exponenciais (do inglês *exponential moving average* - EMA), uma curta (6 períodos) e outra longa (12 períodos). De acordo com Elder [Eld06], “as médias móveis estão entre as ferramentas mais antigas, mais simples e mais úteis para os operadores”. As médias móveis, sejam elas aritméticas ou exponenciais, ajudam a identificar tendências. Quanto maior for a janela temporal utilizada na construção de uma média móvel, mais regular ela será (médias móveis com períodos grandes reagem de forma mais lenta às mudanças de tendências). Já as médias móveis com períodos curtos monitoram melhor os preços, mas estão mais sujeitas a guinadas inesperadas e, quando o período analisado é muito pequeno, a média fica excessivamente exposta às variações de preços, perdendo sua utilidade como indicadora de tendências [Eld06].

Um sistema de médias móveis duplas (que foi aplicado na aplicação desenvolvida) é utilizado para identificar tendências e verificar posições de compra e venda de ações. Nesses sistemas, a média móvel mais longa é utilizada para indicar a

tendência e a média móvel mais curta para identificar os pontos de entrada [Eld06]. Quando a média móvel de curto período cruza a média móvel de longo período para cima, considera-se tendência de alta. Esse seria um bom momento de entrar no mercado (comprar ações). Já se a média móvel de curto período cruza a média móvel de longo período para baixo, é um sinal de saída (ou momento de vender ações). Assim, no seu comportamento padrão, um agente do tipo `BrokerAgent` aguarda o cruzamento das médias móveis exponenciais de períodos 6 e 12 para ofertar as ações no mercado e realizar a venda (as ações são ofertadas no momento que a média móvel de período 6 cruza a média móvel de período 12 para baixo), conforme estratégia implementada na ação `ActionVerifyBestTimeToSellPessimistic`.

Como médias móveis com períodos curtos são mais suscetíveis as variações do mercado, a estratégia padrão utilizada pelo agente `BrokerAgent` consiste em vender as ações a qualquer sinal de queda. Até aqui, não há qualquer adaptação estrutural ou comportamental do agente ao contexto. Porém, quando o mercado está otimista, poderia ser utilizada uma estratégia menos conservadora, com médias móveis de período maior (que são menos suscetíveis a pequenas oscilações), como as de período 12 e 24. Desta forma, a venda das ações poderia ser postergada e os lucros ampliados, uma vez que num contexto otimista, é possível que a ação se valorize ao longo do dia.

Vários fatores externos - como a cotação do dólar e a situação dos índices das bolsas mundiais (se em alta ou em queda) - podem influenciar na cotação de uma ação e indicar o “humor” do mercado. O valor do dólar americano é um forte indicador do que se pode esperar dos preços das ações[18]. No geral, o preço do dólar e o índice Bovespa (principal indicador do desempenho das cotações do mercado de ações brasileiro) caminham em sentidos opostos na imensa maioria das vezes. Assim, quando a cotação do dólar cai, geralmente o preço das ações é valorizado e vice-versa. Uma avaliação do otimismo dos índices mundiais também pode ser importante na hora de decidir por vender ou comprar ações. Os índices de uma bolsa de valores refletem o desempenho das ações nela negociadas. Quando o otimismo prevalece em um índice de uma bolsa mundial, significa que o desempenho médio de um determinado grupo de ações teve valorização[19].

Para acompanhar a cotação do dólar e a situação dos índices mundiais, foram criadas duas especializações da classe `InformationSourceMonitor`: a classe

[18] Com base nas resenhas publicadas no site http://www.investmax.com.br/iM/content.asp?contentid=843.
[19] Com base em http://oglobo.globo.com/economia/seubolso/mat/2006/08/11/285223286.asp.

`ISDollar` e a classe `ISWorldwideIndexes`. As duas especializações utilizam como fonte de informação o *site* da ADVFN[20]. Depois de capturadas as informações no *site*, são utilizados dois diferentes interpretadores para processá-las. O interpretador `InterpreterDollar` (especialização da classe `ContextInterpreter`) converte um valor do tipo "*cotação do dólar/variação*" em uma *string* cujos valores possíveis são "*Rising*" ou "*Descending*". Já o `InterpreterWorldwideIndexes` (também especialização da classe `ContextInterpreter`) foi criado para analisar o otimismo ou pessimismo dos principais índices de bolsas mundiais (a saber, Dow Jones, Nasdaq, Ibovespa e Shangai). O interpretador, depois de avaliar as informações providas por um objeto da classe `ISWorldwideIndexes`, retorna uma *string* com o valor "*Optimistic*", "*No tendency*" ou "*Pessimistic*".

A adaptação realizada no agente `BrokerAgent` consiste em substituir, sempre que o contexto se mostra favorável (ou seja, quando o interpretador `Dollar` estiver indicando o valor "*Rising*" e o interpretador `WordWideIndexes` estiver indicando o valor "*Optimistic*"), a ação `ActionVerifyBestTimeToSellPessimistic` (que utiliza médias móveis com períodos 6 e 12) pela ação `ActionVerifyBestTimeToSellOptimistc` (que utiliza médias móveis com períodos 12 e 24). Assim, no contexto otimista o agente `BrokerAgent` oferta as ações apenas quando a média móvel de período 12 cruza a média móvel de período 24 para baixo.

Para viabilizar a troca de estratégias de venda, foram definidos dois adaptadores (objetos da classe `Adaptor`). O adaptador de nome *AdaptorOptimisticStrategy* é o responsável por substituir a ação `ActionVerifyBestTimeToSellPessimistic` pela ação `ActionVerifyBestTimeToSellOptimistic` no plano de identificação *SellStocksPlan*. Como esse adaptador faz uma modificação estrutural no agente (que permanece após o alcance do objetivo), foi definido também o adaptador *AdaptorPessimisticStrategy* que desfaz a alteração realizada pelo outro adaptador (remove a ação `ActionVerifyBestTimeToSellOptimistic` e acrescenta novamente a ação `ActionVerifyBestTimeToSellPessimistic`).

As Tabelas 5.1 e 5.2 descrevem os dois adaptadores criados na aplicação *SmartBroker*. Como pode ser visto nas tabelas, ambos os adaptadores são do tipo *one shot*, o que significa que eles são desativados depois da primeira execução. Tal

[20] http://br.advfn.com/p.php.

configuração é justificada pelo tipo de adaptação realizada, uma vez que a nova ação precisa ser adicionada apenas uma vez, que é quando o contexto alcança o estado indicado no ponto de adaptação. O fato de um adaptador ser desativado depois da primeira execução não quer dizer que ele não poderá ser executado novamente. Na aplicação *SmartBroker*, por exemplo, como os dois adaptadores estão relacionados (observe a propriedade "Ativar adaptador para anular"), a ativação de um é realizada de forma automática logo após a realização do outro. Assim, o agente vai sendo adaptado sempre que necessário. Cabe salientar que apenas o adaptador *AdaptorOptimisticStrategy* é ativado na criação (o adaptador *AdaptorPessimisticStrategy* é mantido desativado até a execução do adaptador *AdaptorOptimisticStrategy*, visto que sua função é restaurar o comportamento padrão do agente).

Tabela 5.1 – O adaptador *AdaptorOptimisticStrategy*.

Nome: AdaptorOptimisticStrategy	
One shot: (X) Sim ( ) Não	**Ativo**: (X) Sim ( ) Não
Adaptação permanente: ( ) Sim (X) Não **Ativar adaptador para anular:** AdaptorPessimistcStrategy	
Descrição: Dólar, Mercados Mundiais, Vender ações, Bolsa de Valores, Broker, Médias Móveis 12-24.	
Ponto de adaptação: contextUpdate (Interpreter: Dollar, =, value: "Descending") **&** contextUpdate (Interpreter: WordWideIndexes, =, value: "Optimistic")	
Variante: remove (Plan: SellStocksPlan, Action: ActionVerifyBestTimeToSellPessimistic) changeFlow(Plan: SellStocksPlan, Action: ActionSellStock, Action: ActionVerifyBestTimeToSellOptimistic)	

Tabela 5.2 – O adaptador *AdaptorPessimisticStrategy.*

Nome: AdaptorPessimisticStrategy	
***One shot*:** (X) Sim ( ) Não	**Ativo**: ( ) Sim (X) Não
Adaptação permanente: ( ) Sim (X) Não **Ativar adaptador para anular:** AdaptorOptimisticStrategy	
Descrição: Dólar, Mercados Mundiais, Vender ações, Bolsa de Valores, Broker, Médias Móveis 6-12.	
Ponto de adaptação: **!** contextUpdate (Interpreter: Dollar, =, value: "Descending") **|** **!** contextUpdate (Interpreter: WordWideIndexes, =, value: "Optimistic")	
Variante: remove (Plan: SellStocksPlan, Action: ActionVerifyBestTimeToSellOptimistic) changeFlow(Plan: SellStocksPlan, Action: ActionSellStock, Action: ActionVerifyBestTimeToSellPessimistic)	

Para a descrição dos adaptadores, foram utilizadas *strings* que representam os principais conceitos relacionados à adaptação realizada. No futuro, pretende-se utilizar ontologias de domínio para fazer tal descrição. Com o uso de ontologias, poderão ser feitas inferências para verificar a compatibilidade de um adaptador com uma necessidade do agente.

O adaptador *AdaptorOptimisticStrategy* é executado quando o contexto está otimista, ou seja, quando os índices mundiais estão otimistas e a cotação do dólar está negativa. Na Tabela 5.1, isto pode ser visto na linha “Ponto de Adaptação”. Na variante desse adaptador, observa-se a utilização de duas primitivas, a primitiva `remove` (para que seja removida a ação `ActionVerifyBestTimeToSellPessimistic`, que tem uma estratégia dita “pessimista”) e a primitiva `changeFlow`, que permite adicionar uma ação a um plano, alterando o seu fluxo de ações.

Já o adaptador *AdaptorPessimisticStrategy* é executado quando os índices mundiais deixam de ser otimistas ou a cotação do dólar deixa de ser negativa, como pode ser visto na linha “Ponto de Adaptação” da Tabela 5.2. Na variante desse adaptador também são utilizadas as primitivas `remove` e `changeFlow`.

O código-fonte dos aspectos criados em AspectJ quando os adaptadores *AdaptorOptimisticStrategy* e *AdaptorPessimisticStrategy* são ativados no contexto do agente de nome *BrokerAgent1* estão ilustrados, respectivamente, nas Figuras 5.4 e 5.5. Como pode ser visto nas figuras, cada aspecto possui um ponto de atuação onde é verificado continuamente (a cada *tick* da arquitetura) se o agente de nome *BrokerAgent1* possui contexto favorável a adaptação. Caso positivo, é executado o trecho de código conectado ao ponto de atuação.

```
package aspectsLib.bolsa;

import general.Environment;
import general.TickEvent;

public aspect AdaptorOptimisticStrategy
{
   pointcut adapt():
       initialization (TickEvent.new ()) &&
       (if(Environment.getAgentInstance("BrokerAgent1")
                   .hasEvent("contextUpdate", "Dollar", "=", "Descending")) &&
        if(Environment.getAgentInstance("BrokerAgent1")
                   .hasEvent("contextUpdate", "WordWideIndexes", "=", "Optimistic")));

   after(): adapt()
   {
      System.out.println ("[Info] Executando AdaptorOptimisticStrategy.");
      SimpleAgent a = Environment.getAgentInstance("BrokerAgent1");
      Plan p = a.getPlan ("SellStocksPlan");
      a.remove (p, "ActionVerifyBestTimeToSellPessimistic");
      a.changeFlow (p, "ActionSellStock", "ActionVerifyBestTimeToSellOptimistic");
      a.informAdaptationExecution("AdaptorOptimisticStrategy");
   }
}
```

Figura 5.4 – Código-fonte do aspecto criado para implantar o adaptador *AdaptorOptimisticStrategy*.

A configuração inicial do ambiente multiagentes da aplicação *SmartBroker*, o que inclui a criação de monitores de informações contextuais, seus interpretadores e a criação

dos próprios agentes, pode ser vista no código-fonte da classe `SmartBroker` (apresentado na Figura 5.6). Cabe salientar que todos estes elementos poderiam ser instanciados em tempo de execução, graças às interfaces gráficas criadas.

Nas últimas linhas do código-fonte apresentado na Figura 5.6, pode-se visualizar a criação de um agente cliente (objeto da classe `ClientAgent`). Conforme discutido nos Capítulos 3 e 4, o serviço responsável por gerenciar o contexto (o gerente de contexto) utiliza objetos da classe `ClientAgent` para armazenar as informações relevantes de cada agente de domínio registrado. O agente cliente instanciado na Figura 5.6 representa o agente de identificação *BrokerAgent1* (instância da classe `BrokerAgent`) e tem interesse nas informações interpretadas pelos interpretadores com identificação *Dollar* e *WorldwideIndexes*. Mais uma vez, a indicação das informações relevantes para dado agente também pode ser feita em tempo de execução utilizando-se interfaces gráficas.

```
package aspectsLib.bolsa;

import general.Environment;
import general.TickEvent;

public aspect AdaptorPessimisticStrategy
{
   pointcut adapt():
        initialization (TickEvent.new ()) &&
        (!if(Environment.getAgentInstance("BrokerAgent1")
                  .hasEvent("contextUpdate", "Dollar", "=", "Descending")) ||
         !if(Environment.getAgentInstance("BrokerAgent1")
                  .hasEvent("contextUpdate", "WordWideIndexes", "=", "Optimistic")));

   after(): adapt()
   {
       System.out.println ("[Info] Executando AdaptorPessimisticStrategy.");
       SimpleAgent a = Environment.getAgentInstance("BrokerAgent1");
       Plan p = a.getPlan ("SellStocksPlan");
       a.remove (p, "ActionVerifyBestTimeToSellOptimistic");
       a.changeFlow (p, "ActionSellStock", "ActionVerifyBestTimeToSellPessimistic");
       a.informAdaptationExecution("AdaptorPessimisticStrategy");
   }
}
```

Figura 5.5 – Código-fonte do aspecto criado para implantar o adaptador *AdaptorPessimisticStrategy*.

Por fim, para exemplificar o comportamento de um objeto da classe `BrokerAgent` em execução, na Figura 5.7 é apresentado um diagrama de objetos que retrata o momento subseqüente à invocação do objetivo *GoalSellStocks* do agente *BrokerAgent1* por seu usuário. Vamos considerar que isto ocorreu por volta das 11h00min do dia 11 de março de 2010 e que as ações que se desejava vender eram da Petrobrás (PETR4). Como o cenário era favorável para adaptação nesse dia (a cotação do dólar abriu negativa – com variação de -0,1416% no fechamento – e os índices das principais

bolsas mundiais estavam com uma leve alta), foi utilizado um sistema de médias móveis com períodos 12 e 24 para identificar o melhor horário para a venda das ações.

A utilização de tal estratégia é resultado da execução do adaptador de nome *AdaptorOptimisticStrategy* (observe, na Figura 5.7, que o adaptador de nome *AdaptorOptimisticStrategy* foi executado e, por ser do tipo *one shot*, já está desativado). Como resultado da execução do adaptador, foi criada uma instância da ação `ActionVerifyBestTimeToSellOptimistic` (e não da ação `ActionVerifyBestTimeToSellPessimistic`, como inicialmente projetado).

```
public class SmartBroker {

 public static void main(String[] args)
 {
    new Environment ();
    AspectFactory.aspectsLib = "src/aspectsLib/smartBroker/*.aj";

    //criação de monitor e interpretador para o dólar
    InformationSourceMonitor isD = new ISDollar ("DollarQuotations", "Dollar",
                                                 System.currentTimeMillis());

    Environment.getContextManager().addInformationSource(isD);

    ContextInterpreter interD = new InterpreterDollar ("Dollar");
    interD.addInformationSource(isD);
    Environment.getContextManager().addContextInterpreter(interD);


    //criação de monitor e interpretador para os índices mundiais
    InformationSourceMonitor isI = new ISWorldwideIndexes ("WorldwideIndexes",
                                       "Indexes", System.currentTimeMillis());

    Environment.getContextManager().addInformationSource(isI);

    ContextInterpreter interI = new InterpreterWorldwideIndexes
                                      ("WorldwideIndexes");

    interI.addInformationSource(isI);
    Environment.getContextManager().addContextInterpreter(interI);


    //criação do agente BrokerAgent
    BrokerAgent ag = new BrokerAgent ("BrokerAgent1");
    Environment.registerAgent(ag);


    //indicação do contexto relevante para o BrokerAgent
    ClientAgent client = new ClientAgent (ag);
    client.addContextInterpreter(interI);
    client.addContextInterpreter(interD);
    Environment.getContextManager().addClientApplication(client);

 }
}
```

Figura 5.6 – Código-fonte da classe principal da aplicação *SmartBroker*.

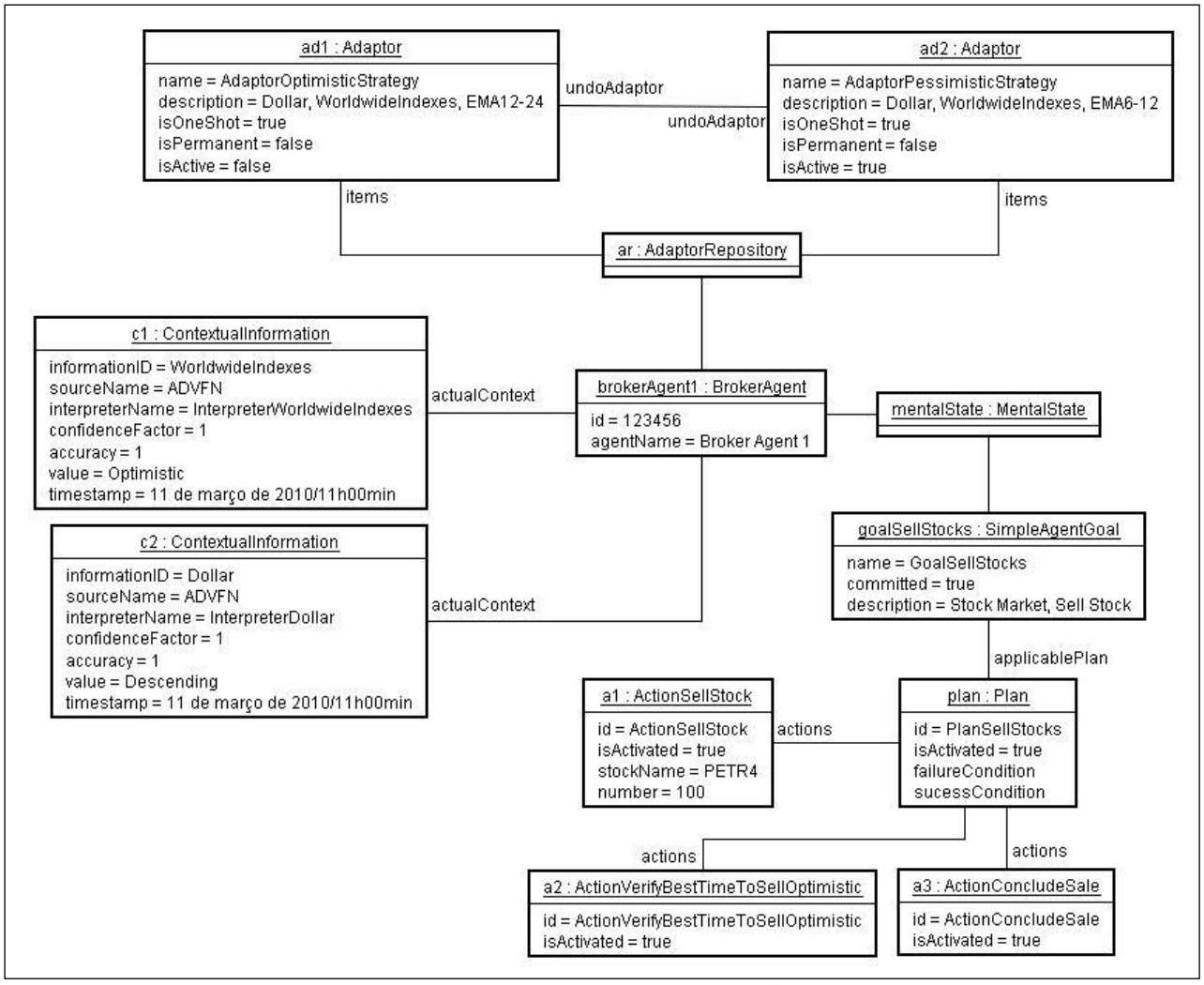

Figura 5.7 – Diagrama de objetos do momento posterior à ativação do objetivo de nome *GoalSellStocks*.

A Figura 5.8 mostra os sistemas de médias móveis duplas criados com as informações da ação PETR4 no dia 11 de março de 2010. Como pode ser visto na Figura 5.8 (a), utilizando-se um sistema de médias com períodos 6-12 a venda ocorreria por volta de 11h40min, sendo que as ações seriam negociadas por aproximadamente R$ 37,00. Com a adaptação, o agente venderia as ações por volta das 13h03min por um preço de aproximadamente R$ 37,29 por ação (Figura 5.8 (b)). A Figura 5.9 apresenta a interface gráfica desenvolvida para o acompanhamento da execução do plano *SellStocksPlan*.

Nesse cenário pode-se ver que, por ser um agente adaptativo ao contexto, o agente *BrokerAgent1* (instância da classe `BrokerAgent`) incrementaria seus lucros em R$ 0,29 por ação negociada (o que corresponde a um valor 0,784% maior do que o valor que seria ganho se ele não fizesse a adaptação). Esse é um valor significativo para grandes volumes de negociações *intraday*.

(a)

(b)

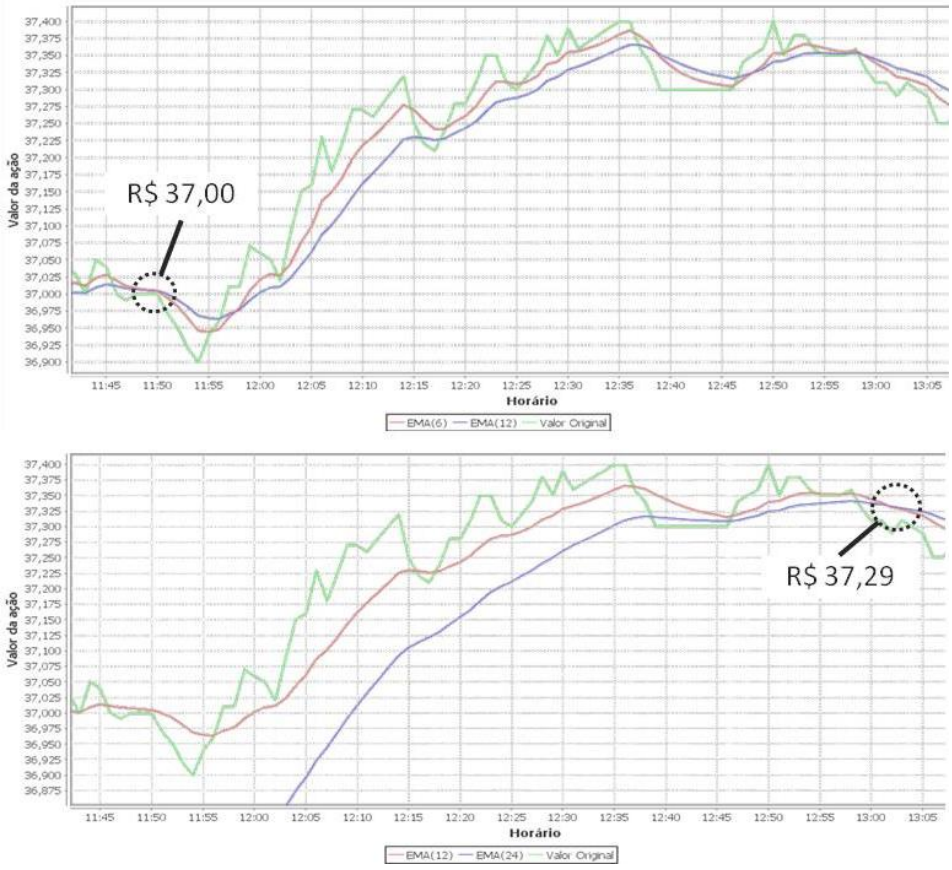

Figura 5.8 – Sistemas de Médias Móveis duplas da Ação PETR4 em 11/03/2010.

