Formalização e Análise de Padrões de Projeto para Agentes Móveis

Emerson Ferreira de Araújo Lima

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Engenharia de Software

Patrícia Duarte de Lima Machado

(Orientadora)

Jorge César Abrantes de Figueiredo

(Orientador)

Campina Grande, Paraíba, Brasil

LIMA, Emerson Ferreira de Araújo
L732F

Formalização e Análise de Padrões de Projeto para Agentes Móveis.

Dissertação de Mestrado, Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande, Paraíba, Fevereiro de 2004.

172 p. Il.

Orientadores: Patrícia Duarte de Lima Machado
Jorge César Abrantes de Figueiredo

Palavras Chave:

1. Engenharia de Software
2. Agentes Móveis
3. Padrões de Projeto
4. Redes de Petri

CDU - ~~519.683~~ 004.41(043)

"FORMALIZAÇÃO A ANÁLISE DE PADRÕES DE PROJETO PARA AGENTES MÓVEIS"

EMERSON FERREIRA DE ARAÚJO LIMA

DISSERTAÇÃO APROVADA COM DISTENÇÃO EM 19.02.2004

PROFª PATRICIA DUARTE DE LIMA MACHADO, Ph.D
Orientadora

PROF. JORGE CÉSAR ABRANTES DE FIGUEIREDO, D.Sc
Orientador

PROF. DALTON DARIO SEREY GUERRERO , D.Sc
Examinador

PROF. PAULO HENRIQUE MONTEIRO BORBA, Ph.D
Examinador

CAMPINA GRANDE – PB

Resumo

Agentes Móveis são programas que podem migrar entre diferentes computadores de forma autônoma, continuando sua execução do ponto onde pararam antes de migrar. Muitos estudos de caso têm mostrado a aplicabilidade dessa abordagem, porém ainda se encontra dificuldade no desenvolvimento desse tipo de aplicação, devido à complexidade inerente e falta de metodologias adequadas. Uma abordagem que pode facilitar e aprimorar o desenvolvimento de aplicações baseadas em agentes móveis é o uso de Padrões de Projeto. Vários trabalhos têm proposto padrões de projeto para agentes móveis, entretanto estes apresentam alguns problemas como a escassez de documentação, ausência de classificação unificada, falta de relacionamento entre os padrões e carência de estudos de caso. O objetivo principal do nosso trabalho de dissertação é a classificação e formalização de padrões de projeto para agentes móveis, e análise de desempenho de combinações destes padrões. Com isto, pretendemos prover suporte e facilitar o acesso aos diversos padrões que têm sido propostos, o que conseqüentemente pode ajudar, melhorar e incentivar o desenvolvimento de aplicações baseadas em agentes móveis. No desenvolvimento de nosso trabalho, inicialmente faremos um levantamento de padrões propostos, catalogando-os e definindo uma classificação. Então, estes padrões serão formalizados, de maneira a obtermos uma descrição do comportamento destes sem ambigüidade e podermos fazer análises sobre eles. Finalizando, um estudo de caso será definido para aplicação de combinações dos padrões de projeto e, a partir destas, serão feitas análises, verificando que combinações têm o melhor desempenho em cada situação a ser avaliada.

Abstract

Mobile Agents are programs that can autonomously migrate between different computers, continuing their execution in the point where they stopped before migrating. Many case studies have shown the applicability of this approach, however difficulties are still found in this kind of application development. An approach that can make easier and improve mobile agent-based applications development are the Design Patterns. Many researches propose mobile agent design patterns, nevertheless there are some problems, as lack of documentation, absence of classification and relation among the various patterns. Besides, few case studies are presented, what makes difficult know the applicability of these patterns in the several platforms. The main goal of our work is the classification and formalization of mobile agents design patterns, and performance analysis of these patterns combinations. In this way, we intend to provide support and make the access to these patterns easier, assisting, improving and encouraging mobile agent-based applications development. In this work, we will make a survey and classification of the already proposed patterns. Then, these models will be formalized, in such a way that we obtain the patterns‘ description without ambiguity and perform analysis in them. Finally, a case study will be defined in order to apply mobile agent design patterns combinations, and from them, analysis will be performed, verifying which combinations have better performance in each of the assessed conditions.

Agradecimentos

À minha família, pela oportunidade e condições de realizar este trabalho.

A Aglair, pelo amor, apoio e paciência constantes, mesmo estando geralmente distante.

A todos os amigos de Maceió que sempre estiveram torcendo por mim.

Aos amigos do LabPetri, por tornarem prazeroso o trabalho do dia-a-dia.

A Flávio, Jaime e Laísa, que se envolveram mais diretamente com este trabalho, pelos conhecimentos compartilhados e auxílio sempre constante.

Aos grandes companheiros "campinenses" Amancio, Cássio, Cidinha, Cláudia, Ely, Érica, Glaucimar, Juliana, Massillania, Nazareno, Patrícia, Sandro e tantos outros que me ajudaram nesses dois anos de mestrado e com os quais eu aprendi bastante.

