3
Engenharia de Software para Sistemas Multi-Agentes

No nível arquitetural, a principal abordagem atualmente utilizada para o desenvolvimento de ASCs é o uso de *middlewares* (Capítulo 2). Entretanto, nos níveis de projeto e implementação, algumas pesquisas têm verificado a usabilidade de novas abstrações e mecanismos de decomposição, propostos por outros paradigmas, que não a Orientação a Objetos, no desenvolvimento de ASCs (Harroud et al, 2004). Destacamos o interesse da comunidade acadêmica em verificar a aplicabilidade de ESSMA na Computação Móvel (Endler, 2007).

Neste capítulo, são introduzidos os conceitos fundamentais de ESSMA (Seção 3.1) e o modo como tais conceitos são suportados pelos serviços e API de JADE (Bellifemine et al, 1999), a plataforma de suporte a agentes utilizada em nossos estudos de caso. Além disso, é apresentada a nossa proposta de reengenharia das ASCs VL e de WMS (Seção 2.3) usando as APIs de JADE e de MoCA (Seção 3.2). Os conceitos recorrentes de ESSMA em ASCs, tendo em vista o processo de reengenharia anterior, constitui a base de elaboração do *framework* CAAF (Capítulo 4).

3.1.
Definições Básicas de ESSMA

Agentes de software constituem a unidade de modularização básica dos Sistemas Multi-Agentes (SMAs), isto é, dos sistemas desenvolvidos a partir de ESSMA. Eles são desenvolvidos a partir de conceitos básicos definidos no nível de aplicação, como os elementos do conhecimento de um agente (Seção 3.1.1) e tipos de agentes (Seção 3.1.2). Outros conceitos, mais complexos, relacionados às propriedades básicas de agência (Seção 3.1.3), são geralmente suportados pelo uso de ferramentas como JADE (Seção 3.1.4). A seguir, introduzimos os conceitos

fundamentais de SMAs usados neste trabalho segundo o *framework* TAO[2] desenvolvido por Garcia (2004) e Silva (2004).

3.1.1. Elementos do Conhecimento de Agentes

O conhecimento de um agente consiste em: (1) informações sobre o ambiente do agente ou sobre o agente em si, (2) além do conjunto de serviços providos pelo agente. O *conhecimento intrínseco* é o conhecimento relacionado às informações e serviços básicos do agente. Já o *conhecimento extrínseco* é o conhecimento relacionado aos diferentes papéis desempenhados pelo agente nos vários contextos de colaboração.

Para representar o conhecimento, os elementos freqüentemente utilizados são designados por crenças, objetivos, ações e planos. As *crenças* são os elementos que descrevem as informações sobre a agente em si, sobre o ambiente e outros agentes. Um *plano* descreve a estratégia usada para se alcançar um objetivo do agente e a seleção de planos é baseada nas crenças do agente. O comportamento de um agente é dirigido pela execução de seus planos, os quais selecionam *ações* específicas a fim de alcançar seus *objetivos*.

3.1.2. Tipos de Agentes

Os SMAs podem possuir vários tipos de agentes, classificados de acordo com o conjunto de serviços que oferecem. Para o escopo deste trabalho, os agentes são classificados em: (1) agentes de informação e (2) agentes de usuário.

Agentes de informação são tipos de agentes *fortemente acoplados* às fontes de informação com o objetivo de encontrar informações em resposta a consultas solicitadas. Estes agentes *monitoram ativamente* as fontes de informação para detecção de condições pré-especificadas e podem ser utilizados para *automatizar o processo de decisão* quanto à utilidade de uma informação disponível. Tornam assim transparente a complexidade e heterogeneidade do acesso à informação e fornecem mecanismos de divulgação eficientes dos resultados encontrados.

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

No comércio eletrônico, são encontrados exemplos clássicos de agentes de informação. Por exemplo, um agente de informação pode ser usado para montar uma página com as principais notícias do dia em um servidor de *newsgroup*. Em outro exemplo, um agente de informação é utilizado para fornecer conselhos sobre produtos em um mercado de vendas. Nos dois casos, o agente de informação pode ser projetado a fim de se comunicar com agentes do usuário.