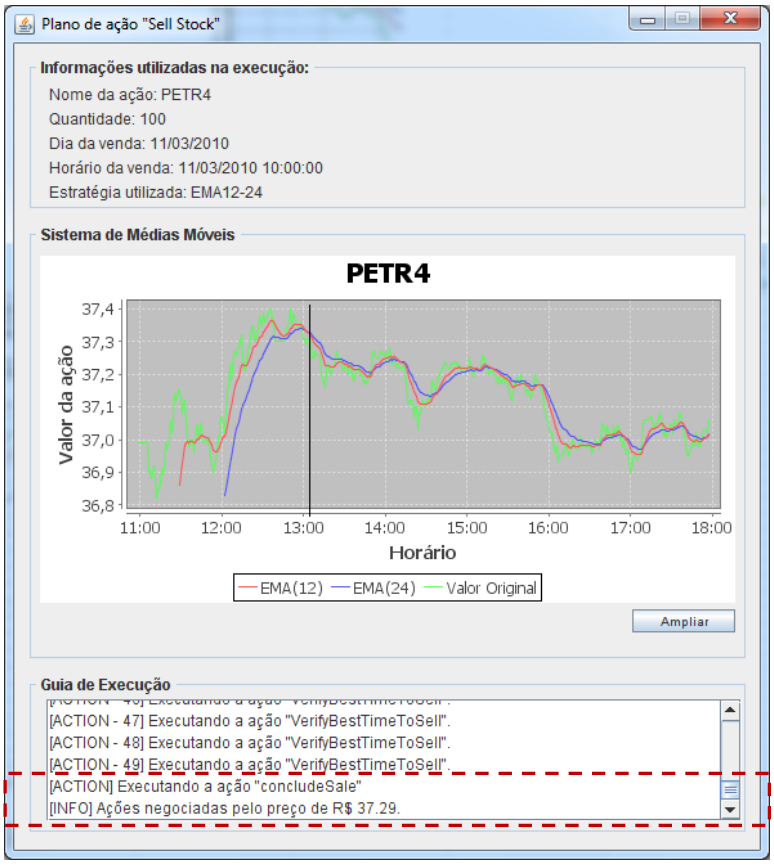

Figura 5.9 – Interface gráfica para o acompanhamento da execução do plano de nome *PlanSellStocks*.

5.2. Painel de informações adaptáveis – a Aplicação *ContextAwarePanel*

A aplicação *ContextAwarePanel* foi desenvolvida para demonstrar a aplicabilidade da arquitetura **K2** em ambientes pervasivos, onde a principal informação contextual é a localização dos dispositivos móveis. Com a localização dos dispositivos, é possível deduzir a presença ou não de pessoas em um ambiente e, a partir disso, é possível adaptar os serviços disponíveis no ambiente de acordo com as pessoas nele inseridas.

Na aplicação *ContextAwarePanel*, há um agente de software responsável por atualizar um painel de informações de acordo com as preferências das pessoas próximas a ele. Na verdade, o agente possui dois objetivos: o primeiro consiste em atualizar as informações do painel de tempos em tempos, mesmo sem identificar a presença de pessoas próximas; o segundo objetivo consiste em atualizar as informações do painel de acordo com as preferências das pessoas próximas, personalizando o conteúdo das informações. A execução de um ou de outro objetivo é feita de forma automática, conforme informações de pessoas próximas capturadas por monitores de informações contextuais espalhados pelo ambiente.

A motivação para o desenvolvimento de tal aplicação surgiu a partir de algumas inovações que procuram aplicar a tecnologia ao varejo. Em janeiro deste ano, aconteceu o evento mundial *NRF Retail's Big Show*[21] em Nova Iorque. Nesse evento, que já é organizado há 100 anos, muito se falou sobre a tecnologia disponível para os clientes e a forma como as empresas devem se adaptar para se relacionar com este novo tipo de cliente, que de posse de seu *tablet*, computador portátil ou celular inteligente, tem acesso a uma grande gama de informação, praticamente o tempo todo. De acordo com a reportagem publicada na Revista Pequenas Empresas e Grandes Negócios (de fevereiro de 2011)[22], entre as inovações discutidas no evento estão as vitrines interativas, os desenhos de produtos em 3D e os *outdoors* digitais.

Da mesma forma que os clientes possuem uma fonte praticamente inesgotável de informação, as empresas também podem utilizar ferramentas digitais para registrar dados de compra e interesses dos clientes para aperfeiçoar o seu atendimento. Ainda na reportagem da revista citada, André Friedheim indica "Com o perfil de consumo, o lojista pode antecipar as demandas do cliente. Aliadas às tecnologias de localização, permite

---

[21] http://events.nrf.com/annual2011.
[22] Reportagem disponível em (mediante login) http://revistapegn.globo.com/Revista/Common/0,,EMI208507-17171,00-VAREJO+FORA+DA+CAIXA.html.

individualizar o atendimento e fazer *marketing* personalizado: por exemplo, o cliente passa na frente da loja e você envia um SMS, convidando-o a entrar".

O mundo digital e a crescente difusão da tecnologia (com preços cada vez mais acessíveis) criaram também um novo padrão de estudante, que está acostumado às facilidades da tecnologia para resolver problemas do dia-a-dia e acessar informações. Para se adequar a esse novo perfil, as instituições de ensino precisam rever suas práticas e a forma como se comunicam com os seus alunos. Assim, tais instituições também poderiam se beneficiar das inovações tecnológicas já utilizadas no varejo para envolver os alunos, dando uma boa aplicação para a tecnologia já utilizada por eles.

O cenário utilizado no desenvolvimento da aplicação *ContextAwarePanel* atua justamente no sentido de aproximar a tecnologia do ambiente de ensino. O painel atualizado pelo agente de software desenvolvido é, na verdade, um painel com notícias, eventos e curiosidades relacionados à FACIN – Faculdade de Informática da PUCRS. No futuro, a idéia é instalar um painel de informações no saguão do prédio da faculdade, o que permitirá aos alunos visualizar informações atualizadas sobre os eventos ocorridos na faculdade, possíveis alterações de salas ou laboratórios, curiosidades, lembretes de avaliações, entre outros. Muitas das informações que podem ser utilizadas para compor as preferências dos alunos (como as disciplinas cursadas e os eventos freqüentados por eles) já estão disponíveis no sistema acadêmico da faculdade, o que facilitará a implantação do sistema no futuro.

A Figura 5.10 mostra um esquema geral da aplicação *ContextAwarePanel*. Nela pode-se ver que existe um agente adaptativo ao contexto, o agente *Painel*, que tem dois objetivos relacionados à seleção e apresentação de informações em um painel eletrônico. O alcance de cada objetivo é garantido por um plano que é composto por apenas uma ação. Para se adaptar, ativando um ou outro objetivo, o agente possui dois adaptadores, um para o caso de pessoas próximas e outro para quando não há pessoas próximas ao painel. A identificação das pessoas próximas ao painel é garantida por um monitor de fonte de informação (que utiliza tecnologia Bluetooth) e um interpretador.

A Figura 5.11 mostra um fragmento do diagrama de classes em projeto da aplicação desenvolvida. Os pacotes `hotspots` e `actionsLib` encapsulam as classes que foram criadas exclusivamente para essa aplicação. A classe `PanelAgent` representa o agente que seleciona as informações e atualiza o conteúdo do painel. As informações disponíveis para publicação são representadas por uma lista de objetos da classe `Information`. Cada informação possui um tipo (se curiosidade, alerta ou notícia), o

conteúdo e uma lista de metadados (que são utilizados para a seleção da informação de acordo com as preferências das pessoas próximas).

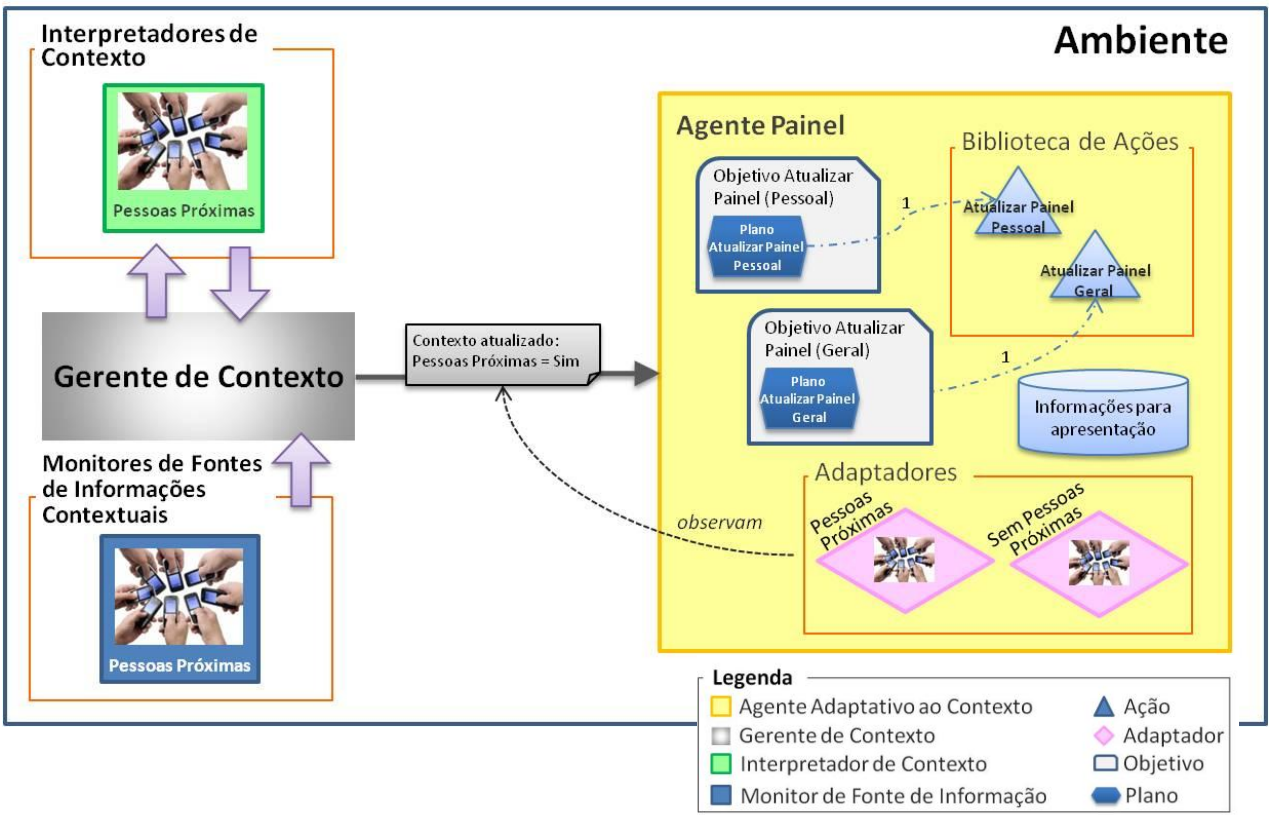

Figura 5.10 – Visão geral dos elementos da aplicação *ContextAwarePanel.*

Para a identificação de pessoas próximas, foi implementada a classe `ISNearPeople` (especialização da classe `InformationSourceMonitor`), que faz descoberta de serviços com base na tecnologia Bluetooth (o módulo de descoberta implementado é responsável por localizar dispositivos no raio de alcance da antena Bluetooth). Para que o monitor consiga identificar um dispositivo, esse deve estar com o Bluetooth ativado e executando a aplicação `AppCel`, que é responsável pelo gerenciamento dos dados do perfil do aluno e pelo envio de tais informação ao monitor. A aplicação `AppCel` executa em modo servidor, ficando no aguardo por conexões Bluetooth durante toda sua execução. Uma vez estabelecida uma conexão, ela envia os dados do perfil do aluno para o monitor. A aplicação `AppCel` foi construída utilizando-se a API JME, com a ferramenta Java Wireless Toolkit 2.5.2 para CLDC 1.1, com perfis MIDP 1.0 a 2.0.

No futuro, quando a tecnologia for de fato implantada na FACIN, os alunos deverão cadastrar seus dispositivos móveis e gerenciar seus perfis em um *site* Web, o que tornará dispensável a execução da aplicação `AppCel` (a própria aplicação *ContextAwarePanel*, de posse da identificação do dispositivo, poderá buscar o perfil do aluno em um repositório de perfis *online*).

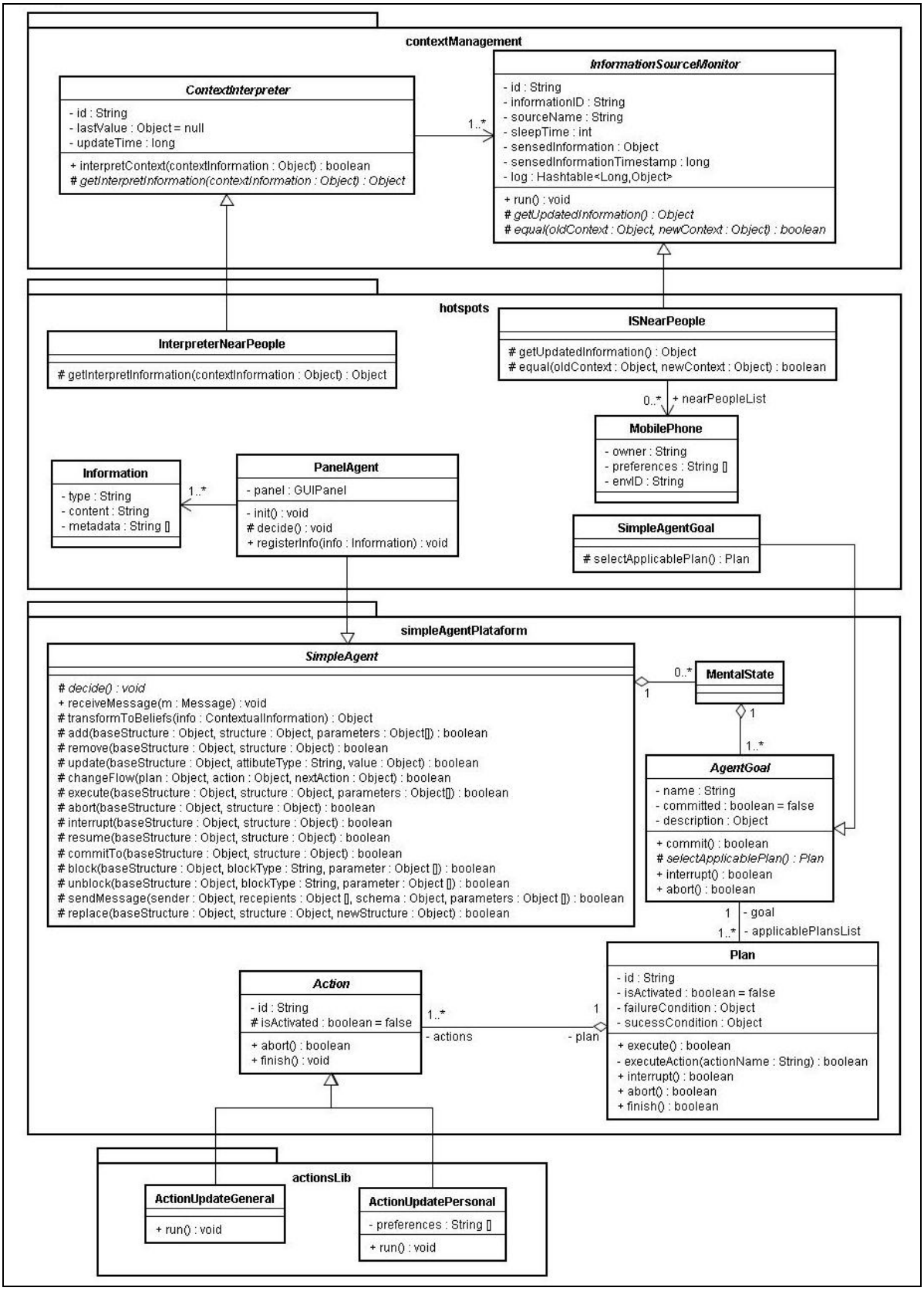

Figura 5.11 – Fragmento do diagrama de classes em projeto da aplicação *ContextAwarePanel*.

Quando um dispositivo móvel é detectado nas proximidades do painel, é criado um objeto da classe `MobilePhone`. É a classe `ISNearPeople` quem mantém uma lista de objetos da classe `MobilePhone`, sendo que cada um desses objetos traz uma referência ao proprietário do dispositivo, uma lista de preferências (recuperada do dispositivo) e a identificação do dispositivo no ambiente (como será visto no decorrer da seção, como pode haver mais de uma pessoa próxima ao painel, para melhorar a identificação das informações relevantes, elas são identificadas por uma paleta de cores). A classe `InterpreterNearPeople` é a responsável por interpretar as informações percebidas pelos objetos da classe `ISNearPeople`, gerando o valor verdadeiro ou falso para a informação contextual `NearPeople`.

Dentro do pacote `actionsLib`, há duas especializações da classe `Action`. A ação `ActionUpdateGeneral` é utilizada pelo plano de nome *PlanUpdatePanelGeneral* no contexto do objetivo de nome *GoalUpdatePanelGeneral*. Depois de ativada, ela é executada de tempos em tempos para selecionar novas informações gerais e atualizar o painel. Já a ação `ActionUpdatePersonal` é utilizada pelo plano *PlanUpdatePanelPersonal* no contexto do objetivo *GoalUpdatePanelPersonal*. Nessa ação, são consideradas as preferências das pessoas próximas para a seleção das informações que serão publicadas no painel. Depois de ativada, essa ação também atualiza o conteúdo do painel de tempos em tempos. Na Figura 5.12 é mostrado o código-fonte da classe `PanelAgent` (onde é feita a inicialização de seus objetivos e planos).

As Tabelas 5.3 e 5.4 descrevem os dois adaptadores criados na aplicação *ContextAwarePanel*. Ambos os adaptadores são responsáveis por fazer adaptações externas em agentes do tipo `PanelAgent`. De acordo com Iman (2004), uma adaptação é considerada externa quando os sistemas internos não são adaptativos, mas as ações externas refletem comportamento adaptativo. De fato, como pode ser visto na descrição dos adaptadores, as mudanças realizadas por eles apenas refletem um comportamento adaptativo, uma vez que só é iniciado o processo de alcance de um objetivo já conhecido pelo agente.

No momento da criação, os dois adaptadores já são ativados (observe a propriedade “Ativo” nas Tabelas 5.3 e 5.4). A partir daí, o painel de notícias vai sendo adaptado de tempos em tempos, seja pela execução do plano *PlanUpdatePanelPersonal* ou pela execução do plano *PlanUpdatePanelPersonal*. Cabe salientar também que os dois adaptadores não são do tipo *one shot*, mas foi adicionada uma cláusula no ponto de

adaptação para verificar se o agente já não está executando um plano no contexto do objetivo que se deseja iniciar (caso estiver, não há porque executar o adaptador).

```
public class PanelAgent extends SimpleAgent {

    private GUIPanel panel;
    private ArrayList<Information> listInfo;

    public PanelAgent (String name)
    {
       super (name);
       init ();
    }

    private void init ()
    {
       listInfo = new ArrayList<Information> ();
       panel = new GUIPanel ();
       gui.setVisible(true);

       Plan p1 = new Plan ("PlanUpdatePanelPersonal");
       p1.addAction("ActionUpdatePersonal");
       p1.addResource("panel", this.panel);
       this.add(this, new Goal (this, "GoalUpdatePanelPersonal", p1), null);

       Plan p2 = new Plan ("PlanUpdatePanelGeneral");
       p2.addAction("ActionUpdateGeneral");
       p2.addResource("panel", this.panel);

       this.add (this, new Goal (this, "GoalUpdatePanelGeneral", p2), null);
    }

    public void registerInfo (Information info)
    {
       this.listInfo.add(info);
    }

    protected void decide ()
    {
          // a atualização do painel é inicializada pelos adaptadores
    }
}
```

Figura 5.12 – Código-fonte da classe `PanelAgent`.

De maneira geral, o adaptador de nome *AdaptorNearPeople* é executado sempre que há pessoas próximas ao painel e não está sendo executado nenhum plano no contexto do objetivo de nome *GoalUpdatePanelPersonal* (o que indica que o agente não está perseguindo o objetivo). A adaptação realizada pelo adaptador consiste no cancelamento da execução do objetivo *GoalUpdatePanelGeneral* (o cancelamento só terá efeito se o objetivo estiver sendo perseguido pelo agente no momento da adaptação) e no posterior comprometimento com o alcance do objetivo *GoalUpdatePanelPersonal*. Assim, na variante do adaptador *AdaptorNearPeople* são utilizadas as primitivas `abort` e `commitTo`. Já o adaptador *AdaptorInfoGeneral* é executado quando não há informações de pessoas próximas ao painel e não está sendo executado nenhum plano no contexto do

objetivo *GoalUpdatePanelGeneral* (cujo alcance será indicado como efeito da adaptação). As Figuras 5.13 e 5.14 mostram os aspectos criados em AspectJ para implantar os dois adaptadores descritos.

Tabela 5.3 – O adaptador *AdaptorNearPeople*.

Nome: AdaptorNearPeople	
One shot: ( ) Sim (X) Não	**Ativo**: (X) Sim ( ) Não
Adaptação permanente: (X) Sim ( ) Não **Ativar adaptador para anular:** -	
Descrição: preferências, informações adaptadas, pessoas próximas, personalização.	
Ponto de adaptação: contextUpdate (Interpreter: NearPeople, =, Value: true) **&** **!** executionStart (Goal: GoalUpdatePanelPersonal, ?Plan)";	
Variante: abort (Agent: FACINPanelAgent, Goal: GoalUpdatePanelGeneral) commitTo (Agent: FACINPanelAgent, Goal: GoalUpdatePanelPersonal)	

Tabela 5.4 – O adaptador *AdaptorInfoGeneral*.