A Aninha, pela amizade e auxílio sempre constantes.

Aos professores, funcionários, colegas do DSC e a todos aqueles que colaboraram para a realização deste trabalho.

Aos meus orientadores, Patrícia Machado e Jorge Abrantes, pela dedicação, apoio e incentivo, e pela confiança de que poderíamos realizar um bom trabalho.

Aos membros da Banca Examinadora pela atenção e ricas sugestões.

À CAPES, pelo apoio financeiro.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização e Motivação

O crescente uso da Web associado ao desenvolvimento de tecnologias de computação móvel definiu um novo cenário para o desenvolvimento de sistemas. Neste cenário, os serviços devem ser providos por aplicações que possam ser configuradas e adaptadas dinamicamente aos diversos perfis de usuário, em um ambiente global e heterogêneo.

Computação móvel amplia a noção de computação distribuída convencional. Dispositivos podem ser movidos de uma rede para outra de forma dinâmica (mobilidade física) e programas podem interromper a sua execução em uma máquina e se mover para outros nós da rede a fim de realizar uma tarefa (mobilidade virtual). Com isso, elimina-se a necessidade de conectividade fixa a um determinado nó da rede, dando mais flexibilidade a usuários com capacidade de mobilidade e liberando, por exemplo, a necessidade de manter ativas conexões intermitentes e/ou de alto custo. De uma forma geral, mobilidade introduz problemas inexistentes na computação distribuída tradicional: variações nas condições de comunicação, gerenciamento de energia e restrição de recursos, já que estes terão que ser realocados sempre que o dispositivo se mover. Pode-se dizer que computação móvel é o pior cenário de um sistema distribuído onde problemas de comunicação e desconexão são constantes e a topologia do ambiente é dinâmica [Car99; FPV98]. Métodos e técnicas convencionais para o desenvolvimento de sistemas distribuídos não têm conseguido lidar de forma satisfatória com as características desta nova demanda.

Diante deste cenário, agentes móveis apresentam-se como uma alternativa a mecanismos

convencionais utilizados no desenvolvimento de aplicações distribuídas. Agentes móveis podem ser definidos como programas autônomos que podem migrar entre computadores de uma rede durante sua execução, carregando consigo o seu estado de execução.

O uso de agentes móveis pode trazer várias vantagens para a aplicação e para o usuário. Vantagens como redução do tráfego na rede, pois as interações poderão ser realizadas localmente, o que conseqüentemente diminui a dependência da latência da rede; o agente móvel tem execução assíncrona e autônoma, não existindo um controle central e permitindo que o cliente se desconecte enquanto o agente enviado realiza determinada tarefa. Estando desconectado, o dispositivo do cliente pode, por exemplo, ser desligado, evitando um gasto desnecessário de energia, principalmente para dispositivos que dependam de bateria; o agente pode ter a capacidade de perceber mudanças no ambiente de execução e reagir de forma autônoma, o que pode tornar mais fácil a construção de sistemas distribuídos robustos e tolerantes a faltas. Esta capacidade de perceber o ambiente e agir de forma autônoma, pode ser importante para o agente reagir diante de diferentes condições de comunicação ou diante de restrição de recursos. Pode-se argumentar que cada uma destas vantagens seria obtida com outros paradigmas, o que é verdade. Mas a grande vantagem dos agentes móveis é a soma destas vantagens, uma vez que elas são conseguidas de forma conjunta usando um único paradigma, como exposto em [HCK95].

Uma desvantagem do uso de agentes móveis é a necessidade de instalação de um ambiente para execução do agente em cada computador que o agente visitar. É este sistema, chamado de plataforma, que dará suporte ao funcionamento dos agentes, provendo funcionalidades como criação, migração e destruição de agentes. Além disso, deve-se ter cuidado para não tornar o agente muito pesado, devido ao excesso de código ou de dados, pois isso pode tornar a migração mais lenta e aumentar o tráfego na rede.

Apesar de existirem várias áreas potenciais para aplicação de agentes móveis, como comércio-eletrônico, assistentes pessoais e recuperação de informações, ainda existe uma certa resistência em aplicar o paradigma. Isso pode se dever à motivos como: falta de uma aplicação chave, que só possa ser desenvolvida com agentes móveis; falta de segurança, tanto em relação à proteção do agente quanto em relação à proteção do ambiente de execução do agente; e dificuldade no desenvolvimento de sistemas usando agentes móveis. Esta dificuldade existe pois, além destas aplicações serem mais complexas, elas têm sido desenvolvidas

de forma *ad-hoc*, já que a maioria das metodologias voltadas para orientação a objetos não contemplam aspectos importantes da mobilidade. Algumas abordagens em direção a uma metodologia para o desenvolvimento de sistemas multi-agentes, como [MKC01], têm sido propostas. Contudo, a mobilidade ainda não foi considerada de forma apropriada no processo de desenvolvimento de software como um todo. Em [Ode01], por exemplo, é proposta uma notação para desenvolvimento de aplicações baseadas em agentes que só considera a característica de mobilidade nos diagramas de implantação.