Agentes de usuário representam entidades físicas, em geral seres humanos, que agem como assistentes pessoais no cumprimento de tarefas do usuário. Dentre outras responsabilidades, adaptam seu comportamento de acordo com os interesses dos usuários. É possível também utilizá-los para gerenciar as interfaces gráficas com as quais os usuários interagem.

Como exemplo, um agente de usuário pode ser utilizado para interagir com *sites* da Internet em busca de um livro e comprar o produto no local em que o preço é menor. O agente então pesquisa na rede segundo os critérios estabelecidos pelo usuário e retorna com o resultado já após a compra do livro. Para o cumprimento desta tarefa, é possível que o agente de usuário se comunique com agentes de informação acoplados às bases de dados das livrarias.

3.1.3. Propriedades de Agência

Segundo Garcia (2004) e Silva (2004), parece haver consenso de que as propriedades básicas de agência são interação, adaptação e autonomia. Um agente de software básico consiste então em: (1) no conhecimento e serviços associados ao tipo do agente (Seção 3.1.1), (2) nos mecanismos para a interação com o ambiente a sua volta, (3) na adaptação do conhecimento e do comportamento do agente de acordo com as interações realizadas e (4) na autonomia, isto é, em um comportamento orientado à satisfação dos objetivos do agente.

A *interação* é a propriedade de agente que define a comunicação com o ambiente externo. O comportamento de interação consiste em receber e enviar mensagens para outros agentes através de sensores e efetuadores. Os *sensores* detectam a chegada de novas mensagens de outros agentes e as alterações nos objetos do ambiente. Os *efetuadores* enviam mensagens ou geram eventos no ambiente. Quando um evento externo é detectado por um sensor, ele é traduzido

para um formato de mensagem interna e armazenado em uma caixa de mensagens de agente. Ao contrário, antes de enviar uma mensagem, ela é armazenada em uma caixa de saída de agente e traduzida em um formato de mensagem específico de forma que o agente receptor possa interpretá-la.

A *adaptação* é a propriedade que modifica o comportamento e o conhecimento do agente de acordo com os eventos internos e externos. Uma adaptação de conhecimento resulta na modificação de alguma parte do conhecimento de agente. A adaptação de comportamento resulta no cancelamento do plano ou na seleção de novos planos que devem ser executados. A propriedade de adaptação consiste então em observar os eventos relevantes, juntar as informações necessárias, selecionar e chamar os adaptadores associados.

A *autonomia* é a propriedade que um agente possui quando é capaz de controlar suas próprias ações e agir de forma independente de outros, de acordo com seus objetivos. Para alcançar tais objetivos, os agentes têm suas próprias *threads* de controle e podem realizar ações sem uma intervenção externa direta.

3.1.4. Serviços e API de JADE

Java Agent DEvelopment Framework, ou simplesmente, JADE (Bellifemine et al, 1999) é uma plataforma de agentes que disponibiliza uma API para desenvolvimento de SMAs em conformidade com o padrão FIPA[3] (FIPA).

O objetivo de JADE é simplificar o desenvolvimento de SMAs ao mesmo tempo em que assegura um conjunto de serviços de sistema que obedecem ao padrão FIPA (Bellifemine et al., 1999). Mais especificamente: as classes e interfaces da API de JADE são projetadas para permitir o gerenciamento da plataforma de agentes e dos agentes individuais.

Para o gerenciamento da plataforma, três elementos foram identificados como obrigatórios: o Agente de Administração do Sistema[4] (AMS), que controla o acesso à plataforma sendo responsável pela autenticação e controle de inscrições de agentes; o Agente de Canal de Comunicação[5] (ACC), responsável pela

[3] FIPA: *The Foundation for Intelligent Physical Agents.*

[4] Do inglês: *Agent Management System.*

[5] Do inglês: *Agent Communication Channel.*

comunicação entre agentes internos ou externos a plataforma; o Diretório Facilitador[6] (DF), que provê um serviço de páginas amarelas para a plataforma.