Nome: AdaptorInfoGeneral	
***One shot*:** ( ) Sim (X) Não	**Ativo**: (X) Sim ( ) Não
Adaptação permanente: (X) Sim ( ) Não **Ativar adaptador para anular:** -	
Descrição: seleção de informação, notícias, curiosidade, alertas.	
Ponto de adaptação: contextUpdate (Interpreter: NearPeople, =, Value: false) **&** **!** executionStart (Goal: GoalUpdatePanelGeneral, ?Plan)";	
Variante: abort (Agent: FACINPanelAgent, Goal: GoalUpdatePanelPersonal) commitTo (Agent: FACINPanelAgent, Goal: GoalUpdatePanelGeneral)	

```
package aspectsLib.painel;
import general.Environment;
import general.TickEvent;

public aspect AdaptorNearPeople
{
   pointcut adapt():
     initialization (TickEvent.new ()) &&
     (if(Environment.getAgentInstance("FACINPanelAgent")
            .hasEvent("contextUpdate", "NearPeople", "=", "true")) &&
      !if(Environment.getAgentInstance("FACINPanelAgent")
            .hasEvent("executionStart", "Goal: GoalUpdatePanelPersonal", "?Plan")));

   after(): adapt()
   {
      System.out.println ("[Info] Executando AdaptorNearPeople.");
      SimpleAgent a = Environment.getAgentInstance("FACINPanelAgent");
      a.abort (a, "Goal: GoalUpdatePanelGeneral");
      a.commitTo (a, "Goal: GoalUpdatePanelPersonal");
      a.informAdaptationExecution("AdaptorNearPeople");
   }
}
```

Figura 5.13 – Código-fonte do aspecto criado para implantar o adaptador *AdaptorNearPeople*.

```
package aspectsLib.painel;

import general.Environment;
import general.TickEvent;

public aspect AdaptorInfoGeneral
{
   pointcut adapt():
     initialization (TickEvent.new ()) &&
     (if(Environment.getAgentInstance("FACINPanelAgent")
            .hasEvent("contextUpdate", "NearPeople", "=", "false")) &&
      !if(Environment.getAgentInstance("FACINPanelAgent")
            .hasEvent("executionStart", "Goal: GoalUpdatePanelGeneral", "?Plan")));

   after(): adapt()
   {
      System.out.println ("[Info] Executando AdaptorInfoGeneral.");
      SimpleAgent a = Environment.getAgentInstance("FACINPanelAgent");
      a.abort (a, "Goal: GoalUpdatePanelPersonal");
      a.commitTo (a, "Goal: GoalUpdatePanelGeneral");
      a.informAdaptationExecution("AdaptorInfoGeneral");
   }
}
```

Figura 5.14 – Código-fonte do aspecto criado para implantar o adaptador *AdaptorInfoGeneral*.

A Figura 5.15 mostra a configuração inicial do ambiente multiagentes da aplicação *ContextAwarePanel*. Ao final do código, observa-se o registro de diferentes informações no agente de identificação *FacinPanelAgent*. São essas as informações que serão publicadas no painel a cada atualização. No futuro, pretende-se integrar a aplicação com um banco de dados.

Por fim, para exemplificar o comportamento da aplicação *ContextAwarePanel* em execução, vamos considerar que o painel já esteja instalado no saguão da FACIN e que, num determinado momento, haja dois alunos (identificados como João e Maria) em frente a ele (conforme indicado na Figura 5.16). João é aluno do 1º semestre do curso de Ciência da Computação e se interessa por assuntos relacionados a jogos e também por esportes. Atualmente, cursa as disciplinas Algoritmos e Cálculo A. Maria é formanda do curso de Sistemas de Informação e já procura por cursos de extensão e especializações para cursar depois de faculdade. Atualmente, faz seu trabalho de conclusão de curso na área de Sistemas Multiagentes, portanto tem interesse em notícias sobre esse tema. Outro tema de interesse de Maria é esportes. Nas Tabelas 5.5 e 5.6 são detalhados os perfis dos alunos.

A Figura 5.17 mostra o painel de informações organizado de acordo com o perfil dos dois alunos citados. No painel há tanto informações relevantes para ambos os alunos quanto informações relevantes só para um deles. Para auxiliar na identificação das informações relevantes, é utilizado um ícone colorido antes de cada informação apresentada. Por enquanto, o casamento entre as preferências dos alunos e os

metadados das informações cadastradas é feito por comparação de *strings*. Das informações consideradas relevantes, apenas três de cada tipo (três alertas, três notícias e três curiosidades) são selecionadas para a publicação no painel.

Para finalizar, esse exemplo mostrou que a arquitetura proposta pode ser facilmente utilizada para auxiliar na construção de ambientes pervasivos, onde os usuários não precisam fornecer informações de maneira explícita ao computador. Acredita-se que tal aplicação poderia melhorar consideravelmente a comunicação entre as instituições de ensino e seus alunos, aproveitando a tecnologia existente para a disseminação das informações relevantes de maneira individualizada.

```
public class ContextAwarePanel {

   public static void main(String[] args)
   {
      new Environment ();
      AspectFactory.aspectsLib = "src/aspectsLib/caPanel/*.aj";

       //criação de monitor e interpretador para a identificação de pessoas próximas
      InformationSourceMonitor isNearP = new ISNearPeople ("ISNearPeople",
                                    "NearPeople", System.currentTimeMillis());

      Environment.getContextManager().addInformationSource(isNearP);

      ContextInterpreter interNear = new InterpreterNearPeople ("NearPeople");
      interNear.addInformationSource(isNearP);

      Environment.getContextManager().addContextInterpreter(interNear);

      //criação do agente PanelAgent
      PanelAgent ag = new PanelAgent ("FacinPanelAgent");
      Environment.registerAgent(ag);

      //exemplos de registro das informações para colocar no painel
      ag.resgisterInfo(new Information ("alerta", "A aula de Redes será na " +
                                    "sala 41. A aula começará em 15 minutos.",
                                    new String [] {"RedesT11"}));
      ag.registerInfo (new Information ("noticia", "A Facin cria mais um " +
                                    "curso de extensão em POO. Inscreva-se já.",
                                    new String [] {"POO"}));
      ag.registerInfo (new Information ("alerta", " Alunos matriculados em " +
                                    " Algoritmos: sala 216 - Prédio 32.",
                                    new String [] {"AlgoritmosT11"}));


      //indicação do contexto relevante para o PanelAgent
      ClientAgent client = new ClientAgent (ag);
      client.addContextInterpreter(interNear);
      Environment.getContextManager().addClientApplication(client);

   }
}
```

Figura 5.15 – Código-fonte da classe principal da aplicação *ContextAwarePanel*.

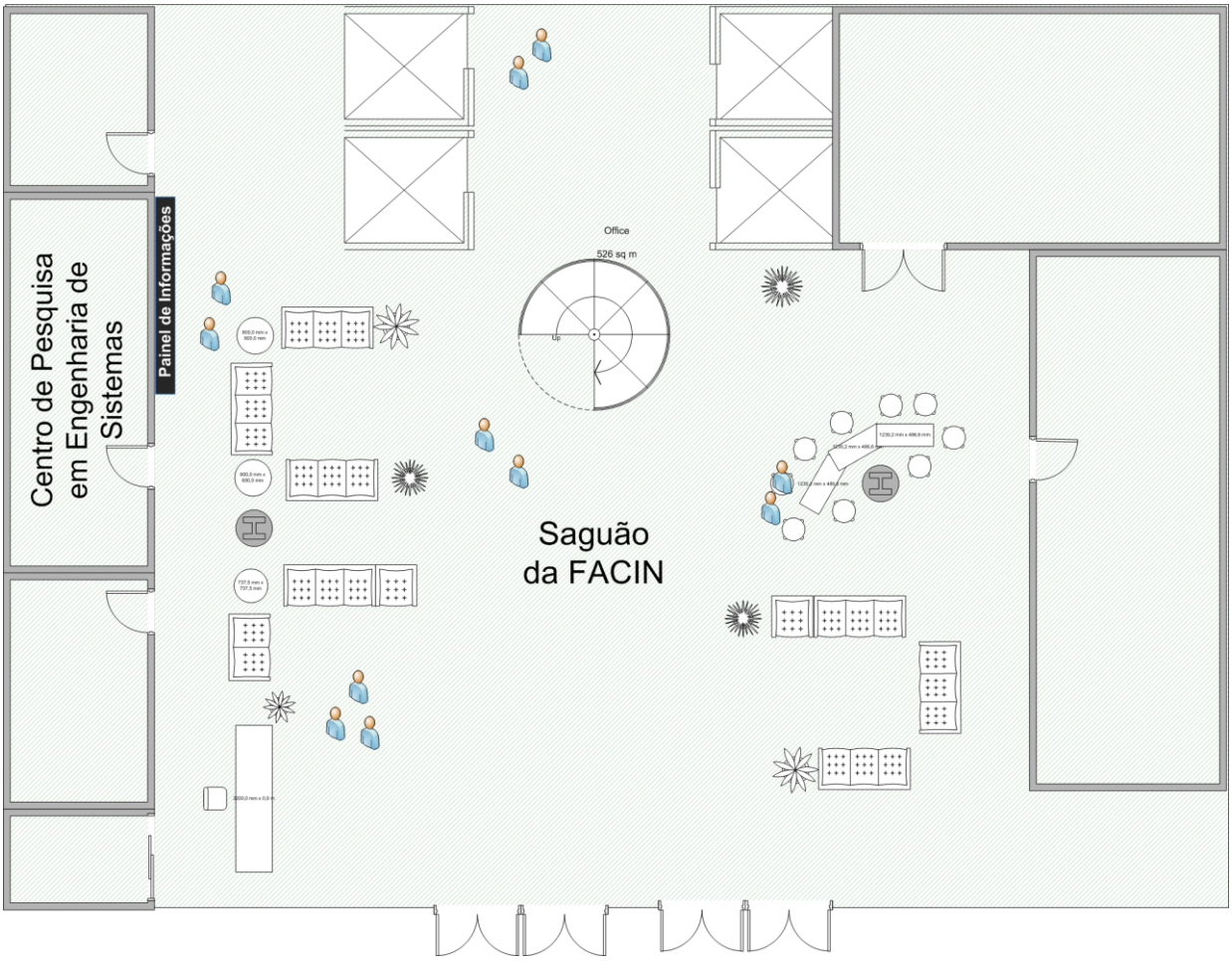

Figura 5.16 – Planta baixa do saguão da FACIN com o painel já instalado.

Tabela 5.5 – Informações do perfil da aluna Maria.

Nome:	**Maria**
Curso:	Sistemas de Informação
Semestre:	8
Disciplinas cursadas:	Trabalho de Conclusão de Curso 2
Assuntos de interesse:	Cursos de extensão, especializações, sistemas multiagentes, esportes.

Tabela 5.6 – Informações do perfil do aluno João.

Nome:	João
Curso:	Ciência da Computação
Semestre:	1
Disciplinas cursadas:	Algoritmos e Cálculo A
Assuntos de interesse:	Jogos, esportes, estágios.

Figura 5.17 – Painel de informações considerando os perfis de Maria e João.

5.3. Veículos urbanos adaptáveis – a Aplicação *SmartCars*

A terceira e última aplicação desenvolvida refere-se ao domínio de veículos inteligentes. Nela, são simulados veículos que, através de interação e cooperação, tomam decisões relacionadas à rota a seguir.

No Brasil e no mundo, devido aos problemas causados pelo número cada vez maior de veículos circulando pelas grandes rodovias, muito se fala sobre a criação de "veículos inteligentes". Atualmente, uma das pesquisas que tem obtido destaque é a CVIS - *Cooperative vehicle-infrastructure systems*[23], cuja tecnologia foi desenvolvida pelo engenheiro norueguês Knut Evensen, vice-presidente de pesquisa da empresa Q-free. O projeto CVIS, conforme descrito em seu material de divulgação, "trará grandes melhorias funcionais para motoristas e operadores rodoviários, possibilitando que os veículos se comuniquem e cooperem diretamente com outros veículos próximos e com a infra-estrutura rodoviária" [QFr11].

Uma engenhosa solução tecnológica está sendo desenvolvida no projeto CVIS para permitir que os veículos se comuniquem utilizando um terminal multi-canal capaz de manter a conexão com a internet contínua, organizando o trânsito e oferecendo

[23] Site do projeto: http://www.cvisproject.org/.

informações relevantes para os motoristas. Dentro dos carros, há um computador com uma tela um pouco maior do que uma tela de GPS. Esse computador, além de servir como GPS, é capaz de se conectar com dispositivos espalhados pelas ruas e avisar ao motorista a proximidade de uma escola, os horários do trem (quando o veículo está perto de uma estação), a ocorrência de acidentes ou engarrafamentos (ocorrências avisadas de forma automática por outros veículos trafegando na via), entre outros. A tecnologia já está sendo utilizada na cidade de Trondheim (Noruega). No Brasil, a partir de 2011, devido a uma resolução do Conselho Nacional de Trânsito, começará a ser implantado um chip nos carros que poderá receber e transmitir informações a leitores espalhados pelas ruas.

A aplicação *SmartCars*, desenvolvida para verificar a aplicabilidade da arquitetura **K2**, simula a existência de veículos inteligentes utilizando robôs Lego Mindstorms NXT[24]. Na verdade, há um agente de software adaptativo ao contexto guiando cada um dos robôs. Devido ao limitado poder computacional dos robôs Lego NXT, tais agentes ficam hospedados em outro dispositivo, sendo que os comandos que definem a movimentação do robô (andar para frente, parar, dobrar a direita ou dobrar a esquerda) são enviados a ele via conexão Bluetooth[25].

A Figura 5.18 apresenta um esquema geral da aplicação *SmartCars*. Na figura estão ilustrados o agente que representa o robô e suas estruturas em termos de objetivos, planos e ações. Para se adaptar, o agente possui um adaptador que leva em conta a existência ou não de outros veículos perto de cruzamentos na rota do robô. A identificação de tais veículos é feita por dois monitores de fontes de informação (que utilizam imagens de vídeo para verificar a presença de veículos em determinados locais) e um interpretador. Já a Figura 5.19 mostra um fragmento do diagrama de classes em projeto da aplicação desenvolvida.

A classe `RobotAgent` (especialização da classe `SimpleAgent`) representa o agente adaptativo ao contexto responsável por guiar os robôs Lego NXT. Nela, são definidos três objetivos: o objetivo *GoalTrackRoute*, responsável por indicar os comandos de movimentação para que o robô siga até o destino desejado; o objetivo *GoalSolveConflict*, que é ativado quando detectado perigo em algum cruzamento, sendo responsável por estabelecer a comunicação com o robô que trafega na via perpendicular

[24] http://mindstorms.lego.com/.
[25] No robô Lego NXT é executada uma aplicação que fica aguardando por conexões Bluetooth e que reconhece comandos predefinidos, que são responsáveis por indicar a movimentação do robô. Para o desenvolvimento de tal aplicação, foi utilizada a API LEJOS 0.8.5 (mais informações disponíveis em http://lejos.sourceforge.net/index.php).

para resolver quem cruzará primeiro; e o objetivo *SendProposal*, que é ativado em resposta ao recebimento de uma mensagem de conflito no cruzamento (está relacionado com a negociação entre os agentes). Na aplicação, são criadas duas instâncias da classe `RobotAgent` (uma para cada guiar cada robô).

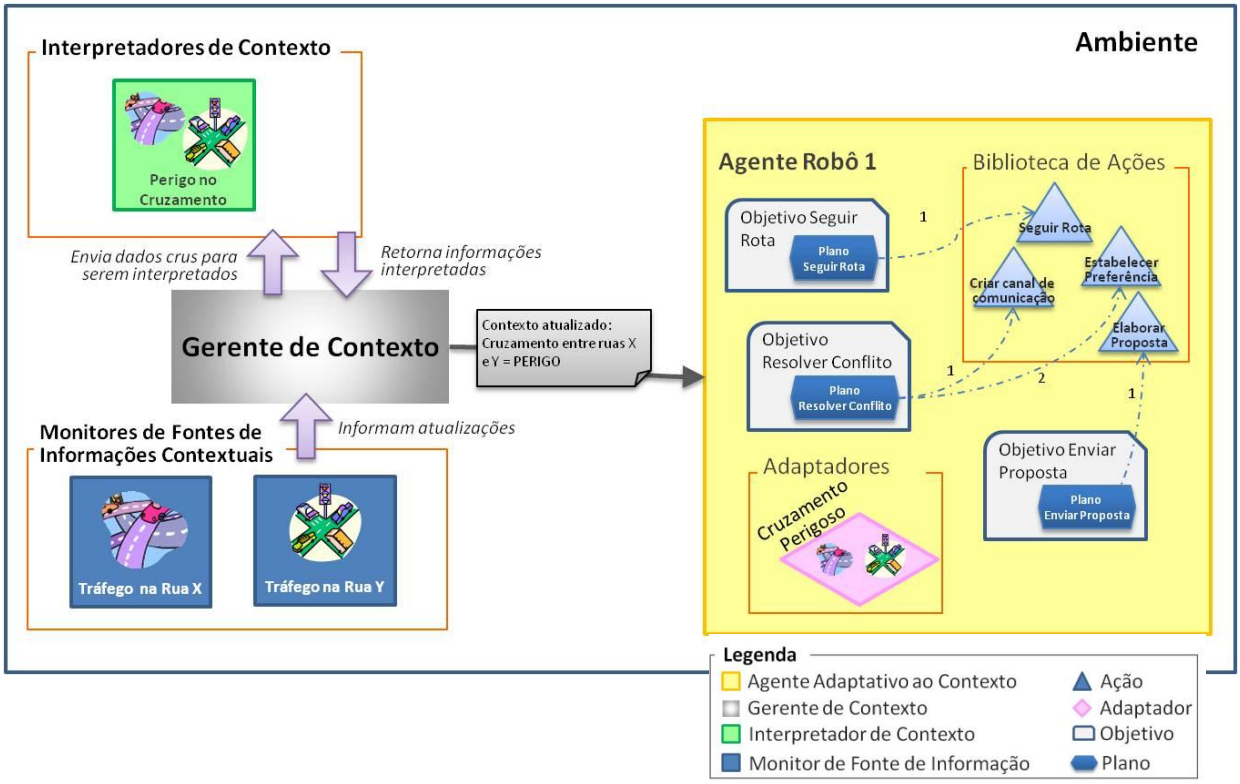

Figura 5.18 – Visão geral dos elementos da aplicação *SmartCars.*

A Figura 5.20 mostra o código-fonte da classe `RobotAgent`. No código-fonte pode-se ver (no método `init`) a declaração dos objetivos, planos e ações do agente. Também, como há interação entre os agentes, foi implementado o método `decide` (nos outros exemplos, como não havia interação, não foi necessário implementar esse método). A implementação do método `decide` mostra como um agente do tipo `RobotAgent` reage as mensagens recebidas de outros agentes através do ambiente. Quando é recebida uma mensagem com assunto *SolveConflict*, é inicializado o objetivo de nome *GoalSendProposal*. O recebimento de uma mensagem com esse assunto indica que tanto o agente que enviou a mensagem quanto o agente que a recebeu estão se aproximando de um cruzamento com rotas perpendiculares. Então, é necessário negociar para que não haja colisão. Mensagens com assunto *SolveConflict* são criadas ao final da execução da ação `ActionCreateCommunicationChannel`, que é executada no contexto do plano de identificação *PlanSolveConflict* para atingir o objetivo *GoalSolveConflict*. Assim, pode-se deduzir que o remetente de uma mensagem com assunto *SolveConflict* está tentando atingir o objetivo *GoalSolveConflict*.

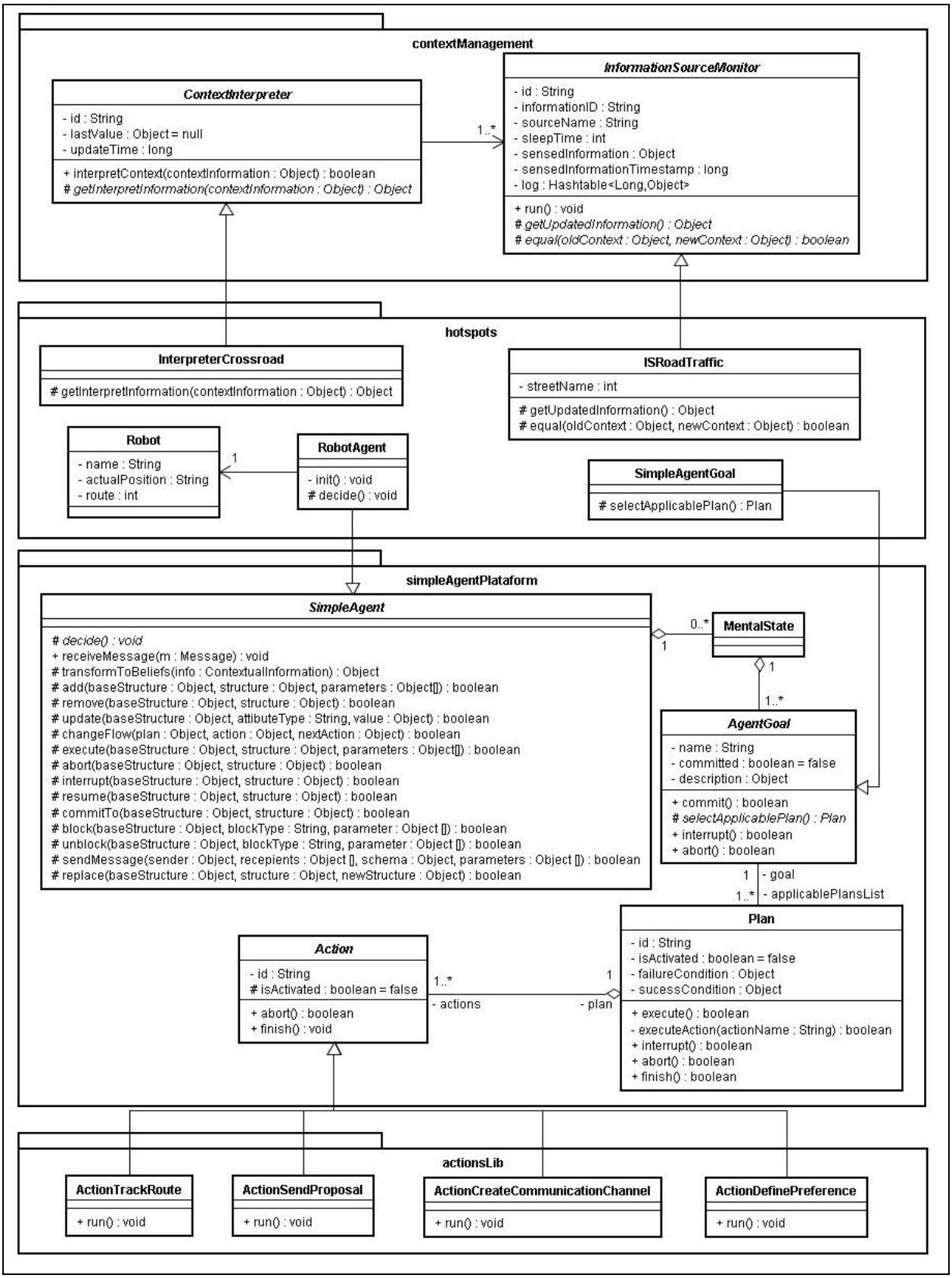

Figura 5.19 – Fragmento do diagrama de classes em projeto da aplicação *SmartCars*.

```
public class RobotAgent extends SimpleAgent
{
     private Robo robot;
     private Message lastReceivedMessage;
     public RobotAgent (String name, Robot robot)
     {
           super (name);
           this.robot = robot;
           init ();
     }

     private void init ()
     {
        Plan p1 = new Plan ("PlanTrackRoute");
        p1.addAction("ActionTrackRoute");

        Plan p2 = new Plan ("PlanSendProposal");
        p2.addAction("ActionSendProposal");

        Plan p3 = new Plan ("PlanSolveConflict");
        p3.addAction("ActionCreateCommunicationChannel");
        p3.addAction("ActionDefinePreference");

        this.add (this, new Goal (this, "GoalTrackRoute", p1), null);
        this.add (this, new Goal (this, "GoalSendProposal", p2), null);
        this.add (this, new Goal (this, "GoalSolveConflict", p3), null);

     }

     protected void decide ()
     {
        if (lastReceivedMessage.getSubject().equals("SolveConflict"))
            this.commitTo(this, this.getGoal ("GoalSendProposal"));

        else if (lastReceivedMessage.getSubject().equals("Proposal"))
            this.getExecEngine().getRunningPlan ("PlanSolveConflict")
                                          .addResource("Proposal", m);

        else if (lastReceivedMessage.getSubject().equals("FreeCrossroad"))
            this.resume (this, this.getGoal ("GoalTrackRoute"));
      }
}
```

Figura 5.20 – Código-fonte da classe `RobotAgent`.

Agentes do tipo `RobotAgent` também reagem a mensagens com assunto *Proposal*. Essas mensagens são criadas durante a execução da ação `ActionSendProposal` e indicam se o agente que recebeu a mensagem com assunto *SolveConflict* irá dar a preferência no cruzamento ou continuará seu percurso. Ao receber uma mensagem com o assunto *Proposal*, o agente a redireciona para o plano de nome *PlanSolveConflict* que já deve estar em execução. No plano, a ação `ActionDefinePreference` aguarda uma mensagem com esse assunto para continuar sua execução. Na ação `ActionDefinePreference` o agente decide, com base na proposta do outro agente, se o robô guiado por ele irá parar no cruzamento ou prosseguir com sua rota. Na implementação atual, o agente que começa a negociação sempre

atende a proposta recebida (ou seja, não há uma segunda rodada de negociações). Por fim, quando o agente recebe uma mensagem com o assunto *FreeCrossroad* (enviada pelo agente que obteve a preferência quando esse sai do cruzamento), ele reinicia o alcance do objetivo de nome *GoalTrackRoute*.

A classe `ISRoadTraffic` (especialização da classe `InformationSourceMonitor`) é a responsável por indicar se há carros trafegando próximos ao cruzamento de determinada rua (o nome da rua é indicado no atributo `streetName`). Já o interpretador `InterpreterCrossroad` interpreta as informações capturas por monitores do tipo `ISRoadTraffic`, indicando se há perigo ou não no cruzamento entre duas ou mais ruas.

Para se adaptar, os agentes do tipo `RobotAgent` possuem um adaptador de nome *AdaptDangerousCrossroad*. Tal adaptador utiliza as informações contextuais providas por um objeto da classe `InterpreterCrossroad` para avaliar a necessidade ou não de adaptação do agente. A adaptação realizada pelo adaptador consiste em inicializar o alcance do objetivo *GoalSolveConflict* sempre que há perigo no cruzamento entre duas vias. Assim, é realizada uma adaptação comportamental no agente. A Tabela 5.7 descreve o adaptador *AdaptDangerousCrossroad*. Cabe salientar que a interrupção temporária do objetivo *GoalTrackRoute* (para que um dos robôs pare no cruzamento e aguarde a travessia do outro robô) é realizada no corpo das ações `ActionSendProposal` ou `ActionDefinePreference`, conforme negociação realizada pelos agentes. A Figura 5.21 mostra o código-fonte do aspecto criado em AspectJ para implantar o adaptador de nome *AdaptDangerousCrossroad.*

Tabela 5.7 – O adaptador *AdaptDangerousCrossroad.*

Nome: AdaptDangerousCrossroad
***One shot*:** ( ) Sim (X) Não **Ativo**: (X) Sim ( ) Não
Adaptação permanente: ( ) Sim (X) Não **Ativar adaptador para anular:** -
Descrição: cruzamento perigoso, definição de preferência.
Ponto de adaptação: contextUpdate (Interpreter: InterpreterCrossroad, =, Value: Dangerous)
Variante: commitTo (Agent: CarRobotAgent, Goal: GoalSolveConflict)

A Figura 5.22 apresenta o cenário desenvolvido para ilustrar a aplicação *SmartCars* em execução. No cenário, há dois veículos (representados por robôs Lego NXT) trafegando em vias perpendiculares (a Rua Santana e a Rua Laurindo). Cada rua contém uma série de sensores, cujas informações captadas são associadas e

processadas por dois monitores de informações contextuais (existe um monitor para cada rua). Para simular os sensores das ruas (que indicam se os robôs estão próximos ou não do cruzamento), foi utilizada uma câmera de vídeo. Na imagem capturada por essa câmera foram marcadas diferentes regiões, sendo que cada região representa um dos sensores indicados na Figura 5.22.

```
package aspectsLib.smartCars;

import general.Environment;
import general.TickEvent;

public aspect AdaptorDangerousCrossroad
{
   pointcut adapt():
     initialization (TickEvent.new ()) &&
     (if(Environment.getAgentInstance("CarRobotAgent")
             .hasEvent("contextUpdate", "InterpreterCrossRoad", "=", "Dangerous")))

   after(): adapt()
   {
       System.out.println ("[Info] Executando AdaptorDangerousCrossroad.");
       SimpleAgent a = Environment.getAgentInstance("CarRobotAgent");
       a.commitTo (a, "Goal: GoalSolveConflict");
       a.informAdaptationExecution("AdaptorDangerousCrossroad");
   }
}
```

Figura 5.21 – Código-fonte do aspecto criado para implantar o adaptador *AdaptorDangerousCrossroad*.

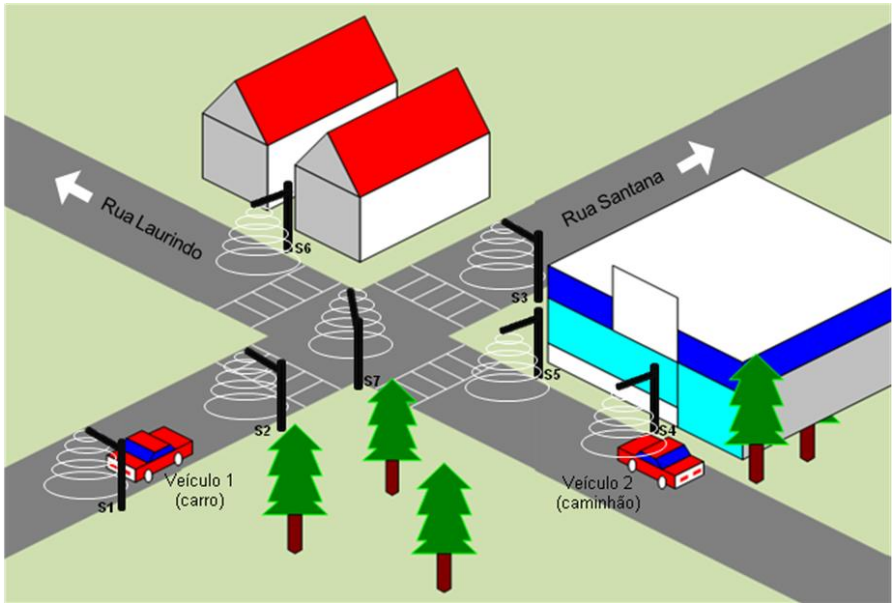

Figura 5.22 – Cenário desenvolvido na aplicação *SmartCars*.

A configuração inicial do ambiente multiagentes da aplicação *SmartCars*, de acordo com o cenário apresentado, pode ser vista na Figura 5.23. Nela é possível observar a criação de duas instâncias da classe `RobotAgent` e também de duas instâncias da classe `ISRoadTraffic` (uma para controlar o tráfego da Rua Santana e outra para controlar o tráfego da Rua Laurindo). A inicialização do objetivo

GoalTrackRoute, o que inclui a indicação da rota que deve ser seguida pelo robô, é feita em tempo de execução utilizando-se uma interface gráfica.

No cenário desenvolvido com os robôs Lego NXT, algumas simplificações foram definidas. Por exemplo, apenas o agente de identificação *CarRobotAgent* (objeto da classe `RobotAgent`) possui o adaptador *AdaptDangerousCrossroad*. Assim, é sempre ele que inicia o processo de negociação para a definição da preferência na travessia do cruzamento. Também, o outro agente utilizado no cenário (de identificação *TruckRobotAgent*) costuma dar preferência para quem inicia a negociação.

Por fim, a Figura 5.24 mostra os robôs Lego NXT em ação durante a execução da aplicação *SmartCars*. Para que os robôs se locomovessem na parte central das ruas, foi feita uma marcação no chão. Os robôs possuem sensores de cores e estão programados para andar sempre na linha marcada.

A aplicação *SmartCars* mostra que a utilização de agentes adaptativos ao contexto pode auxiliar na organização do trânsito das grandes cidades, diminuindo a ocorrência de acidentes, a emissão de multas e os engarrafamentos.

```
public class SmartCars
{
    public static void main(String[] args)
    {
      new Environment ();
      AspectGenerator.aspectsLib = "src/aspectsLib/smartCars/*.aj";

      Robot truck = new Robot ("truck", "Laurindo");
      Robot car = new Robot ("car", "Santana");

      RobotAgent agRobot1 = new RobotAgent ("TruckRobotAgent", truck);
      Environment.registerAgent(agRobot1);

      RobotAgent agRobot2 = new RobotAgent ("CarRobotAgent", car);
      Environment.registerAgent(agRobot2);

      InformationSourceMonitor i = new ISRoadTraffic ("Santana", "S1,S2");
      InformationSourceMonitor i1 = new ISRoadTraffic ("Laurindo", "S3,S4");
      Environment.getContextManager().addInformationSource(i);
      Environment.getContextManager().addInformationSource(i1);

      ContextInterpreter c = new InterpreterCrossroad ("InterpreterCrossroad");
      c.addInformationSource(i);
      c.addInformationSource(i1);
      Environment.getContextManager().addContextInterpreter(c);

      ClientAgent client = new ClientAgent (agRobo2);
      client.addContextInterpreter(c);
      Environment.getContextManager().addClientApplication(client);
    }
}
```

Figura 5.23 – Código-fonte da classe principal da aplicação *SmartCars*.

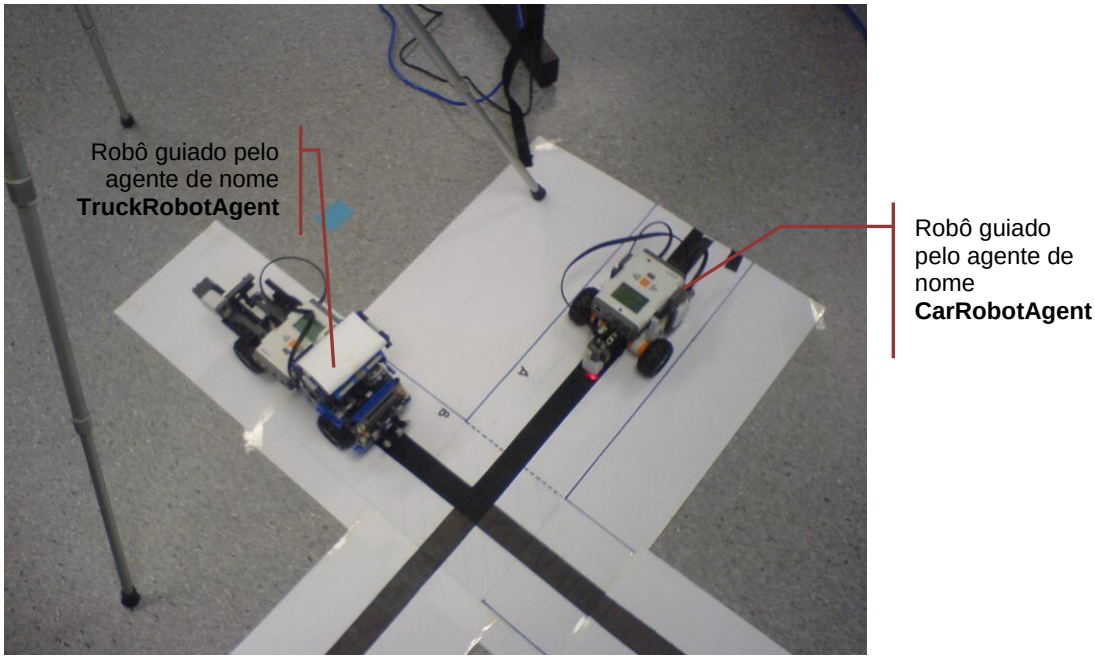

Figura 5.24 – Robôs Lego NXT em ação durante a execução da aplicação *SmartCars*.

5.4. Considerações sobre o capítulo

Esse capítulo apresentou o desenvolvimento de três aplicações que demonstraram a aplicabilidade da arquitetura **K2** em diferentes domínios de negócio. Durante a descrição das aplicações, foram enfatizadas algumas características e conceitos da arquitetura proposta, como é o caso dos adaptadores.

Como pôde ser visto ao longo do capítulo, com o uso de adaptadores é possível modularizar a solução, separando o comportamento adaptativo do restante do comportamento do agente. A implantação dos adaptadores com programação orientada a aspectos também se demonstrou viável neste primeiro momento. Nas aplicações, foram utilizados alguns tipos de eventos internos e primitivas para compor, respectivamente, os pontos de adaptação e as variantes dos adaptadores. Nos pontos de adaptação, foram utilizados, basicamente, dois tipos de eventos: o `contextUpdate` e o `executionStart`. Já nas variantes, foram utilizadas as primitivas genéricas `remove`, `changeFlow`, `abort` e `commitTo`. Embora se possa considerar pequeno o número de diferentes tipos de eventos e primitivas utilizados nos adaptadores criados (tendo em vista a quantidade de eventos internos definidos e os usos possíveis para as primitivas genéricas), o objetivo do desenvolvimento das aplicações não era mostrar a aplicabilidade de cada um dos tipos de eventos e primitivas, mas sim demonstrar de forma geral o uso de tais elementos na adaptação dos agentes.

Durante a descrição das aplicações, não foram salientadas questões referentes ao compartilhamento de adaptadores e à avaliação das adaptações realizadas, visto que os mecanismos que garantirão tais funcionalidades não se encontram completamente implementados (conforme discutido ao final do Capítulo 4). Posteriormente, pretende-se desenvolver aplicações mais abrangentes, com vários agentes interagindo e compartilhando adaptadores a fim de atingir seus objetivos de maneira mais satisfatória.

Por fim, cabe salientar os tipos de adaptações realizadas nas aplicações desenvolvidas. De acordo com Iman [Ima04][26], a adaptação de agentes inteligentes pode ser agrupada nas categorias adaptação interna, adaptação externa e adaptação completa. Na primeira aplicação descrita (a aplicação *SmartBroker*), tem-se um exemplo de adaptação completa. Nela, a adaptação na estrutura de um agente do tipo `BrokerAgent` (isto é, a alteração do conjunto de ações contidas em um plano) implica em modificações no comportamento apresentado por ele. Já as aplicações *ContextAwarePanel* e *SmartCars* são compostas por agentes que sofrem adaptações externas. Nessas duas aplicações, as estruturas dos agentes não são adaptativas, mas as suas ações externas refletem comportamento adaptativo.

Alguns autores (como [Spl03]) consideram que as adaptações externas nada mais são do que uma reação do agente e, portanto, não deveriam ser consideradas como um tipo de adaptação. Contudo, os autores deste trabalho se baseiam nas categorias de adaptação propostas por Imam e na definição de sistemas adaptativos ao contexto de Sitou e Spanfelner [Sit07]. Logo, com base nos trabalhos citados, há adaptação em todas as aplicações desenvolvidas.

[26] As categorias propostas por Iman estão detalhadas na Seção 2.1.3, onde se define o conceito de sistemas adaptativos ao contexto.

6. CONSIDERAÇÕES FINAIS

Considerando-se a importância da recuperação e do uso de informações contextuais no desenvolvimento de aplicações adaptativas, o presente trabalho apresentou o desenvolvimento de uma arquitetura para a criação de agentes adaptativos ao contexto que possibilita, entre outras coisas, o gerenciamento das informações contextuais disponíveis e a criação de adaptadores que podem ser compartilhados pelos agentes para a realização das mais diferentes adaptações. Permitir aos agentes analisar o ambiente em que estão inseridos, tendo também ciência de suas capacidades e tarefas em execução (autoconsciência), possibilita que essas entidades realizem suas adaptações de forma automática.

Como se pôde perceber ao longo do Capítulo 2, mesmo existindo várias pesquisas relacionadas ao uso de contexto e à criação de infra-estrutura específica para o desenvolvimento de aplicações sensíveis ao contexto, ainda há muito campo de pesquisa nesta área. A dificuldade de se trabalhar com adaptação ao contexto foi observada logo no início da execução da revisão sobre sistemas adaptativos ao contexto, uma vez que muitos dos artigos recuperados pelos mecanismos de busca descreviam apenas de forma superficial o processo de adaptação. Muitas vezes a “adaptação ao contexto” foi citada apenas para mostrar as contribuições de certas propostas (os autores não chegavam a descrever como a adaptação seria realizada) - ou seja, o processo de adaptação propriamente dito parece mais um desafio do que uma realidade.

De forma geral, a realização de adaptação consciente de contexto requer o estudo de várias questões, tais como: captura, modelagem e interpretação do contexto, disseminação das informações contextuais e a adaptação propriamente dita da aplicação. Todas essas questões são abordadas, em maior ou menor profundidade, na arquitetura proposta. Para a aquisição e representação do contexto, foi definido um modelo de referência para informações contextuais. Esse modelo foi usado para identificar e classificar as informações contextuais utilizadas em diferentes domínios de aplicação, auxiliando os desenvolvedores nas tarefas de identificação e seleção das informações contextuais relevantes. Uma das preocupações na criação do modelo foi respeitar as propriedades das informações contextuais, como é o caso da precisão. Não considerar a possível imprecisão das informações, seja por problemas na coleta ou pelas regras utilizadas para sua inferência, pode prejudicar o processo de adaptação do sistema e a satisfação do usuário final.

No entanto, sabe-se que construir aplicações adaptativas ao contexto significa mais do que simplesmente criar uma ontologia para representar o contexto e definir regras de inferência com base nesse modelo - o contexto é uma noção muito volátil e um sistema consciente de contexto deve estar pronto para lidar com esse fato através da implementação de mecanismos para gerenciar decisões e ações que podem ocorrem dinamicamente, periodicamente, ou sob demanda [Kar07].

De qualquer forma, a arquitetura proposta atenua boa parte das dificuldades encontradas no desenvolvimento de aplicações conscientes de contexto citadas por Dey em [Dey00]. Segundo o autor, há quatro razões que dificultam o uso de contexto: (i) as informações contextuais são adquiridas através de dispositivos não tradicionais, os quais não são muito conhecidos pelos desenvolvedores; (ii) as informações capturadas precisam ser trabalhadas para terem sentido dentro das aplicações; (iii) contexto é adquirido através de múltiplas fontes heterogêneas e distribuídas; e (iv) contexto é dinâmico, o que indica que as mudanças no ambiente devem ser detectadas em tempo real e as aplicações devem se adaptar a essas constantes mudanças. Todas as questões citadas por Dey são consideradas pelo módulo de gerenciamento de contexto contido na arquitetura.

Já para a adaptação dos agentes, foi definido o conceito de adaptador. Na arquitetura **K2**, os adaptadores são as estruturas responsáveis por encapsular o gatilho (eventos internos que, quando gerados, iniciam o processo de adaptação) e o efeito (conjunto de primitivas) de uma adaptação. Com a utilização dos adaptadores, as variantes não precisam ficar embutidas no núcleo dos componentes funcionais dos agentes. Isto evita a modificação direta do código dos componentes quando, por exemplo, é necessário alterar o efeito de determinada adaptação ou os eventos que a disparam. Se o comportamento adaptativo não fosse desvinculado do comportamento padrão do agente, seria necessário fazer uma reengenharia em toda a arquitetura do sistema a cada atualização no processo adaptativo.

Ainda em relação à adaptação dos agentes, a definição das primitivas genéricas, que mapeiam as operações de adaptação possíveis, atenua uma das limitações das pesquisas disponíveis na literatura, que é o conjunto limitado de tipos de adaptação. Considerando todos os possíveis usos das primitivas definidas, na arquitetura **K2** é possível fazer 39 tipos de adaptações, o que é um número bastante expressivo. A escassez de tipos de adaptações é citada por Gunasekera e co-autores, em [Gun08b].

Por fim, os exemplos desenvolvidos mostraram que a arquitetura é genérica o suficiente para ser utilizada em diferentes domínios de aplicação, como é o caso da bolsa de valores e dos veículos autônomos. Na verdade, a arquitetura se demonstrou aplicável tanto no desenvolvimento de aplicações adaptativas com informações de contexto mais tradicionais (por “tradicionais” entende-se as informações comumente utilizadas em aplicações da área da computação pervasiva, como localização de dispositivos e pessoas) quanto com fontes de contexto não tradicionais, como é o caso da cotação do dólar.

6.1. Trabalhos futuros

Vários trabalhos futuros estão sendo planejados para que a arquitetura **K2** seja finalizada e possa, então, ser disponibilizada para uso da comunidade acadêmica. Em termos conceituais, ainda é preciso estudar, principalmente, questões relacionadas à adaptação de agentes que buscam alcançar múltiplos objetivos simultaneamente e questões relacionadas à ativação também simultânea de diferentes (e muitas vezes incompatíveis) adaptadores. A forma como é feita a verificação da satisfação alcançada pelos agentes também requer mais estudo. Atualmente, tal verificação é um ponto de flexibilidade da arquitetura, ficando a cargo do desenvolvedor. Além disso, é preciso definir um meio de verificar dependências entre os componentes antes que sejam realizadas as adaptações.