Quanto aos agentes individuais, as abstrações mais importantes são "Agente", "Identificador", "Ambiente" e "Comportamento". Os agentes JADE são implementados como *threads* Java e inseridos dentro de repositórios de agentes chamados de *contêineres*, os quais fornecem um *ambiente* de execução para agentes executando concorrentemente os seus respectivos *comportamentos*.

Para criar um agente JADE, deve-se estender a classe `Agent`, implementar os comportamentos da aplicação, instanciá-los e associá-los ao agente. Existem vários tipos de comportamentos pré-definidos na API; todos eles são implementações da interface `Behavior`. Além de comportamentos, um contêiner deve ser instanciado a partir da classe `AgentContainer`.

Cada objeto `AgentContainer` permite o gerenciamento do ciclo de vida do conjunto de agentes associados a ele. O identificador de um agente JADE é implementado através da classe `AID`. Este identificador possui um papel fundamental no mecanismo de comunicação, sendo usado para a especificação de emissores e receptores de mensagens do tipo `ACLMessage`.

Portanto, do ponto de vista do engenheiro de SMAs, um agente JADE é uma classe Java que estende a classe `Agent` da plataforma JADE. A extensão da classe `Agent` permite a utilização de um conjunto de métodos para manipulação do ciclo de vida de agentes e um conjunto de métodos abstratos para definição de *ações* específicas nas aplicações. Alguns exemplos destes métodos são:

- `setup()`: utilizado na inicialização de um agente;
- `void doDelete()`: executa procedimentos referentes à destruição do agente, sendo chamado pela plataforma ou pelo próprio agente;
- `doMove(Location destino)`: é invocado pela plataforma ou pelo próprio agente para iniciar o processo de migração;
- `beforeMove()`: é invocado automaticamente pela plataforma imediatamente antes da partida do agente para outro host;
- `afterMove()`: é invocado automaticamente pela plataforma imediatamente após a chegada do agente em novo host.

[6] Do inglês: *Directory Facilitator*.

3.2. Reengenharia de ASCs usando JADE

Nesta seção, apresentamos a reengenharia das aplicações VL e WMS (Seção 2.2) usando ESSMA e, particularmente, as APIs de JADE (Seção 3.1.4).

3.2.1. Virtual Lines

A Figura 15 apresenta a reengenharia da aplicação cliente de VL. A principal mudança em relação à Figura 6 é a introdução da classe `AgenteUsuario` e a implementação das responsabilidades deste agente como comportamentos de JADE. Note que as classes `Utils`, `Mensagem` e `TCPConnection`, presentes na Figura 6, são suprimidas da Figura 15, porque estão agora encapsuladas na propriedade de interação (Seção 3.1.3) suportada por JADE (Seção 3.1.4). A classe `MainCliente` também é redefinida para instanciar o `AgenteUsuario`.

Uma outra diferença é que o gerenciamento da classe `FramePrincipal` está agora concentrado diretamente na classe `AgenteUsuario`, visto que funcionalidades como a visibilidade para a fila de atrações e a solicitação pelo usuário de uma nova reserva são agora responsabilidades diretas da classe `AgenteUsuario`. Em outras palavras, a classe `FramePrincipal`, antes fortemente acoplada com o serviço LIS, está agora livre do uso direto das APIs de MoCA.

Pelo mesmo motivo, a classe `ListenerFilas`, responsável por permanecer alerta quanto às filas de atrações, prescinde do uso direto da API referente ao serviço DS de MoCA. Em resumo, as classes `ListenerFilas` e `MainCliente` deixam de utilizar os serviços e APIs de MoCA, pois tais dependências estão encapsuladas na classe `AgenteUsuario`, que, por sua vez, passa a ter a responsabilidade de gerenciar a classe `FramePrincipal`.