Outros trabalhos futuros, agora relacionados à implementação da arquitetura, seriam a completa verificação e customização dos mecanismos oriundos do *framework* Ontowledge (os mecanismos já se encontram parcialmente implementados, sendo preciso realizar testes que talvez impliquem no desenvolvimento de melhorias), a exploração da integração da arquitetura em alguma plataforma para o desenvolvimento de SMAs disponível (como Jason [Bor07], Jade [Til11] ou SemantiCore [Blo07]), e a finalização do *plugin* para o Eclipse, cuja implementação já se encontra em desenvolvimento.

Depois que todas as funcionalidades se encontrarem desenvolvidas e testadas, deverão ser realizados experimentos para verificar o tempo de resposta obtido pelos agentes adaptativos. Segundo Ayed e co-autores [Aye08], os atrasos adicionados pela adaptação não deveriam ser percebidos pelos usuários. Também, como uma das características da arquitetura desenvolvida é a separação do comportamento adaptativo do comportamento padrão do agente, estuda-se a possibilidade de realizar medições

relacionadas ao espalhamento e embaralhamento do código *(scattering e tangling)*, o que deverá evidenciar os ganhos obtidos com a modularização do comportamento adaptativo.

REFERÊNCIAS BIBLIOGRÁFICAS

[Abo00] Abowd, G. D.; Mynatt, E. D. “Charting past, present, and future research in ubiquitous computing”. *ACM Transactions on Computer-Human Interaction*, vol. 7-1, 2000, pp. 29-58.

[Ali10] Ali, M.; Babar, M.; Chen, L.; Stol, J. “A Systematic Review of Comparative Evidence of Aspect-Oriented Programming”. *Information and Software Technology*, vol. 52, 2010, pp. 871-887.

[Ama05] Amara-Hachmi, N. “A Framework for Building Adaptive Mobile Agents”. In: AAMAS, 2005, pp. 1369-1372.

[Ama06] Amara-Hachmi, N. “An Ontology-based Model for Mobile Agents Adaptation in Pervasive Environments”. In: AICCSA, 2006, pp. 1106-1109.

[Amo06] Amor, M.; Fuentes, L. “Madl: a Description Language for Component and Aspect-oriented Agent Architectures”. In: Multiagent Systems and Software Architecture, 2006, pp. 136-143.

[Amo09] Amor, M.; Fuentes, L. “Malaca: A component and aspect-oriented agent architecture”. *Information and Software Technology*, vol. 51-6, Jun 2009, pp. 1052-1065.

[Aru10] Arulraj, J. “Adaptive Agent Generation Using Machine Learning For Dynamic Difficulty Adjustment”. In: ICCCT, 2010, pp. 746-751.

[Awa09] Awang, N. H.; Kadir, W. M.; Shahibuddin, S. "Comparative Evaluation of the State-of-the Art on Approaches to Software Adaptation". In: Fourth International Conference on Software Engineering Advances, 2009, pp. 425-430.

[Aye08] Ayed, D.; Taconetb, C.; Bernardb, G.; Berbers, Y. “CADeComp: Context-aware deployment of component-based applications”. *Journal of Network and Computer Applications*, vol. 31-3, Ago 2008, pp. 224-257.

[Bar10] Barrett, S.; Taylor, M.; Stone, P. “Transfer Learning for Reinforcement Learning on a Physical Robot”. In: Ninth International Conference on Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop (AAMAS - ALA), 2010, pp. 24-29.

[Bey09] Beydoun, G.; Low, G.; Henderson-Sellers, B.; Mouratidis, H.; Gomez-Sanz, J.; Pavon, J.; Gonzalez-Perez, C. “FAML: A Generic Metamodel for MAS Development”. *IEEE Transactions on Software Engineering*, vol. 35-6, Nov-Dez 2009, pp. 841-863.

[Blo07] Blois, M.; Escobar, M.; Choren, R. “Using Agents and Ontologies for Application Development on the Semantic Web”. *Journal of the Brazilian Computer Society*, vol. 13-2, Jun 2007, pp. 35-44.

[Bor07] Bordini, R.; Hubner, J.; Wooldridge, M. “Programming Multi-Agent Systems in AgentSpeak Using Jason”. John Wiley & Sons, 2007, 273p.

[Car07] Carton, A.; Senart, A.; Cahill, V. “Aspect-Oriented Model-Driven Development for Mobile Context-Aware Computing”. In: First Workshop on Software Engineering of Pervasive Computing Applications, Systems and Environments, 2007, 5p.

[Cha03] Chavez, C.; Lucena, C. “A Theory of Aspects for Aspect-Oriented Software Development”. In: Simpósio Brasileiro de Engenharia de Software, 2003, 16p.

[Cha06] Chaari, T.; Ejigu, D.; Laforest, F.; Scuturici, V. “Modeling and Using Context in Adapting Applications to Pervasive Environments”. In: 2006 ACS/IEEE International Conference on Pervasive Services, 2006, pp. 111-120.

[Che00] Chen, G.; Kotz, D. “A Survey Of Context-Aware Mobile Computing Research”. Technical Report, Dartmouth College, 2000, 16p.

[Che03] Chen, H.; Finin, T.; Joshi, A. “An ontology for context-aware pervasive computing environments”. *The Knowledge Engineering Review*, vol. 18-3, Set 2003, pp. 197-207.

[Cor05] Cortese, G.; Davide, F.; Lunghi, M. “Context Awareness for Physical Service Environments”. *Ambient Intelligence*, vol. 6, Jan 2005, pp. 71-97.

[Dey00] Dey, A. “Providing Architectural Support for Building Context-Aware Applications”. PhD Thesis, Georgia Institute of Technology, 2000, 188p.

[Dey01] Dey, A.; Abowd, G.; Salber, D. “A conceptual framework and a toolkit for supporting the rapid prototyping context-aware applications”. *Human-Computer Interaction*, vol. 16-2, Dez 2001, pp. 97-166.

[Dey98] Dey, A. “Context-aware computing: The CyberDesk project”. In: AAAI 1998 Spring Symposium on Intelligent Environments, 1998, pp. 51-54.

[Dey99] Dey, A.; Abowd, G. “Towards a better understanding of context and context-awareness”. Technical Report, Georgia Institute of Technology, 1999, 12p.

[Dou04] Dourish, P. “What we talk about when we talk about context”. *Personal and Ubiquitous Computing*, vol. 8-1, Fev 2004, pp. 19-30.

[Ecl11a] Eclipse.org. “IDE Eclipse”. Capturado em: http://www.eclipse.org/, Fevereiro 2011.

[Ecl11b] Eclipse.org. “The AspectJ Project”. Capturado em: http://www.eclipse.org/aspectj/, Fevereiro 2011.

[Ecl11c] Eclipse.org. “Introduction to AspectJ”. Capturado em: http://eclipse.org/aspectj/doc/released/progguide/starting-aspectj.html, Fevereiro 2011.

[Ecl11d] Eclipse.org. “The AspectJ™ Development Environment Guide”. Capturado em: http://www.eclipse.org/aspectj/doc/released/devguide/ltw.html, Fevereiro 2011.

[Efs04] Efstratiou, C. “Coordinated Adaptation for Adaptive Context-aware Applications”. PhD Thesis, Lancaster University, 2004, 200p.

[Eld06] Elder, A. “Aprenda a operar no mercado de ações”. Rio de Janeiro: Campus, 2006, 11ª edição, 317p.

[Eli06] Elian, D. “Estudo sobre Adaptação de Agentes de Software”. Trabalho Individual, Faculdade de Informática - PUCRS, 2006, 53p.

[Eli08] Elian, D. “Um Framework para Agentes Adaptativos na Web Semântica”. Dissertação de mestrado, Faculdade de Informática - PUCRS, 2008, 135p.

[Fer99] Ferreira, A. “Novo Aurélio século XXI: o dicionário da língua portuguesa”. Rio de Janeiro: Nova Fronteira, 1999, 2128p.

[Gan04] Gandon, F.; Sadeh, N. “Semantic Web Technologies to Reconcile Privacy and Context Awareness”. *Web Semantics Journal*, vol. 1-3, Abr 2004, 36p.

[Gar08] Garcia, A.; Lucena, C. “Taming Heterogeneous Agent Architectures - Using aspect-oriented techniques to construct high-quality multi-agent systems”. *Communications of the ACM*, vol. 51-1, Maio 2008, pp. 75-81.

[Gon07] González, S.; Mens, K.; Heymans, P. “Highly dynamic behaviour adaptability through prototypes with subjective multimethods”. In: Symposium on Dynamic languages, 2007, pp. 77-88.

[Gra07] Grassi, V.; Sindico, A. “Towards model driven design of service-based context-aware applications”. In: International workshop on Engineering of software services for pervasive environments, 2007, pp. 69-74.

[Gun08a] Gunasekera, K.; Zaslavsky, A.; Krishnaswamy, S.; Loke, S. “VERSAG: Context-Aware Adaptive Mobile Agents for the Semantic Web”. In: 32nd Annual IEEE International Computer Software and Applications Conference, 2008, pp. 521-522.

[Gun08b] Gunasekera, K.; Zaslavsky, A.; Loke, S.; Krishnaswamy, S. "Context Driven Compositional Adaptation of Mobile Agents". In: International Conference on Mobile Data Management Workshops, 2008, pp. 201-208.

[Gun09a] Gunasekera, K.; Loke, S.; Zaslavsky, A.; Krishnaswamy, S. "Runtime Adaptation of Multiagent Systems for Ubiquitous Environments". In: International Conference on Web Intelligence and Intelligent Agent Technology, 2009, pp. 486-490.

[Gun09b] Gunasekera, K.; Krishnaswamy, S.; Loke, S.; Zaslavsky, A. “Runtime Efficiency of Adaptive Mobile Software Agents in Pervasive Computing Environments”. In: ICPS’09, 2009, pp. 123-131.

[Han07] Han, S.; Song, S.; Youn, H. “Dynamic Software Adaptation with Dependence Analysis for Multi-Agent Platform”. In: Fifth International Conference on Computational Science and Applications, 2007, pp. 185-191.

[Har05] Harroud, H.; Karmouch, A. “A policy based context-aware agent framework to support users mobility”. In: AICT/SAPIR/ELETE, 2005, pp. 177-182.

[Has05] Haseloff, S. “Context Awareness in Information Logistics”. PhD Thesis, Technical University of Berlin, 2005, 263p.

[Hay96] Haynes, T.; Sen, I. “Adaptation using cases in cooperative groups”. In: Working Notes of the AAAI-96 Workshop on Intelligent Adaptive Agents, 1996, 9p.

[Hef03] Heflin, J. “Web ontology language (OWL) use cases and requirements”. Capturado em: http://www.w3.org/TR/2004/REC-webont-req-20040210/, Maio 2003.

[Hen02] Henricksen, H.; Indulska, J.; Rakotonirainy, A. “Modeling context information in pervasive computing systems”. In: First International Conference on Pervasive Computing, 2002, pp. 167-180.

[Hu08] Hu, P.; Indulska, J.; Robinson, R. “An Autonomic Context Management System for Pervasive Computing”. In: Sixth Annual IEEE International Conference on Pervasive Computing and Communications, 2008, pp. 213-223.

[Ima04] Imam, I. “Adaptive applications of intelligent agents”. In: International Symposium on Computers and Communications, 2004, pp. 7-12.

[Jar05] Jars, I.; Kabachi, N.; Lamure, M.; “Adaptive Agent’s Integration in a New Environment: Interactions as a Source of Learning”. In: AAMAS, 2005, pp. 1103-1104.

[Jia03] Jiao, W.; Zhou, M.; Wang, Q. “Formal framework for adaptive multi-agent systems”. In: IAT, 2003, pp.442-445.

[Kap08] Kapitsaki, G.; Venieris, I. “PCP: privacy-aware context profile towards context-aware application development”. In: 10th International Conference on Information Integration and Web-based Applications & Services, 2008, pp. 104-110.

[Kar07] Kara, N.; Dragoi, O. “Reasoning with Contextual Data in Telehealth Applications”. In: Third IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, 2007, 8p.

[Kha07] Khan, M.; Reichle, R.; Geihs, K. “Applying Architectural Constraints in the Modelling of Self-adaptive Component-based Applications”. In: 1st Workshop on Model-driven Software Adaptation, 2007, pp. 13-22.

[Khe05] Khedr, M. "Enhancing Applicability of Context-Aware Systems Using Agent-Based Hybrid Inference Approach". In: IEEE VTC Spring, 2005, pp. 2785- 2789.

[Kic97] Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J.; Irwin, J. “Aspect-Oriented Programming”. In: European Conference on Object-Oriented Programming, 1997, 14p.

[Kle08] Klein, C.; Schmid, R.; Leuxner, C.; Sitou, W.; Spanfelner, B. “A Survey of Context Adaptation in Autonomic Computing”. In: Fourth International Conference on Autonomic and Autonomous Systems, 2008, pp. 106-111.

[Kru07] Krummenacher, R.; Strang, T. “Ontology-Based Context Modeling”. Capturado em http://www.sti-innsbruck.at/fileadmin/documents/CAPS_2349720_krummenacher.pdf, Maio 2007.

[Lac09] Lacouture, J.; Aniorté, P. "Self-Adaptation of Semantic Services Based on a Component/Agent Approach: Application to e-Training". In: ICAIS-International Conference on Adaptive and Intelligent Systems, 2009, pp. 21-27.

[Lee07] Lee, J.; Seoa, D.; Rheea, G. “Visualization and interaction of pervasive services using context-aware augmented reality”. *Expert Systems with Applications*, vol. 35-4, Nov 2007, pp. 1873-1882.

[Lee10] Lee, J. Lin, K-J. “A context management framework for real-time SOA”. In: 8th IEEE International Conference on Industrial Informatics, 2010, pp. 559-564.

[Lem07a] Lemke, A. P.; Blois, M.; Choren, R. “A Knowledge Management Framework for Semantic Multi-Agent Systems”. In: Seventeenth International Workshop on Agent-Oriented Information Systems, 2007, pp. 683-696.

[Lem07b] Lemke, A. P. “Uso de Informações Contextuais em Sistemas Computacionais”. Trabalho Individual 3, Faculdade de Informática - PUCRS, 2007, 44p.

[Lem07c] Lemke, A. P.; Blois, M. “Representação de Informações Contextuais em Sistemas Computacionais”. In: I Workshop on Pervasive and Ubiquitous Computing, 2007, 6p.

[Lem09] Lemke, A. P.; Blois, M. “Using Knowledge Objects to Exchange Knowledge in a MAS platform”. In: Software Engineering & Knowledge Engineering, 2009, pp. 206-211.

[Lem10a] Lemke, A. P. “Uma Arquitetura para a Adaptação de Agentes de Software ao Contexto”. Proposta de Tese, Faculdade de Informática - PUCRS, 2010, 111p.

[Lem10b] Lemke, A.P.; Blois, M. “Adaptação de Agentes de Software ao Contexto com o uso de Aspectos”. In: AutoSoft 2010, 2010, pp. 81-90.

[Ler03] Lerman, K.; Galstyan, A. “Agent Memory and Adaptation in Multi-Agent Systems”. In: AAMAS, 2003, pp. 797-803.

[Li06] Li, W. "Towards a Person-Centric Context-Aware System", Licentiate Thesis, Stockholm University, 2006, 92p.

[Lob04] Lobato, C.; Garcia, A.; Romanovsky, A.; SantAnna, C.; Kulesza, U.; Lucena, C. “Mobility as an Aspect: The AspectM Framework”. In: WASP'04, 2004, pp. 116-123.

[Man07] Mani, A.; Sundaram, H. “Modeling User Context with Applications to Media Retrieval”. *Multimedia Systems*, vol. 12, Mar 2007, pp. 339-353.

[Mar06] Marín, C.; Mehandjiev, N. “A Classification Framework of Adaptation in Multi-Agent Systems”. In: CIA 2006, 2006, pp. 198-212.

[McK04] McKinley, P.; Sadjadi, S.; Kasten, E.; Cheng, B. “Composing Adaptive Software”. *Computer*, vol. 37-7, Jul 2004, pp. 56-64.

[Men06] Menon, V. “Adaptation in Pervasive Computing Environments: A Multi-agent approach”. In: International Symposium on Ad Hoc and Ubiquitous Computing, Surathkal, 2006, pp. 184-185.

[Men08] Meng, D.; Poslad, S. “A reflective context-aware system for spatial routing applications”. In: 6th international workshop on Middleware for pervasive and ad-hoc computing, 2008, pp. 54-59.

[Mik06] Mikalsen, M.; Floch, J.; Paspallis, N.; Papadopoulos, G.; Ruiz, P. “Putting Context in Context: The Role and Design of Context Management in a Mobility and Adaptation Enabling Middleware”. In: 7th International Conference on Mobile Data Management, 2006, pp. 76-84.

[Moh06] Mohomed, I.; Cai, J.; Chavoshi, S.; Lara, E. “Context-aware interactive content adaptation”. In: 4th international conference on Mobile systems, applications and services, 2006, pp. 42-55.

[Oat06] Oates, B. “Researching Information Systems and Computing”. Sage Publications Ltda, 2006, 341p.

[OMG00] OMG - Object Management Group. “Agent Technology – the green paper – version 1.0”. Capturado em: http://www.jamesodell.com/ec2000-08-01.pdf, Agosto 2005.

[Ora11] Oracle. “Java”. Capturado em: http://www.java.com/, Fevereiro 2011.

[Oza04] Ozawa, S.; Tsumori, K. “A memory-based neural network model for efficient adaptation to dynamic environments”. In: EEE International Conference on Fuzzy Systems, 2004, pp. 437- 442.

[Par72] Parnas, D. “On the criteria to be used in decomposing systems into modules”. *Communications of ACM*, vol. 15-12, Dez 1972, pp. 1053-1058.

[Pen03] Pender, T. “UML Bible”. Wiley, 2003, 984 p.

[Per00] Perry, W. "Effective Methods for Software Testing". Wiley, 2000, 2ª edition, 832p.

[Piv04] Piveta, E.; Garcia, V.; Zancanella, L.; Prado, A. “Termos em português para Desenvolvimento de Software Orientado”. In: WASP’04, 2004, pp. 155-156.

[Pla07] Platon, E.; Mamei, M.; Sabouret, N.; Honiden, S.; Parunak, H. “Mechanisms for Environments in MAS: Survey and Opportunities”. *Journal Autonomous Agents and Multi-Agent Systems*, vol. 14-1, 2007, pp. 31-47.

[QFr11] Q-Free. “CVIS - Cooperative vehicle-infrastructure systems”. Capturado em: http://www.cvisproject.org/download/qfree_CommCalmCvis_aalborgbrochure.pdf, Fevereiro 2011.

[Ran03] Ranganathan, A.; Campbell, R. “A Middleware for Context-Aware Agents in Ubiquitous Computing Environments”. In: Middleware, 2003, pp. 143-161.

[Ras04] Rashid, A.; La, L. "Adaptation as an Aspect in Pervasive Computing". In: Workshop on Building Software for Pervasive Computing, 2004, 6p.

[Rib02] Ribeiro, M. “Web Life: Uma arquitetura para a implementação de sistemas multi-agentes para a Web”. Tese de Doutorado, Pontifícia Universidade Católica do Rio de Janeiro, 2002, 204p.

[Roc05] Rocha, R.; Endler, M. “Evolutionary and Efficient Context Management in Heterogeneous Environments”. In: 3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing, 2005, pp. 1-7.

[Rya98] Ryan, N.; Pascoe, J.; Morse, D. “Enhanced reality fieldwork: the context-aware archaeological assistant”. In: Computer Applications and Quantitative Methods in Archaeology, 1998, pp. 269-274.

[Sar04] Sardinha, J.; Garcia, A.; Lucena, C.; Milidiú, R. “The Agent Learning Pattern”. In: Fourth Latin American Conference on Pattern Languages of Programming, 2004, 9p.

[Sat01] Satyanarayanan, M. “Pervasive computing: vision and challenges”. *IEEE Personal Communication*, vol. 8-4, Ago 2001, pp.10-17.

[Sil08] Silva, V.; Choren R.; Lucena, C. "MAS-ML: A Multi-Agent System Modelling Language". *International Journal of Agent-Oriented Software Engineering (Special Issue on Modeling Languages for Agent Systems)*, vol. 2-4, 2008, pp. 382-421.

[Sim08] Simpkins, C.; Bhat, S.; Isbell, C.; Mateas, M. “Towards adaptive programming: integrating reinforcement learning into a programming language”. In: 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications, 2008, pp. 603-614.

[Sin08] Sindico, A.; Bartolomeo, G.; Grassi, V.; Salsano, S. “Design and development of a context oriented language for middleware based applications”. In: 2008 Workshop on Next generation Aspect Oriented Middleware, 2008, pp. 1-5.

[Sit07] Sitou, W.; Spanfelner, B. “Towards Requirements Engineering for Context Adaptive Systems”. In: 31st Annual International Computer Software and Applications Conference, 2007, pp. 593-600.

[Slp03] Splunter, S. van; Wijngaards, N.; Brazier, F. “Structuring Agents for Adaptation”. *Adaptive Agents and Multi-Agent Systems*, vol. 2636, 2003, pp. 174-186.

[Str04] Strang, T.; Linnhoff-Popien, C. “A context modelling survey”. In: First International Workshop on Advanced Context Modelling, Reasoning and Management, 2004, 8p.

[Til11] TILAB. “JADE - Java Agent DEvelopment Framework”. Capturado em: http://jade.tilab.com/, Fevereiro 2011.

[Tot07] Tóth, M.; Belicak, M. “Dynamic Restructuralization of Software Systems Using Aspect-Oriented Programing”. In: 5th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence and Informatics, 2007, pp. 457-468.

[Vay05] Vaysse, G.; André, F.; Buisson, J. “Using Aspects for Integrating a Middleware for Dynamic Adaptation”. In: 1st Workshop on Aspect Oriented Middleware Development, 2005, pp. 1-6.

[W3C11] W3C. “OWL - Web Ontology Language”. Capturado em: http://www.w3.org/2004/OWL/, Fevereiro 2011.

[Web04] Weber, R.; Wu, D. “Knowledge Management for Computational Intelligence Systems”. In: Proceedings of Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE 2004), 2004, pp. 116-125.

[Wei91] Weiser, M. “The computer for the twenty-first century”. *Scientific American*, vol. 265-3, Set 1991, pp. 94-104.

[Wei96] Weiss, G. “Adaptation and learning in multi-agent systems: Some Remarks and a Bibliography”. *Lecture Notes in Artificial Intelligence*, vol. 1042, 1996, pp. 1-21.

[Wei99] Weiss, G. “Multiagent systems: a modern approach to distributed artificial intelligence”. Cambridge: The MIT Press, 1999, 619p.

[Wey08] Weyns, D.; Boucké, N.; Holvoet, T. “A field-based versus a protocol-based approach for adaptive task assignment”. *Autonomous Agents and Multi-Agent Systems*, vol. 17-2, Out 2008, pp. 288-319.

[Win02] Winograd, T. “Architecture for Context”. *Human Computer Interaction*, vol. 16, 2002, pp. 401-419.

APÊNDICE A. PRINCÍPIOS DA PROGRAMAÇÃO ORIENTADA A ASPECTOS

A Programação Orientada a Aspectos (POA), cuja definição e princípios foram propostos por Kiczales e co-autores em [Kic97], pode ser definida como uma forma de organizar o código da aplicação de acordo com a sua importância (*separation of concerns*[27]), modularizando os ditos *interesses transversais* (*crosscutting concerns*). O código que implementa esses interesses (ou o *aspecto*) é desenvolvido de forma separada das outras partes do sistema [McK04]. De acordo com Ali e co-autores [Ali10], “no centro do paradigma de desenvolvimento orientado a aspectos está a idéia de *concern*”.

De acordo com Rashid e La [Ras04], técnicas de POA devem fornecer abstrações e construtores para modularizar tais propriedades transversais, permitindo a especificação de seus relacionamentos de composição e a sua integração com outros elementos do sistema, ou seja, deve ser provido um mecanismo de composição que permita “tecer” os aspectos nos componentes do sistema – processo chamado de *weaving*. A Figura A.1 ilustra, de maneira geral, o processo de *weaving* de aspectos. Os relacionamentos de composição são especificados utilizando um modelo de pontos de junção (*join points*), o qual identifica pontos bem definidos dentro de um sistema onde um aspecto pode ser integrado [Kic97, Ras04].

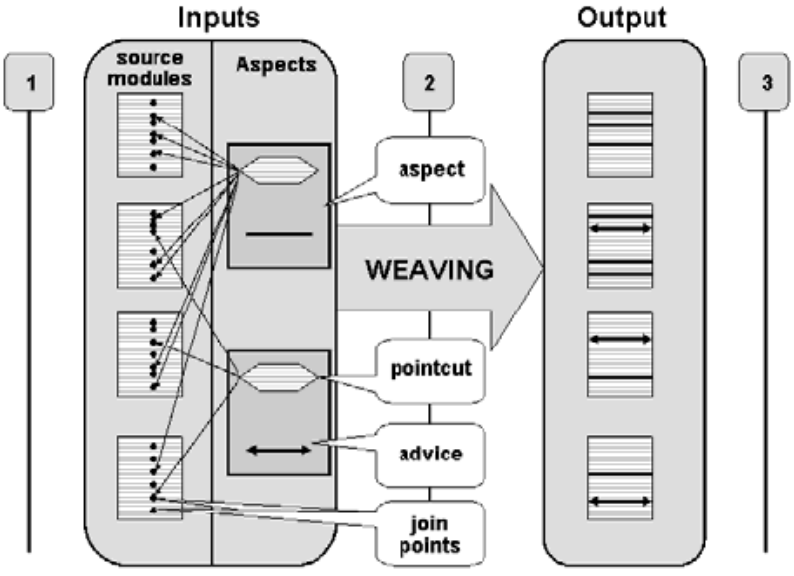

Figura A.1 - Processo de *weaving* de aspectos [Tot07].

[27] Cabe salientar que o conceito por trás da separação de interesses é anterior a POA – muitos autores atribuem a Parnas [Par72] a sua autoria. De acordo com McKinley e co-autores [McK04], atualmente, a POA parece ser a abordagem mais utilizada para prover a separação de interesses.

Existem, basicamente, dois tipos de *weaving* de aspectos [Tot07]: o *weaving* estático e o *weaving* dinâmico. O *weaving* estático é caracterizado pela modificação do código antes de sua execução (em tempo de compilação ou carregamento) e também pode ser entendido como uma forma de transformação para refatoração de código. Já o *weaving* dinâmico, que ocorre em tempo de execução, compreende a modificação da semântica do programa executado. Ou seja, no *weaving* dinâmico são modificadas funcionalidades no nível de módulos individuais (antes do carregamento dos módulos dinâmicos) ou é modificado o ambiente de execução, através da alteração da semântica de certas instruções [Tot07]. De acordo com Amor e Fuentes [Amo09], o *weaving* dinâmico é o mais indicado quando existem requisitos de adaptabilidade para o sistema.

Várias linguagens orientadas a aspectos já foram desenvolvidas[28]. A AspectJ [Ecl11b], apresentada em seu *site* oficial como uma extensão orientada a aspectos da linguagem de programação Java que permite uma completa modularização de interesses transversais, é uma das implementações de POA mais evoluída e popular [Tot07]. AspectJ adiciona à linguagem Java o conceito de ponto de junção[29] (*join point*) e alguns novos construtores, que são: ponto de atuação (*pointcut*), sugestão (*advice*), declarações inter-tipo (*inter-type declarations*) e aspecto (*aspect*). Pontos de atuação e sugestões afetam dinamicamente a execução do programa, enquanto declarações inter-tipo afetam estaticamente a sua hierarquia de classes. A seguir, são dadas definições e exemplos dos conceitos e construtores definidos pela AspectJ.

- Ponto de junção: ponto bem definido ao longo da execução do programa [Ecl11c]. Em [Cha03], um ponto de junção é descrito como “elemento relacionado à estrutura ou à execução de um componente de um programa que é referenciado e possivelmente afetado por um aspecto”. Como exemplos de pontos de junção citam-se: execução e chamada de métodos, criação de objetos, execução de sugestões, ocorrência de exceções, entre outros.
- Ponto de atuação: distingue certos pontos de junção e valores para eles [Ecl11c]. Também pode se entendido com uma descrição lógica de um conjunto de pontos de junção. Para cada tipo de ponto de junção definido, é especificado um designador para ser utilizado no ponto de atuação. Por

[28] Veja em http://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_orientada_a_aspecto a lista de implementações orientadas a aspectos para diferentes linguagens de programação.
[29] A tradução dos termos encontrados na literatura de POA para a língua portuguesa está sendo feita de acordo com o proposto por Piveta e co-autores em [Piv04].

exemplo, para representar o ponto de junção “chamada de método” foi definido o designador “*call*”. Assim, o ponto de atuação “`call (void Point.setX (int))`” distingue todos os pontos de junção que são chamadas para o método `void setX (int)` de objetos da classe `Point`.

- Sugestão: conecta um ponto de atuação a um trecho de código que é executado quando os pontos de junção são alcançados [Ecl11c]. As sugestões podem ser inseridas em três momentos: antes de o programa prosseguir com a execução de um ponto de junção (*before advice*); depois de o programa executar um ponto de junção (*after advice*); ou no lugar de um ponto de junção - quando o ponto de junção é atingido, o controle de sua execução passa a ser da própria sugestão (*around advice*). No caso de *around advice*, o programa pode prosseguir ou não no ponto de junção (a sugestão intercepta e assume o controle da execução do ponto de junção).
- Declarações inter-tipo: permitem a modificação da estrutura estática do programa, ou seja, membros de classes e relações entre as classes [Ecl11c].
- Aspectos: encapsulam os novos construtores descritos, comportando-se de maneira semelhante a classes Java [Ecl11c].

No AspectJ (versão 5), o *weaving* de aspectos pode acontecer em três diferentes tempos: de compilação, de pós-compilação e em tempo de carregamento [Ecl11d]. O *weaving* em tempo de compilação é considerado o mais simples e consiste na compilação do código-fonte da aplicação gerando como resultado arquivos `.class` já combinados com os aspectos. O *weaving* em pós-compilação (também chamado de *weaving* binário) é utilizado para combinar arquivos `.class` e arquivos `.jar`. Por último, o *weaving* em tempo de carregamento é também um tipo de *weaving* binário, mas prevê a existência de um ou mais “carregadores de classes”, que carregam arquivos `.class` e determinam os arquivos `.class` que serão utilizados pela máquina virtual Java. De acordo com a documentação do AspectJ, não há suporte explícito para *weaving* em tempo de execução (*weaving* de classes que já foram definidas para a máquina virtual sem a necessidade de recarregá-las), mas utilizando padrões de codificação é possível habilitar e desabilitar dinamicamente as sugestões.

APÊNDICE B. ONTOLOGIA PARA A REPRESENTAÇÃO DE INFORMAÇÕES CONTEXTUAIS

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
    <!ENTITY semanticore "http://semanticore.pucrs.br#" >
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF xmlns="http://semanticore.pucrs.br#"
     xml:base="http://semanticore.pucrs.br"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:semanticore="http://semanticore.pucrs.br#">
    <owl:Ontology rdf:about=""/>

    <!--

///////////////////////////////////////////////////////////////////////////////
///////
    //
    // Object Properties
    //

///////////////////////////////////////////////////////////////////////////////
///////
     -->

    <!-- http://semanticore.pucrs.br#context_of -->

    <owl:ObjectProperty rdf:about="#context_of">
        <rdfs:domain rdf:resource="#Context"/>
        <rdfs:range rdf:resource="#Entity"/>
    </owl:ObjectProperty>


    <!-- http://semanticore.pucrs.br#deduced_from -->

    <owl:ObjectProperty rdf:about="#deduced_from">
        <rdfs:range rdf:resource="#Contextual_Information"/>
        <rdfs:domain rdf:resource="#Inferred_Information"/>
    </owl:ObjectProperty>


    <!-- http://semanticore.pucrs.br#generated_by -->

    <owl:ObjectProperty rdf:about="#generated_by">
        <rdfs:range rdf:resource="#Information_Source"/>
        <rdfs:domain rdf:resource="#Sensed_Information"/>
    </owl:ObjectProperty>
```

```
    <!-- http://semanticore.pucrs.br#generates -->

    <owl:ObjectProperty rdf:about="#generates">
        <rdfs:domain rdf:resource="#Information_Source"/>
        <rdfs:range rdf:resource="#Sensed_Information"/>
        <owl:inverseOf rdf:resource="#generated_by"/>
    </owl:ObjectProperty>


    <!-- http://semanticore.pucrs.br#has_Information_Source -->

    <owl:ObjectProperty rdf:about="#has_Information_Source">
        <rdfs:domain rdf:resource="#Entity"/>
        <rdfs:range rdf:resource="#Information_Source"/>
    </owl:ObjectProperty>


    <!-- http://semanticore.pucrs.br#has_Non_Temporal_Information -->

    <owl:ObjectProperty rdf:about="#has_Non_Temporal_Information">
        <rdfs:domain rdf:resource="#Entity"/>
        <rdfs:range rdf:resource="#Non_Temporal_Information"/>
        <owl:inverseOf rdf:resource="#non_Temporal_Information_of"/>
    </owl:ObjectProperty>


    <!-- http://semanticore.pucrs.br#has_context -->

    <owl:ObjectProperty rdf:about="#has_context">
        <rdfs:range rdf:resource="#Context"/>
        <rdfs:domain rdf:resource="#Entity"/>
        <owl:inverseOf rdf:resource="#context_of"/>
    </owl:ObjectProperty>


    <!-- http://semanticore.pucrs.br#has_relationship_with -->

    <owl:ObjectProperty rdf:about="#has_relationship_with">
        <rdf:type rdf:resource="&owl;SymmetricProperty"/>
        <owl:inverseOf rdf:resource="#has_relationship_with"/>
    </owl:ObjectProperty>


    <!-- http://semanticore.pucrs.br#informed_by -->

    <owl:ObjectProperty rdf:about="#informed_by">
        <rdfs:range rdf:resource="#Entity"/>
        <rdfs:domain rdf:resource="#Explicit_Information"/>
    </owl:ObjectProperty>



    <!-- http://semanticore.pucrs.br#informs -->

    <owl:ObjectProperty rdf:about="#informs">
        <rdfs:domain rdf:resource="#Entity"/>
        <rdfs:range rdf:resource="#Explicit_Information"/>
        <owl:inverseOf rdf:resource="#informed_by"/>
    </owl:ObjectProperty>


    <!-- http://semanticore.pucrs.br#is_composed_of -->

    <owl:ObjectProperty rdf:about="#is_composed_of">
```

```
        <rdfs:domain rdf:resource="#Context"/>
        <rdfs:range rdf:resource="#Contextual_Information"/>
    </owl:ObjectProperty>


    <!-- http://semanticore.pucrs.br#is_created_based_on -->

    <owl:ObjectProperty rdf:about="#is_created_based_on">
        <rdfs:domain rdf:resource="#Context"/>
        <rdfs:range rdf:resource="#Context"/>
    </owl:ObjectProperty>


    <!-- http://semanticore.pucrs.br#non_Temporal_Information_of -->

    <owl:ObjectProperty rdf:about="#non_Temporal_Information_of">
        <rdfs:range rdf:resource="#Entity"/>
        <rdfs:domain rdf:resource="#Non_Temporal_Information"/>
    </owl:ObjectProperty>


    <!--

///////////////////////////////////////////////////////////////////////////////
///////
    //
    // Data properties
    //

///////////////////////////////////////////////////////////////////////////////
///////
     -->

    <!-- http://semanticore.pucrs.br#accuracy -->

    <owl:DatatypeProperty rdf:about="#accuracy">
        <rdfs:domain rdf:resource="#Temporal_Information"/>
        <rdfs:range rdf:resource="&xsd;float"/>
    </owl:DatatypeProperty>


    <!-- http://semanticore.pucrs.br#factor_confidence -->

    <owl:DatatypeProperty rdf:about="#factor_confidence">
        <rdfs:domain rdf:resource="#Temporal_Information"/>
        <rdfs:range rdf:resource="&xsd;double"/>
    </owl:DatatypeProperty>


    <!-- http://semanticore.pucrs.br#last_updated_value -->

    <owl:DatatypeProperty rdf:about="#last_updated_value">
        <rdfs:domain rdf:resource="#Contextual_Information"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>


    <!-- http://semanticore.pucrs.br#source_name -->

    <owl:DatatypeProperty rdf:about="#source_name">
        <rdfs:domain rdf:resource="#Information_Source"/>
        <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
```

```
    <!-- http://semanticore.pucrs.br#timestamp -->

    <owl:DatatypeProperty rdf:about="#timestamp">
        <rdfs:domain rdf:resource="#Temporal_Information"/>
        <rdfs:range rdf:resource="&xsd;int"/>
    </owl:DatatypeProperty>


    <!--

///////////////////////////////////////////////////////////////////////////////
///////
    //
    // Classes
    //

///////////////////////////////////////////////////////////////////////////////
///////
     -->

    <!-- http://semanticore.pucrs.br#Context -->

    <owl:Class rdf:about="#Context">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#context_of"/>
                <owl:onClass rdf:resource="#Entity"/>
                <owl:qualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:qualifiedCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>


    <!-- http://semanticore.pucrs.br#Contextual_Information -->

    <owl:Class rdf:about="#Contextual_Information"/>


    <!-- http://semanticore.pucrs.br#Entity -->

    <owl:Class rdf:about="#Entity"/>


    <!-- http://semanticore.pucrs.br#Explicit_Information -->

    <owl:Class rdf:about="#Explicit_Information">
        <rdfs:subClassOf rdf:resource="#Temporal_Information"/>
    </owl:Class>


    <!-- http://semanticore.pucrs.br#Inferred_Information -->

    <owl:Class rdf:about="#Inferred_Information">
        <rdfs:subClassOf rdf:resource="#Temporal_Information"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#factor_confidence"/>
                <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
            </owl:Restriction>
```

```
        </rdfs:subClassOf>
    </owl:Class>


    <!-- http://semanticore.pucrs.br#Information_Source -->

    <owl:Class rdf:about="#Information_Source"/>


    <!-- http://semanticore.pucrs.br#Non_Temporal_Information -->

    <owl:Class rdf:about="#Non_Temporal_Information">
        <rdfs:subClassOf rdf:resource="#Contextual_Information"/>
    </owl:Class>


    <!-- http://semanticore.pucrs.br#Sensed_Information -->

    <owl:Class rdf:about="#Sensed_Information">
        <rdfs:subClassOf rdf:resource="#Temporal_Information"/>
    </owl:Class>


    <!-- http://semanticore.pucrs.br#Temporal_Information -->

    <owl:Class rdf:about="#Temporal_Information">
        <rdfs:subClassOf rdf:resource="#Contextual_Information"/>
    </owl:Class>