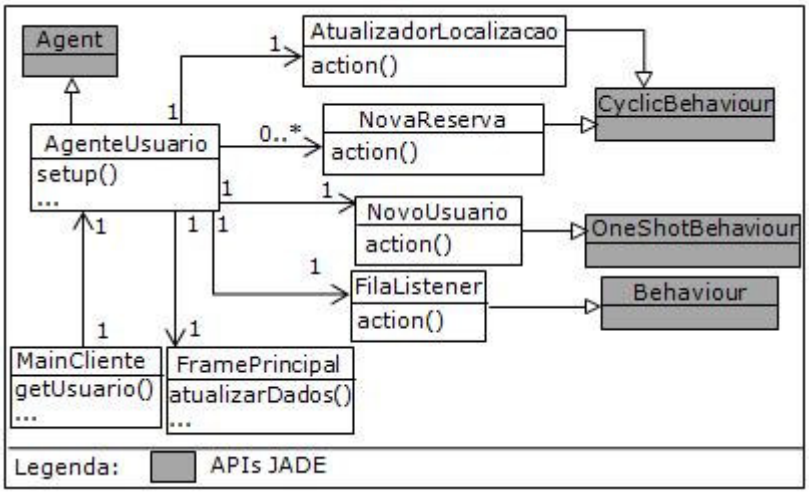

Figura 15. Reengenharia do Módulo Cliente de VL usando ESSMA

A Figura 16 apresenta a reengenharia da aplicação servidora de VL. Como na Figura 15, a principal mudança em relação à Figura 7 é a introdução das classes de agentes, `AgenteServidor` e `AgenteGerenciador`, além da implementação das responsabilidades destes agentes como comportamentos JADE. As classes `Mensagem`, `CaixaDeMensagens` e `TCPConnection`, presentes na Figura 7, são também suprimidas na Figura 16, porque estão agora encapsuladas na propriedade de interação, suportada diretamente por JADE. A classe `MainServidor` também é redefinida para instanciar o `AgenteServidor`.

Além disso, as responsabilidades da Figura 7 são redistribuídas na Figura 16: (1) a inicialização da *thread* de simulação das atrações, antes efetuada pelo `GerenciadorReservas`, é agora de responsabilidade do `AgenteGerenciador`; (2) a manutenção da lista de usuários presentes no parque bem como das reservas nas filas de atrações são agora de responsabilidade do `AgenteServidor`.

Em particular, note a reimplementação da classe `PublicadorReservas` como um comportamento do `AgenteServidor`, para fazer uso da propriedade de interação de um agente JADE. `MainServidor` também substitui a antiga conexão TCP da Figura 7 pela capacidade de interação de JADE. Em resumo, para implementar suas funcionalidades, não é mais necessário que a aplicação servidora se utiliza diretamente das APIs e dos serviços de MoCA.

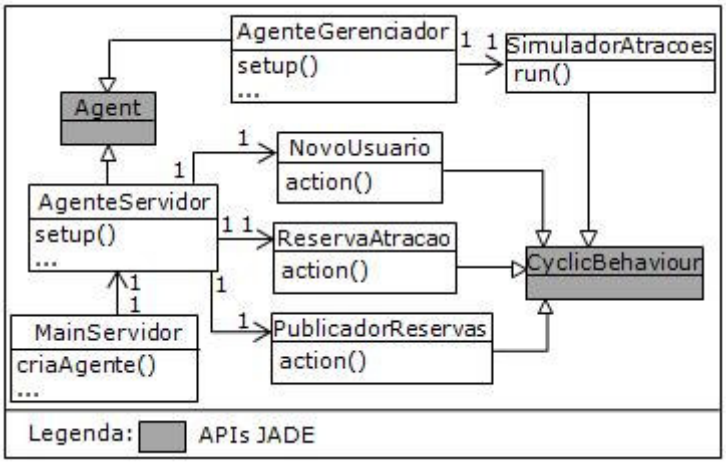

Figura 16. Reengenharia do Módulo Servidor de VL usando ESSMA

A Figura 17 ilustra o mesmo cenário da Figura 8 para a realização da criação de um usuário. A principal diferença aqui decorre do fato de a configuração inicial da posição do usuário ser agora delegada para os agentes da aplicação servidora, os quais retornam a posição para o `AgenteUsuario`, que a mantém como um atributo simples. Neste caso, a comunicação passa a ser assíncrona, pois não é mais realizada pela invocação do serviço LIS de MoCA.