    <!-- http://www.w3.org/2002/07/owl#Thing -->

    <owl:Class rdf:about="&owl;Thing"/>
</rdf:RDF>


<!-- Generated by the OWL API (version 2.2.1.1138) http://owlapi.sourceforge.net
-->
```

APÊNDICE C. ALTERAÇÕES NO METAMODELO FAML

Nas subseções a seguir são descritas todas as alterações realizadas no metamodelo FAML. Para a apresentação das alterações, foi utilizada a mesma distribuição dos conceitos proposta pelos autores do metamodelo. Nas figuras, as classes com preenchimento amarelo indicam classes definidas pelo FAML que sofreram alterações e as classes com preenchimento rosa salientam as novas classes criadas. A mesma divisão por cores foi utiliza nas tabelas que apresentam as definições dos conceitos após as alterações realizadas.

As classes relacionadas às informações contextuais estão com preenchimento amarelo, pois, embora não existisse o conceito `ContextualInformation` e seus derivados no metamodelo original, existiam os conceitos `Facet` e seus derivados.

C.1 Alterações nas classes externas ao agente em tempo de projeto

A Figura C.1 apresenta o diagrama de classes externas ao agente em tempo de projeto alterado. Nesse diagrama foram feitas as seguintes alterações:

- Renomeação da classe `FacetDefinition`.
  - Novo nome da classe: `ContextualInformationDefinition`.
  - Justificativa: no metamodelo FAML original, existia o conceito de faceta ao invés do conceito de informação contextual. Segundo Platon e co-autores [Pla07], um ambiente deve ter mecanismos para o gerenciamento de recursos e de informações contextuais. Como no metamodelo FAML também são definidos recursos, as informações contextuais indicadas em [Pla07] pareciam ser as facetas definidas em [Bey09]. De fato, as características das facetas são bem próximas das características das informações contextuais. Então, para evitar interpretações errôneas, alterou-se o metamodelo substituindo-se o conceito `FacetDefinition` pelo conceito `ContextualInformationDefinition`. Foram mantidos os seguintes atributos de `FacetDefinition` em `ContextualInformationDefinition`: `Name`, `Datatype`, `InitialValue` e `CanBeSensed`. O relacionamento existente entre as classes `FacetDefinition` e `System` foi mantido (cada

ContextualInformationDefinition tem um System como owner). Foi mantido também o relacionamento CanSense entre Role e ContextualInformationDefinition.

- Remoção de atributos da classe ContextualInformationDefinition.
  - Atributos removidos: CanBeChanged e CanChange (esses atributos estavam definidos na classe FacetDefinition).
  - Justificativa: a remoção desses atributos é justificada pelo fato das informações contextuais mudarem seus valores independentemente dos agentes que habitam o ambiente. Além disso, agentes não podem alterar o valor de uma informação contextual, só senti-lo.
- Criação de atributos na classe ContextualInformationDefinition.
  - Atributos criados: Type e Source.
  - Justificativa: de acordo com o metamodelo para classificação de informações contextuais utilizado neste trabalho (vide Seção 3.1.3), as informações contextuais podem sem temporais ou atemporais e, sendo uma informação contextual temporal, ela pode ser classificada como sentida, explicitada ou inferida. O atributo Type foi criado para armazenar o tipo da informação contextual de acordo com essa classificação. Já o atributo Source foi inserido para indicar a fonte utilizada para a captura da informação contextual.
- Remoção de relacionamento entre Role e ContextualInformationDefinition.
  - Relacionamento removido: associação de nome CanChange.
  - Justificativa: a associação CanChange existia entre Role e FacetDefinition. No entanto, considera-se que os agentes não possuem autonomia para modificar o valor de uma informação contextual, só senti-la. Logo, foi necessário remover esse relacionamento.
- Inclusão da classe ResourceSpecification.
  - Justificativa: a classe ResourceSpecification existia apenas no diagrama de classes internas ao agente em tempo de projeto. Como em tempo de execução o ambiente é também uma agregação de recursos, foi incluída a classe ResourceSpecification em tempo de projeto no diagrama de classes externas ao agente. Essa alteração é considerada

corretiva, visto que retifica uma inconsistência do metamodelo FAML original.

- Criação de atributos na classe `ResourceSpecification`.
  - Atributos criados: `Name`, `Creator`, `Schema`, `CanBeAcquired`, `CanBeChanged`, `CanChange`.
  - Justificativa: a inclusão dos atributos `Name`, `Creator`, `Schema` e `CanBeAcquire` é justificada pela própria definição dada pelos autores do FAML – para eles, recursos são definidos como "*Something that has a name* (atributo `Name`)*, may have reasonable representations* (atributo `Schema`) *and that can be acquired* (atributo `CanBeAcquire`)*, shared or produced* (atributo `Creator`)". Os atributos `CanBeChanged` e `CanChange` foram incluídos para controlar as mudanças realizadas nos recursos em tempo de execução. Essas alterações também são consideradas corretivas.

- Criação de relacionamento entre `ResourceSpecification` e `System`.
  - Relacionamento criado: associação para indicação do proprietário (*owner*) de um recurso.
  - Justificativa: o relacionamento foi criado para manter a consistência do metamodelo. No FAML, o ambiente habitado pelos agentes em tempo de execução é resultado da agregação de uma série de estruturas, como os recursos. Em tempo de projeto, essas mesmas estruturas são partes do sistema (conceito `System`). Assim, com a inclusão do conceito `ResourceSpecification`, foi necessário criar um relacionamento para indicar que cada `ResourceSpecification` tem um `owner` que é um `System`. Essa alteração também é considerada corretiva.

- Criação de relacionamentos entre `Role` e `ResourceSpecification`.
  - Relacionamentos criados: associações `CanAcquire` e `CanChange`.
  - Justificativa: com a inserção dos atributos `CanBeAcquire` e `CanBeChanged` na classe `ResourceSpecification`, foi necessário ajustar os relacionamentos do diagrama para indicar quais papéis que podem manipular os recursos públicos no ambiente. Assim, foram criadas duas associações entre `Role` e `ResourceSpecification`. A associação `CanAcquire` indica quais os papéis que podem adquirir determinado recurso. A associação `CanChange` indica quais os papéis que podem

modificar um recurso do ambiente. Por modificar, entende-se alterar, remover ou publicar um recurso no ambiente. Essa também é considerada uma alteração corretiva.

C.2 Alterações nas classes internas ao agente em tempo de projeto

A Figura C.2 apresenta o diagrama de classes internas ao agente em tempo de projeto alterado. Nesse diagrama foram feitas as seguintes alterações:

- Criação e inclusão da classe `SensorDefinition`.
  - Justificativa: o conceito `SensorDefinition` foi incluído para especificar a estrutura de um dado sensor em tempo de projeto e pode ser definido como "especificação da estrutura de um determinado sensor, o que inclui um e somente um padrão para captura de informações". O metamodelo FAML não indicava como diferentes informações eram captadas a partir do ambiente.
- Criação de relacionamento entre `SensorDefinition` e `AgentDefinition`.
  - Relacionamento criado: associação do tipo agregação.
  - Justificativa: a associação foi criada para indicar que um elemento da classe `AgentDefinition` pode conter zero ou mais definições de sensores.
- Criação de atributos na classe `ResourceSpecification`.
  - Atributos criados: `Name`, `Creator`, `Schema`, `CanBeAcquired`, `CanBeChanged`, `CanChange`.
  - Justificativa: já apresentada na Seção C.1.
- Remoção da classe `FacetActionSpecification`.
  - Justificativa: como o conceito `Facet` foi removido e os agentes não podem atualizar os valores das informações contextuais, foi removida a classe `FacetActionSpecification`. Com a remoção dessa classe, também foi removida a referência a classe `FacetDefinition` (tinha-se no diagrama original o relacionamento `changes` entre `FacetActionSpecification` e `FacetDefinition`).

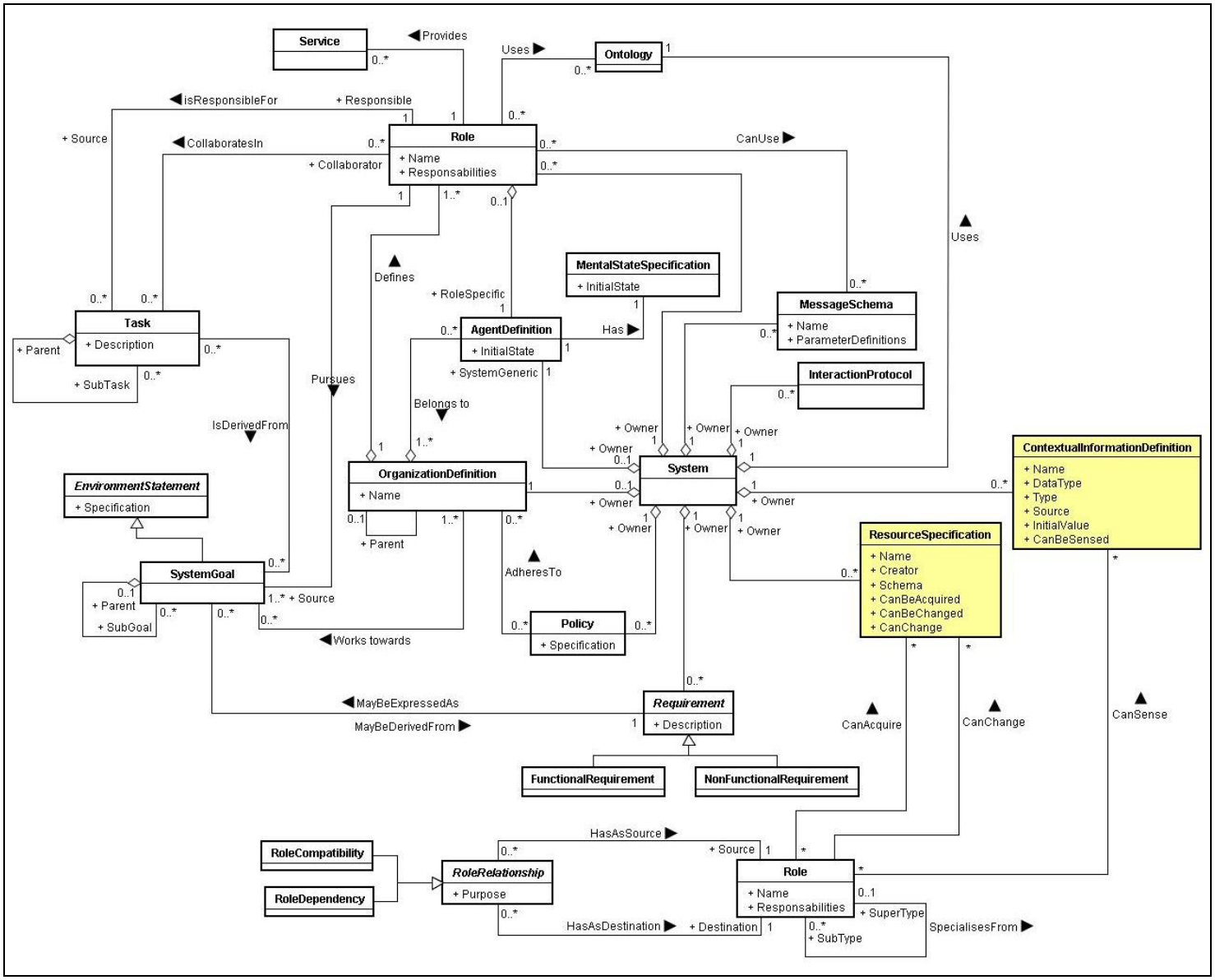

Figura C.1 - Classes externas ao agente em tempo de projeto (alterado de [Bey09]).

- Criação e inclusão da classe `ResourceActionSpecification`.
    - Justificativa: a classe `ResourceActionSpecification` foi criada para permitir que os agentes alterem, removam e publiquem recursos no ambiente.
- Criação de relacionamento entre as classes `ResourceActionSpecification` e `ActionSpecification`.
    - Relacionamento criado: especialização entre `ResourceActionSpecification` e `ActionSpecification`.
    - Justificativa: a classe `ResourceActionSpecification` representa um tipo de ação executada pelo agente.
- Criação de relacionamento entre as classes `ResourceActionSpecification` e `ResourceSpecification`.
    - Relacionamento criado: associação com o nome `changes`.
    - Justificativa: o relacionamento foi criado para indicar que a ação `ResourceActionSpecification` modifica um recurso.

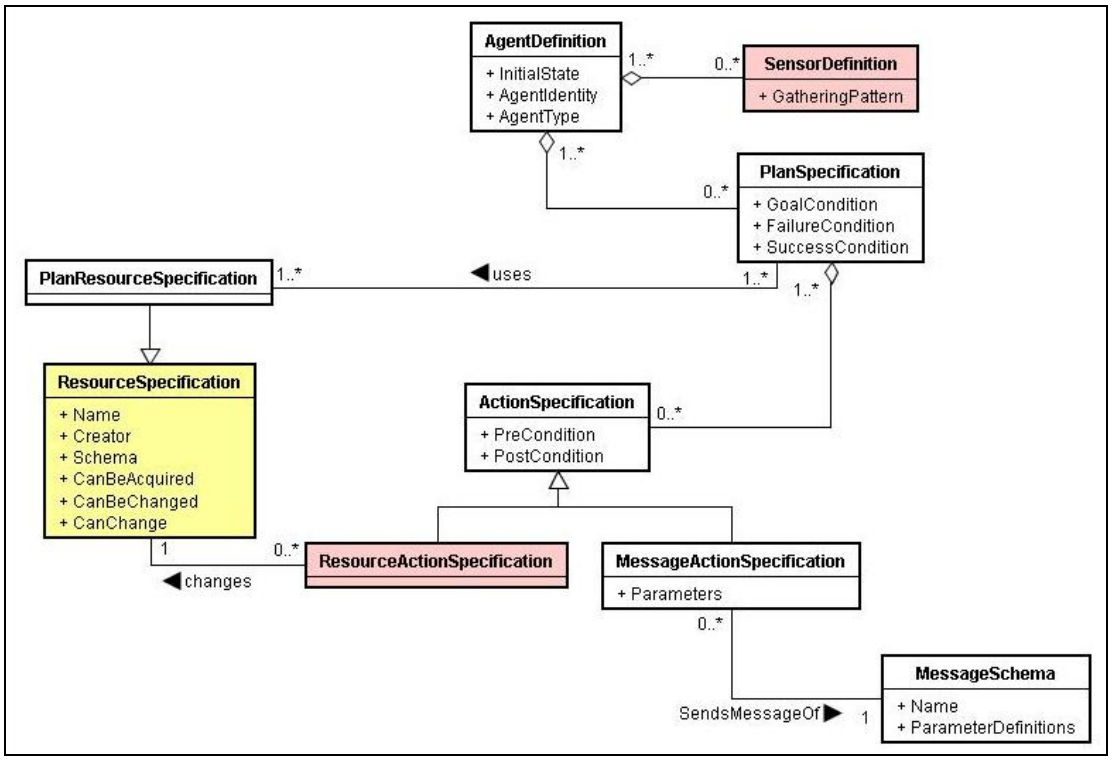

Figura C.2 - Classes internas ao agente em tempo de projeto (alterado de [Bey09])

C.3 Alterações nas classes externas ao agente em tempo de execução

A Figura C.3 apresenta o diagrama de classes externas ao agente em tempo de execução alterado. Nesse diagrama foram feitas as seguintes alterações:

- Renomeação da classe `Facet`.
  - Novo nome da classe: `ContextualInformation`.
  - Justificativa: conforme já discutido na Seção C.1, para evitar interpretações errôneas, a classe `FacetDefinition` foi renomeada para `ContextualInformationDefinition`. Assim, para manter a conformidade entre os diagramas de classes em tempo de projeto e em tempo de execução, a classe `Facet` também foi renomeada para `ContextualInformation`. Uma informação contextual pode ser definida como “informação ou propriedade utilizada para caracterizar o ambiente que pode ser sentida pelos agentes”. Os atributos e relacionamentos da antiga classe `Facet` foram mantidos.

- Renomeação da classe `FacetEvent`.
  - Novo nome da classe: `ContextualInformationEvent`.
  - Justificativa: a alteração foi realizada em decorrência da utilização do conceito informação contextual ao invés de faceta. Os atributos e relacionamentos da antiga classe `FacetEvent` foram mantidos.

- Renomeação da classe `FacetDefinition`.
  - Novo nome da classe: `ContextualInformationDefinition`.
  - Justificativa: já apresentada na Seção C.1.

- Remoção de atributos da classe `ContextualInformationDefinition`.
  - Atributos removidos: `CanBeChanged` e `CanChange` (estes atributos tinham sido definidos na classe `FacetDefinition`).
  - Justificativa: já apresentada na Seção C.1.

- Criação de atributos na classe `ContextualInformationDefinition`.
  - Atributos criados: `Type` e `Source`.
  - Justificativa: já apresentada na Seção C.1.

- Criação e inclusão da classe `ResourceEvent`.
  - Justificativa: como os recursos podem ser criados, alterados, compartilhados e adquiridos pelos agentes, foi criada a classe `ResourceEvent` para registrar os eventos ocorridos sobre os recursos do ambiente. A classe possui como atributos: `TargetResource`, que indica o recurso que sofreu a alteração; `ModificationType`, para indicar o tipo de modificação sofrida

pelo recurso; e `Responsible`, para indicar o agente responsável pela modificação. Considera-se que a inclusão ou remoção de recursos também são tipos de modificação.

- Criação de relacionamento entre as classes `ResourceEvent` e `Event`.
  - Relacionamento criado: especialização entre as classes `ResourceEvent` e `Event`.
  - Justificativa: o relacionamento foi criado para indicar que evento sobre um recurso (`ResourceEvent`) é também um tipo de evento.
- Criação de relacionamento entre as classes `ResourceEvent` e `Resource`.
  - Relacionamento criado: associação com nome `RefersTo`.
  - Justificativa: a associação foi criada para indicar que um `ResourceEvent` refere-se a um recurso (classe `Resource`).
- Inclusão da classe `ResourceSpecification`.
  - Justificativa: a classe `ResourceSpecification`, mesmo definida em outros diagramas, não aparecia no diagrama de classes externas ao agente em tempo de execução do metamodelo original. Como um recurso (classe `Resource`) é inicializado a partir de uma especificação de recurso, se introduziu a classe `ResourceSpecification` no diagrama. Essa alteração também é considerada corretiva, visto que retifica uma inconsistência do metamodelo FAML original.
- Criação de relacionamento entre as classes `Resource` e `ResourceSpecification`.
  - Relacionamento criado: associação com nome `IsInitialisedFrom`.
  - Justificativa: essa associação permite mostrar que um recurso é inicializado a partir de uma definição de recurso (associação `IsInitialisedFrom`). Também é uma alteração corretiva.

C.4 Alterações nas classes internas ao agente em tempo de execução

A Figura C.4 apresenta o diagrama de classes internas ao agente em tempo de execução alterado. Nesse diagrama foram feitas as seguintes alterações:

- Criação e inclusão da classe `ResourceAction`.

  - Justificativa: a classe `ResourceAction` foi definida com base na classe `ResourceActionSpecification` e foi criada para permitir que os agentes alterem, removam e publiquem recursos no ambiente em tempo de execução.

- Criação de relacionamento entre as classes `ResourceAction` e `Action`.
  - Relacionamento criado: especialização entre `ResourceAction` e `Action`.
  - Justificativa: a classe `ResourceAction` possui um relacionamento de especialização com a classe `Action`, visto que representa um tipo de ação executada pelo agente.

- Criação de relacionamento entre as classes `ResourceAction` e `Resource`.
  - Relacionamento criado: associação com nome `Changes`.
  - Justificativa: o relacionamento foi criado para indicar que uma ação do tipo `ResourceAction` modifica um recurso.

- Inclusão da classe `ResourceActionSpecification`.
  - Justificativa: a classe `ResourceActionSpecification`, criada no diagrama de classes internas ao agente em tempo de projeto, foi incluída para indicar que uma `ResourceAction` é gerada a partir de uma `ResourceActionSpecification`.

- Criação de relacionamento entre as classes `ResourceAction` e `ResourceActionSpecification`.
  - Relacionamento criado: associação `IsGeneratedFrom`.
  - Justificativa: permite indicar que um ação do tipo `ResourceAction` é gerada a partir de uma especificação definida em tempo de projeto (uma `ResourceActionSpecification`).

- Remoção das classes `FacetAction`, `FacetActionSpecification` e `Facet`.
  - Justificativa: como os agentes não podem alterar informações contextuais, as classes `FacetAction`, `FacetActionSpecification` e `Facet` foram removidas do diagrama.

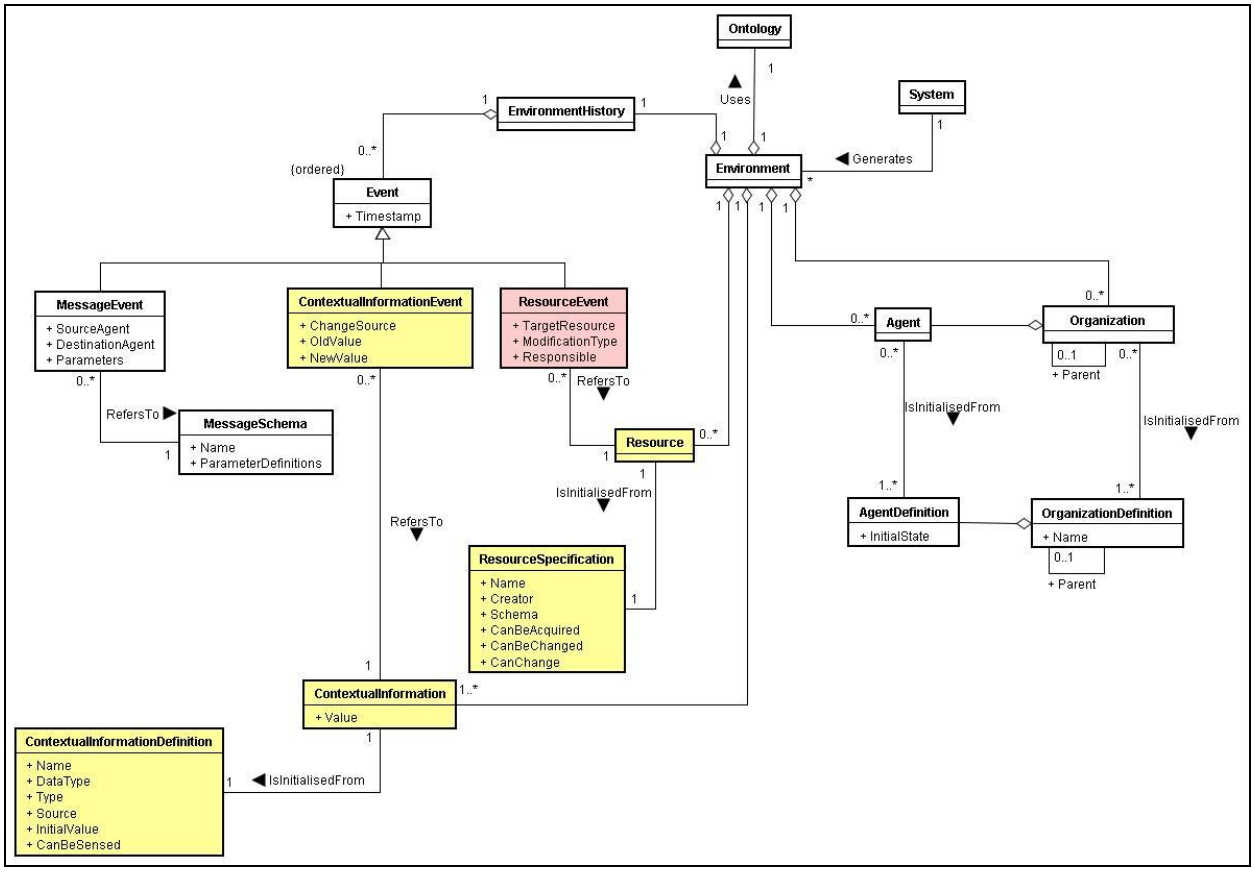

Figura C.3 - Classes externas ao agente em tempo de execução (alterado de [Bey09])

- Criação e inclusão da classe `Sensor`.
    - Justificativa: a classe `Sensor` foi criada para representar as estruturas utilizadas pelos agentes para captar diferentes tipos de informações a partir do ambiente. Com base no exposto em [Rib02], um sensor pode ser definido como “elemento utilizado pelos agentes para capturar determinados tipos de informação do ambiente”. A classe `Sensor` possui o atributo `Activated` para indicar se o sensor está ativo (está captando informações do ambiente) ou não.
- Criação de relacionamento entre as classes `Sensor` e `SensorDefinition`.
    - Relacionamento criado: associação com nome `IsInitialisedFrom`
    - Justificativa: o relacionamento foi criado para indicar que um sensor é inicializado a partir de uma definição de sensor, que é feita em tempo de projeto.
- Criação de relacionamento entre as classes `Agent` e `Sensor`.
    - Relacionamento criado: associação do tipo agregação.

- Justificativa: o relacionamento foi criado para indicar que um agente é também uma agregação de zero ou mais sensores.

- Criação e inclusão das classes `AgentHistory` e `InternalEvent`.
  - Justificativa: as classes foram definidas para que os agentes tivessem ciência dos eventos ocorridos internamente em sua arquitetura. Considera-se que o próprio agente terá um mecanismo para criar esses eventos. A classe `InternalEvent` possui os seguintes atributos: `EventType` para indicar o tipo de evento ocorrido; `Timestamp` para registrar o momento em que o evento ocorreu; `BaseStructure`, para indicar a estrutura base que sofreu alteração; `Structure`, para indicar a estrutura que foi adiciona/removida/alterada/executada na estrutura base; e `parameters`, visto que algumas operações contém, além da estrutura, uma lista de parâmetros.

- Criação de relacionamento entre as classes `AgentHistory` e `Agent`.
  - Relacionamento criado: associação do tipo agregação.
  - Justificativa: o relacionamento foi criado para indicar que cada agente possui um histórico de eventos internos.

- Remoção da classe `Obligation`.
  - Justificativa: segundo os próprios autores do FAML, a classe `Obligation` foi removida do metamodelo depois de atualizações (mas ainda permanecia na figura). Essa é, também, uma alteração corretiva.

C.5 Consolidação dos conceitos e definições

As Tabelas C.1 e C.2 apresentam a consolidação dos conceitos e definições depois de feitas as alterações no metamodelo FAML. As linhas com preenchimento rosa indicam os conceitos inseridos no metamodelo. Os conceitos que tiveram alterações nas definições estão com preenchimento amarelo. As linhas das tabelas sem preenchimento preservam as definições originais dos conceitos dadas pelos autores do metamodelo FAML [Bey09].

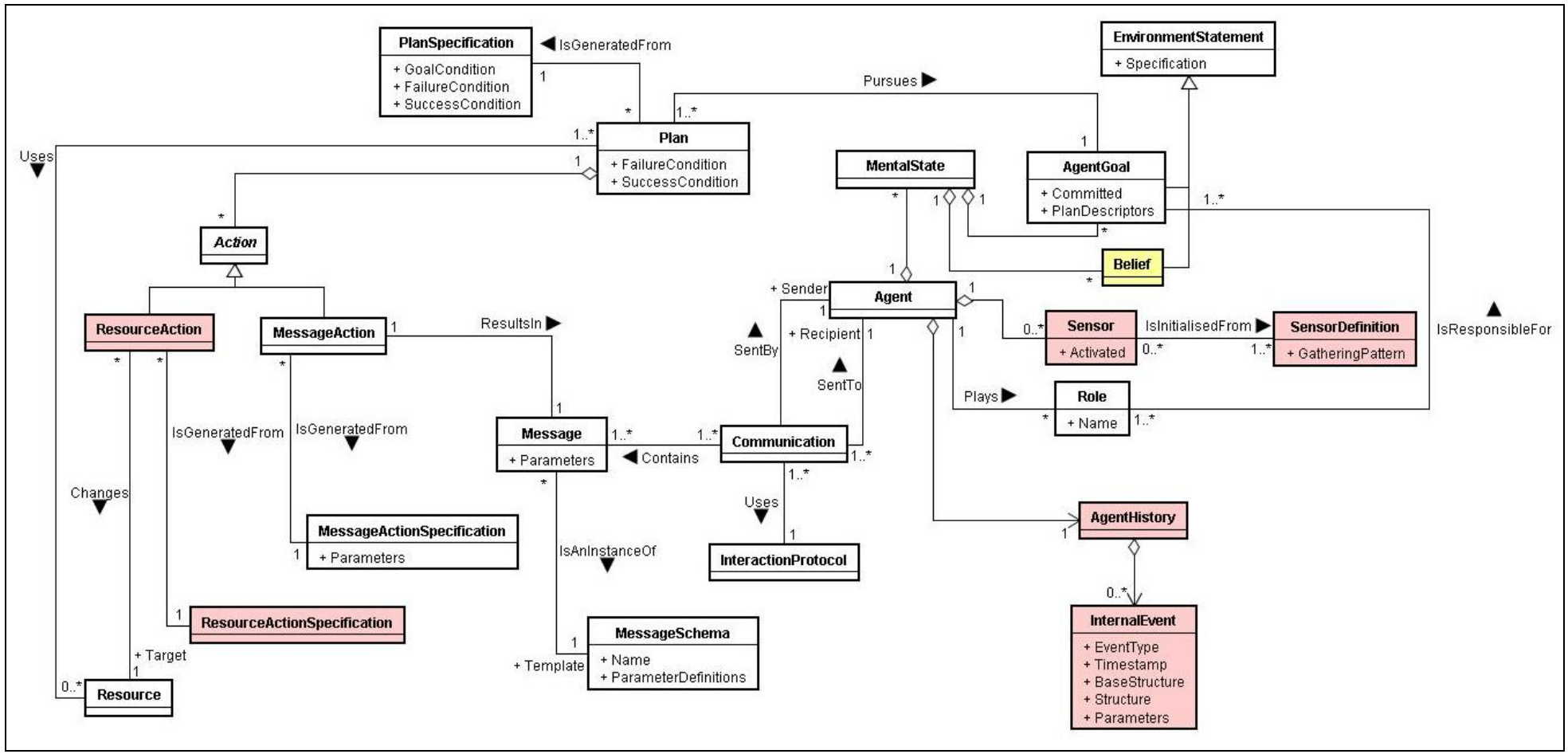

Figura C.4 - Classes internas ao agente em tempo de execução (alterado de [Bey09]).

Tabela C.1 - Novos conceitos em tempo de projeto e suas definições (com base no metamodelo FAML [Bey09].

Conceito	Definição
Action Specification	Specification of an action, including any preconditions and post-conditions.
Agent Definition	Specification of the initial state of an agent just after it is created.
Environment Statement	A Boolean statement about the environment.
Contextual Information Definition	Specification of the structure of a given contextual information, including its name, data type and access mode.
Functional Requirement	Requirement that provides added value to the users of the system.
Interaction Protocol	Specification of patterns of communications that occurs in the system.
Mental State Specification	Specification of the initial mental state in terms of specified beliefs and agent goals.
Message Action Specification	Specification of a message action in terms of the message schema and parameters to use.
Message Schema	Specification of the structure and semantics of a given kind of messages that can occur within the system.
Non-functional Requirement	Requirement about any limits, constraints, or impositions on the system to be built.
Ontology	Structural model of a given domain.
Organization Definition	Specification of a collection of roles and agents co-operating towards a system goal.
Plan Resource Specification	This is a specification of resources that are used in the Plan Specification.
Plan Specification	An organized collection of action specifications.
Policy	A rule that specifies an arrangement of events expected to occur in a given environment.
Requirement	Feature that a system must implement.
Resource Action Specification	Specification of a resource action in terms of the resource specification it will change.
Resource Specification	A resource specification specifies something that has a name and a creator, may have reasonable representations and that can acquired, shared or produced.
Role	Specification of a behavioral pattern expected from some agents in a given system.
Role Compatibility	Role relationship in which the source role is incompatible with the destination role for a given purpose.
Role Dependency	Role relationship in which the source role depends on the destination role for a given purpose.
Role Relationship	Social relationship between two roles for a given purpose.
Sensor Definition	Specification of the structure of a given sensor, including its gathering pattern.
Service	A single, coherent block of activity in which an agent may engage.
System	Final product of an agent-oriented software development project.
System Goal	A specification of a state of the environment that the system tries to achieve.
Task	Specification of a piece of behavior that the system can perform.

Tabela C.2 - Novos conceitos em tempo de execução e suas definições (com base no metamodelo FAML [Bey09]).

Conceito	Definição
Action	Fundamental unit of agent behaviour.
Agent	A highly autonomous, situated, directed and rational entity.
Agent Goal	An environment statement which represents a state pursued by an agent.
Agent History	The sequence of events that have occurred between the agent start-up and any given instant.
Belief	An environment statement held by an agent and deemed as true in a certain timeframe. A belief could be a rule, a fact or contextual information sensed by the agent.
Communication	Composition of more than one message.
Environment	The world in which an agent is situated.
Environment History	The sequence of events that have occurred between the environment start-up and any given instant.
Environment Statement	A statement about the environment.
Event	Occurrence of something that changes the environment history.
Contextual Information	Information about or property of the environment that agents can sense.
Contextual Information Event	Event that happens when the value of a contextual information changes.
Internal Event	Occurrence of something that changes the agent history.
Mental State	Agent goals and beliefs held by an agent at a certain timeframe.
Message	Unit of communication between agents, which conforms to a specific message schema.
Message Action	Action that results in a message being sent.
Message Event	Event that happens when a message is sent.
Organization	A collection of agents with specified roles co-operating towards a system goal.
Plan	An organized collection of actions that can be executed to pursue a particular agent goal.
Resource	Something that has a name, a creator, may have reasonable representations and that can be acquired, shared or produced.
Resource Action	Action that results in the change of a given resource.
Resource Event	Event that occurs when a resource is modified, shared or removed.
Role	Specification of a behavioural pattern expected from some agents in a given system.
Sensor	Element used by agents to capture information from environment.
System	Final product of an agent-oriented software development project.

ANEXO A. METAMODELO FAML

A Tabela A.1 mostra as definições dos conceitos em tempo de projeto utilizadas no metamodelo FAML e a Tabela A.2 as definições dos conceitos em tempo de execução também utilizadas no metamodelo.

As Figuras A.1-4 apresentam, respectivamente, as classes externas ao agente em tempo de projeto, as classes internas ao agente em tempo de projeto, as classes externas ao agente em tempo de execução e as classes internas ao agente em tempo de execução conforme o metamodelo FAML [Bey09]. Todas as definições e diagramas são uma cópia fiel do apresentado em [Bey09].

Tabela A.1. Conceitos em tempo de projeto e suas definições de acordo com o metamodelo FAML [Bey09].

Conceito	Definição
Action Specification	Specification of an action, including any preconditions and post-conditions.
Agent Definition	Specification of the initial state of an agent just after it is created.
Environment Statement	A Boolean statement about the environment.
Facet Action Specification	Specification of a facet action in terms of the facet definition it will change and the new value it will write to the facet.
Facet Definition	Specification of the structure of a given facet, including its name, data type and Access mode.
Functional Requirement	Requirement that provides added value to the users of the system.
Interaction Protocol	Specification of patterns of communications that occurs in the system.
Mental State Specification	Specification of the initial mental state in terms of specified beliefs and agent goals.
Message Action Specification	Specification of a message action in terms of the message schema and parameters to use.
Message Schema	Specification of the structure and semantics of a given kind of messages that can occur within the system.
Non-functional Requirement	Requirement about any limits, constraints, or impositions on the system to be built.
Ontology	Structural model of a given domain.
Organization Definition	Specification of a collection of roles and agents co-operating towards a system goal.
Plan Resource Specification	This is a specification of resources that are used in the Plan Specification.
Plan Specification	An organized collection of action specifications.
Policy	A rule that specifies an arrangement of events expected to occur in a given environment.
Requirement	Feature that a system must implement.
Resource Specification	A resource specification specifies something that has a name, may have reasonable representations and that can acquired, shared or produced.
Role	Specification of a behavioral pattern expected from some agents in a given system.
Role Compatibility	Role relationship in which the source role is incompatible with the destination role for a given purpose.
Role Dependency	Role relationship in which the source role depends on the destination role for a given purpose.
Role Relationship	Social relationship between two roles for a given purpose.
Service	A single, coherent block of activity in which an agent may engage.
System	Final product of an agent-oriented software development project.
System Goal	A specification of a state of the environment that the system tries to achieve.
Task	Specification of a piece of behavior that the system can perform.

Tabela A.2. Conceitos em tempo de execução e suas definições de acordo com o metamodelo FAML [Bey09].

Conceito	Definição
Action	Fundamental unit of agent behaviour.
Agent	A highly autonomous, situated, directed and rational entity.
Agent Goal	An environment statement which represents a state pursued by an agent.
Belief	An environment statement held by an agent and deemed as true in a certain timeframe.
Communication	Composition of more than one message.
Environment	The world in which an agent is situated.
Environment History	The sequence of events that have occurred between the environment start-up and any given instant.
Environment Statement	A statement about the environment.
Event	Occurrence of something that changes the environment history.
Facet	Property of the environment with which agents can interact.
Facet Action	Action that results in the change of a given facet.
Facet Event	Event that happens when the value of a facet changes.
Mental State	Agent goals and beliefs held by an agent at a certain timeframe.
Message	Unit of communication between agents, which conforms to a specific message schema.
Message Action	Action that results in a message being sent.
Message Event	Event that happens when a message is sent.
Organization	A collection of agents with specified roles co-operating towards a system goal.
Plan	An organized collection of actions that can be executed to pursue a particular agent goal.
Resource	Something that has a name, may have reasonable representations and that can acquired, shared or produced.
Role	Specification of a behavioural pattern expected from some agents in a given system.
System	Final product of an agent-oriented software development project.

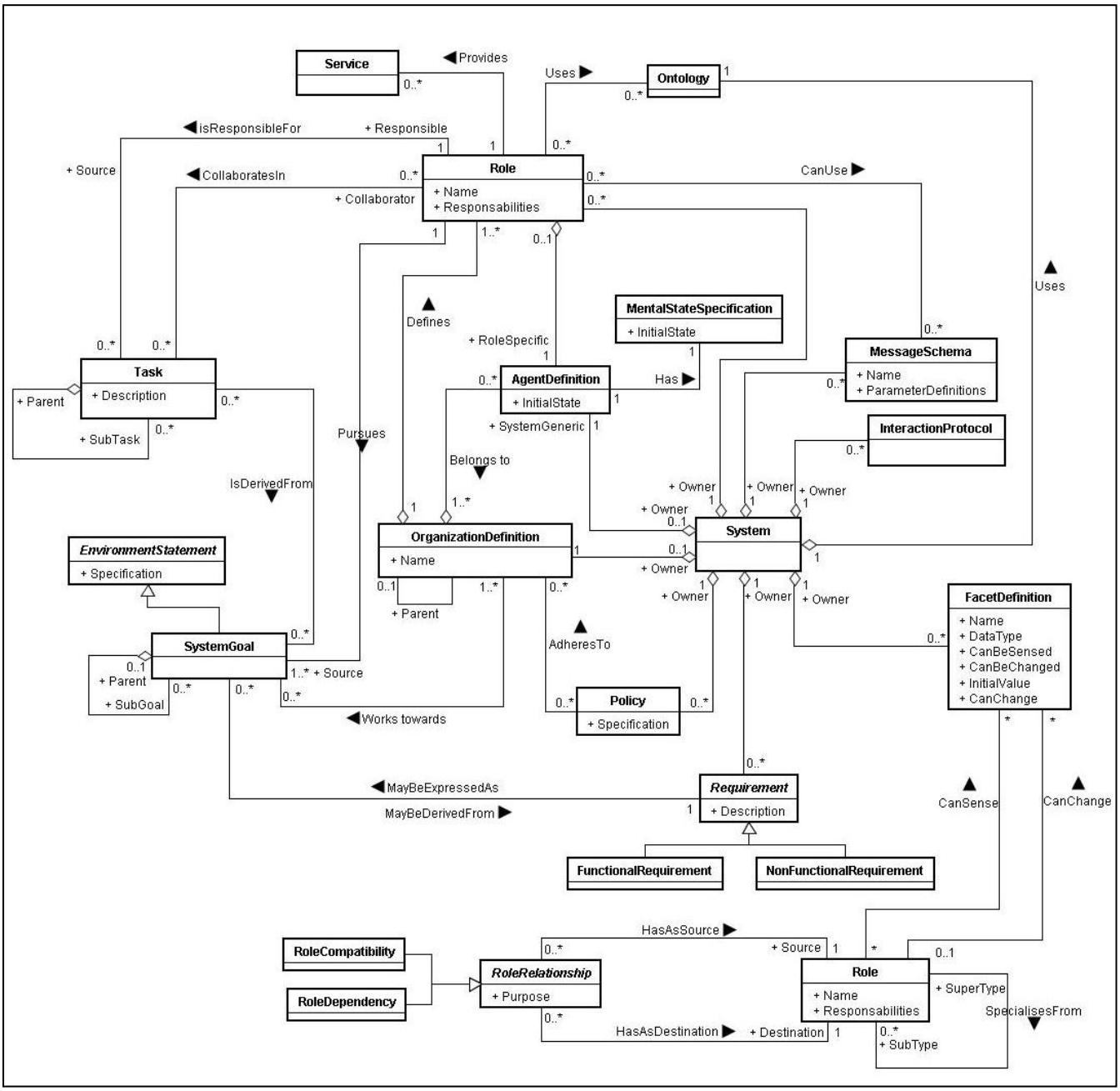

Figura A.1 - Classes externas ao agente em tempo de projeto segundo [Bey09].

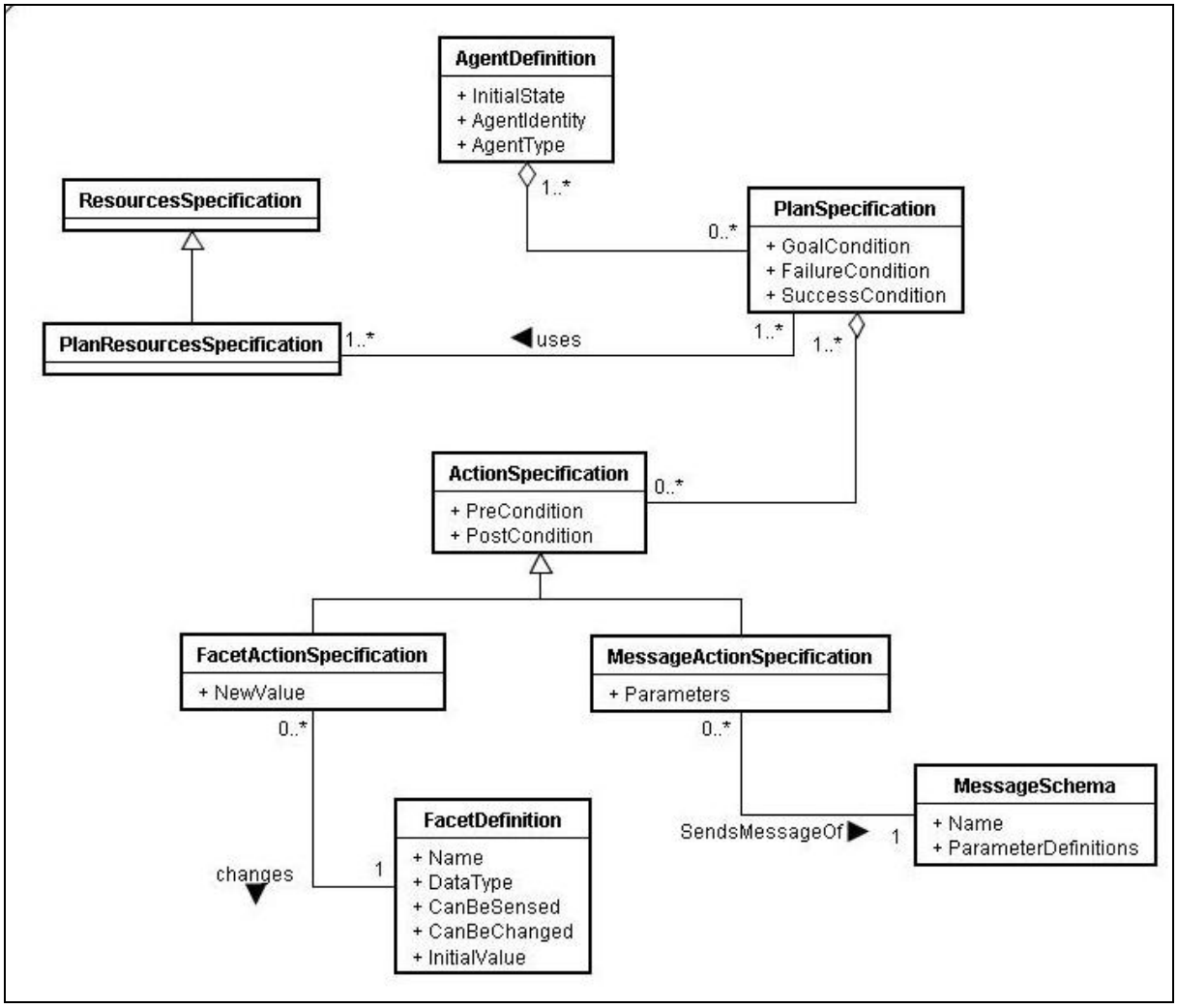

Figura A.2 - Classes internas ao agente em tempo de projeto segundo [Bey09].

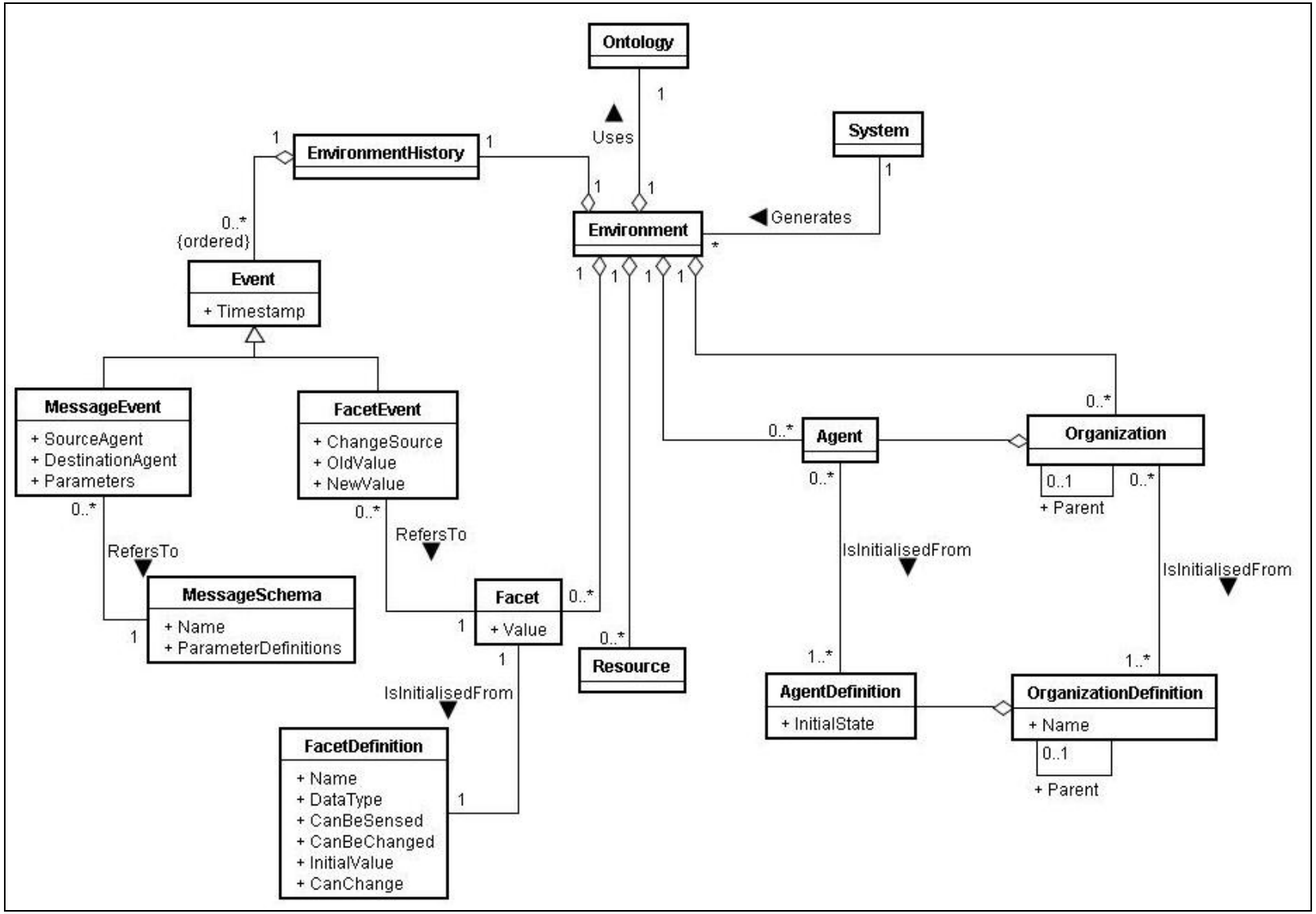

Figura A.3 - Classes externas ao agente em tempo de execução segundo [Bey09].

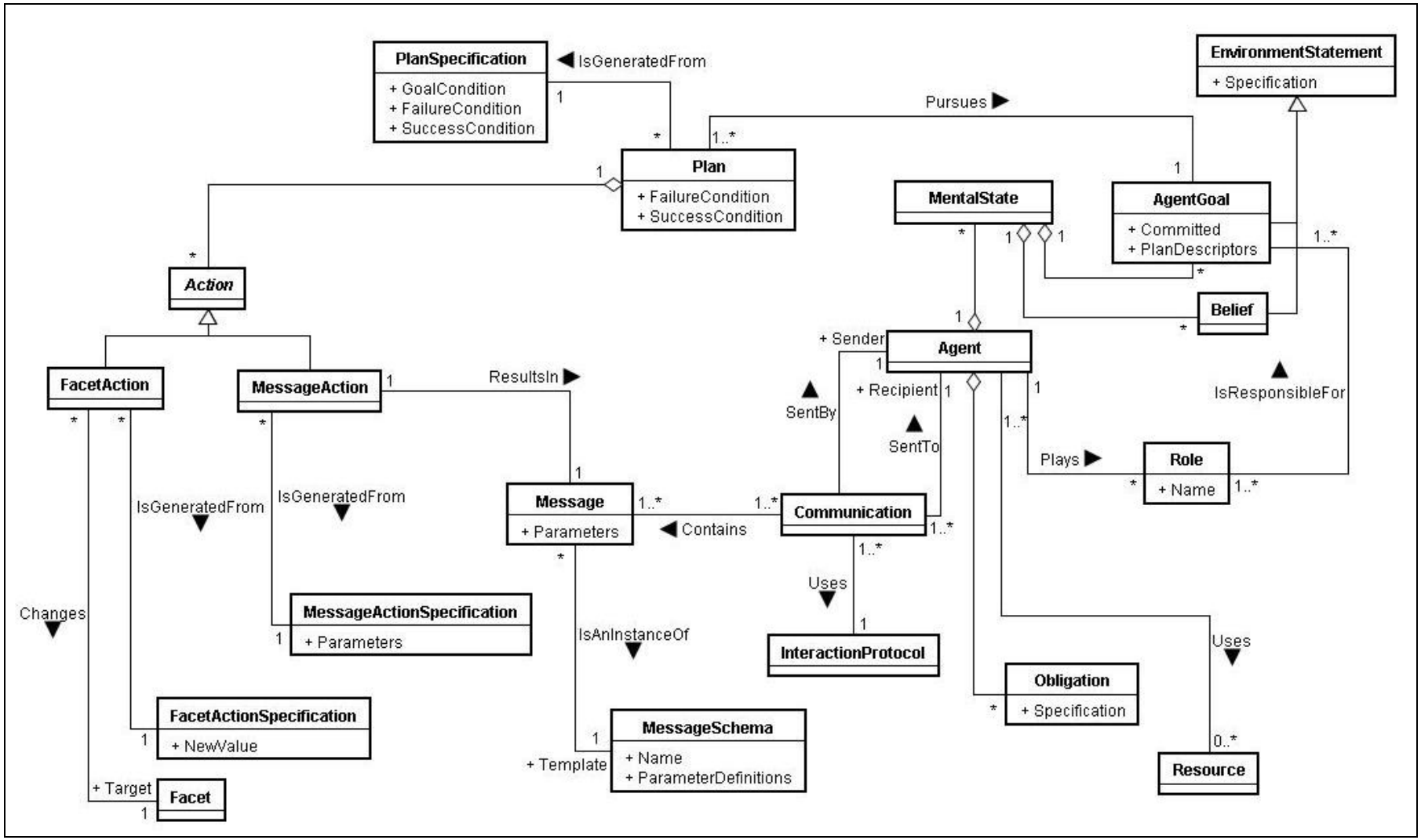

Figura A.4 - Classes internas ao agente em tempo de execução segundo [Bey09]