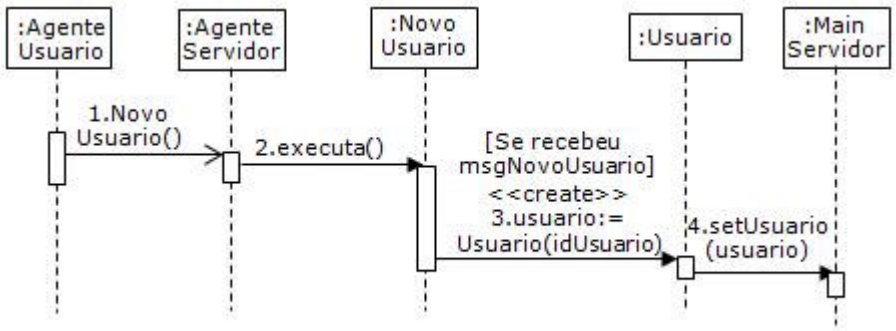

Figura 17. Cenário de Criação do Usuário em VL

Note também a simplificação da Figura 18 quando comparada à Figura 9. De fato, no cenário que envolve a solicitação de uma reserva por um usuário, a comunicação entre as aplicações cliente e servidora agora ocorre entre os agentes, os quais possuem sua propriedade de interação diretamente suportada por JADE.

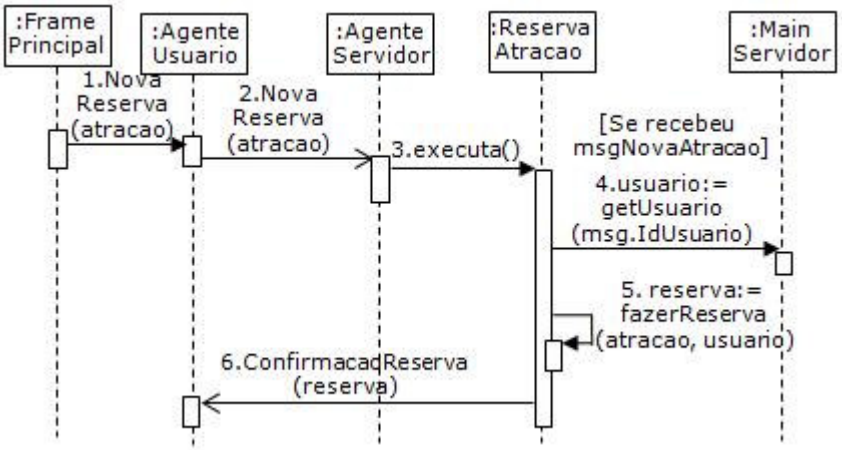

Figura 18. Cenário de Solicitação de uma Reserva em VL

Por fim, note também que, apesar de todas as modificações no projeto e implementação decorrentes da reengenharia de VL, as classes de negócio permanecem inalteradas; a diferença é que agora estas são utilizadas pelos agentes introduzidos nas aplicações, permanecendo ainda independentes de MoCA.

3.2.2. Wireless Marketing Service

A Figura 19 apresenta a reengenharia da do "WMS Servidor". Novamente, a principal diferença em relação à Figura 12 é a introdução da classe `WMSAgent` e a implementação das responsabilidades deste agente como comportamentos de JADE. Conseqüentemente, as classes `WMSSender` e `WMSResponse` não mais implementam as interfaces `DeviceListener` e `ConnectionListener` de MoCA. Por sua vez, as classes `SyncTCPServer` e `ConnectionWaiter` também não são mais necessárias para o monitoramento das portas de comunicação. Toda a troca de mensagens é agora realizada entre agentes e suportada diretamente por JADE.

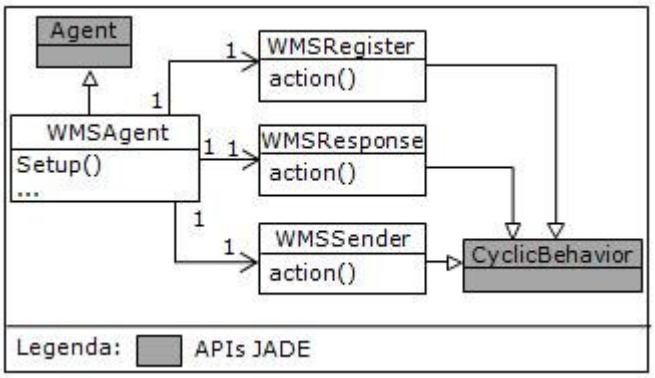

Figura 19. Reengenharia do Módulo Servidor de WMS usando ESSMA

A Figura 20 apresenta a reengenharia de "WMS Cliente". Novamente, a diferença é a introdução do agente `WMSClientAgent` e de seus comportamentos `WMSClientRegister`, `WMSClientNewCoupon` e `WMSClientCoupon`. As classes de MoCA acessadas diretamente na Figura 13 foram suprimidas na Figura 20.

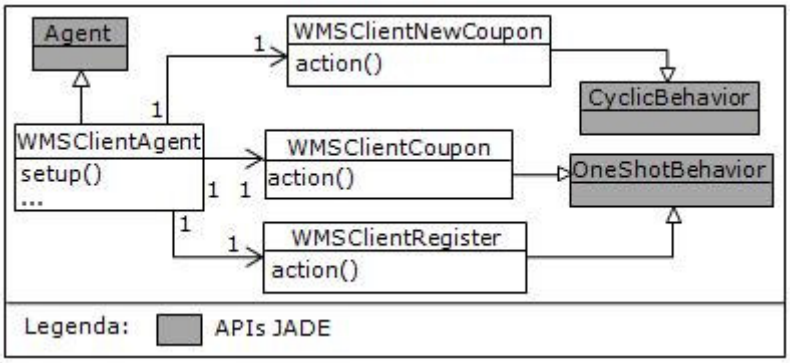

Figura 20. Reengenharia do Módulo Cliente de WMS usando ESSMA

3.3. Análise do Uso de ESSMA na Reengenharia de ASCs

Comparando a seção 2.2 com os resultados da seção 3.2, verificamos que a especificação das novas versões de VL e WMS constitui a primeira contribuição de nosso trabalho, visto que a estrutura e dinâmica destas ASCs é bastante simplificada pelo uso de ESSMA. Não só isso, mas na reengenharia das aplicações cliente e servidor percebe-se um padrão geral de reestruturação.

Quanto ao cliente de VL (Figura 15) e de WMS (Figura 20), este padrão pode ser notado a partir da introdução recorrente de uma classe referente a um "agente de usuário" (Seção 3.1.2) e da redistribuição das funcionalidades da aplicação cliente como ações deste agente (Seção 3.1.1). Além disso, as classes de MoCA não se fazem mais necessárias no cliente, porque são substituídas em sua função por aquelas encapsuladas na propriedade de interação do agente de usuário (Seção 3.1.3). Uma outra característica recorrente nas versões cliente é que o gerenciamento de interface passa a ser de responsibilidade do agente de usuário e, consequentemente, pode ser totalmente desacoplado do LIS de MoCA.

No que diz respeito ao servidor de VL (Figura 16) e de WMS (Figura 19), o padrão de reestruturação também pode ser notado. A introdução de um agente que representa o servidor e a especificação de suas ações como comportamentos JADE é recorrente. Entretanto, ao contrário do que acontece na aplicação cliente,

o uso de serviços de MoCA passa a estar concentrado na aplicação servidora. Evidentemente, isto não impede o uso das APIs de MoCA pela aplicação cliente; apenas previne seu uso. Além disso, os *listeners* de contexto (*publisher* e *subscriber*) também podem prescindir do uso direto das APIs de MoCA, visto que são implementados mais facilmente através de comunicação assíncrona entre os agentes no cliente e no servidor.