Neste sentido, têm sido realizados alguns importantes esforços para modelagem de sistemas baseados em agentes móveis, focando em atividades específicas, técnicas e notações [TOH99; Car99; KRSW01; CLZ01]. Em [GMM03], um modelo de desenvolvimento para aplicações baseadas em agentes móveis baseada no Processo Unificado [Lar99] é introduzido. Este modelo é baseado nas idéias, apresentadas por Tahara [TOH99], de projetar aplicações baseadas em agentes móveis aplicando padrões de acordo com o nível arquitetural, onde, para maximizar o reuso, o nível mais alto é independente de especificidades das plataformas de agentes. Ambas abordagens são baseadas no uso de padrões de projeto para agentes móveis.

Um padrão de projeto descreve um problema que ocorre com certa freqüência e propõe uma solução, que também ocorre com certa freqüência, para este, permitindo a sua utilização em vários projetos diferentes que encontrem o mesmo problema. O uso de padrões facilita o desenvolvimento de aplicações e as torna mais flexíveis, mais compreensíveis e reusáveis. Sem padrões significativos, o desenvolvimento de aplicações requer altos custos. Isso é ainda mais crítico para o desenvolvimento de aplicações baseadas em agentes móveis, devido à complexidade inerente de construir tais aplicações. Dessa forma, os padrões de projeto podem ser usados também para as aplicações de agentes móveis, trazendo todas as suas vantagens e facilitando o desenvolvimento do projeto destas aplicações.

1.2 Declaração do Problema

Alguns trabalhos têm proposto padrões de projeto para agentes móveis ([AL98; AD98; TOH99; DOKO99; TTOH01; JK99; IKV]), no entanto, estes ainda apresentam alguns problemas. Um primeiro problema é que os padrões têm sido sugeridos de forma isolada,

quando se deveria demandar maior esforço para tentar relacioná-los, evitando a proposição de padrões repetidos, e permitindo que estes possam ser aplicados juntos, criando, dessa forma, linguagens de padrões[1]. Uma classificação unificada para os padrões facilitaria a definição de relação entre eles, porém ainda não se tem essa classificação definida e cada autor classifica seus padrões como achar mais interessante.

Um cuidado que se deve ter é tentar não prender padrões a especificidades de plataformas, tentando defini-los para que eles possam ser aplicados em qualquer plataforma. Para isso, são importantes os estudos de caso, mostrando a viabilidade de implementação dos padrões nas mais variadas plataformas. O desenvolvimento seguindo modelos que tem a fase de projeto baseada na utilização de padrões para agentes móveis, como [TOH99] e [GMM03], se torna praticamente inviável sem essa catalogação.

Um último problema é a forma de descrição dos padrões, já que alguns são descritos com texto, outros com figuras e outros com diagramas UML. Estas formas de descrição nem sempre facilitam o entendimento do padrão, levando até a entendimentos equivocados. Uma idéia para tentar descrever os padrões, facilitando assim o entendimento, seria usar algum método formal, que teria a vantagem de não apresentar ambigüidade. Existem vários formalismos que vêm sendo utilizados para modelar agentes móveis e sistemas de agentes [AMM01; MCM01; XD00; SMT98; VBML99], mas poucos trabalhos, como [RB99; RW00], têm sido feitos em relação a padrões para agentes móveis. Um formalismo interessante, tanto pelo seu poder de modelagem, quanto pela disponibilidade de representações gráficas, são as redes de Petri. Com redes de Petri poder-se-ia, além de modelar, simular, verificar propriedades dos padrões e analisar desempenho de propriedades como número de mensagens trocadas ou tempo de execução de uma determinada tarefa remota. Dessa forma, uma vez modelados, os padrões poderiam ser combinados e analisados em várias situações, para verificar que combinações de padrões funcionariam melhor em cada situação, e assim, poder-se-ia iniciar o trabalho de definição de linguagens de padrões, apontados como uma grande necessidade em [DW99] e [DOKO99].

Diante do exposto, o problema tratado neste trabalho é a falta de organização dos padrões de projeto para agentes móveis que têm sido propostos, que pode dificultar o en-

[1]Linguagem de padrões define um grupo de padrões que podem ser combinados para solução de um determinado tipo de aplicação.

tendimento e a utilização dos mesmos.

1.3 Objetivo

O objetivo principal do nosso trabalho é a classificação e formalização de padrões de projeto para agentes móveis e análise de desempenho de combinações destes padrões, a fim de facilitar a sua utilização no desenvolvimento de aplicações deste tipo.

Para atingimos este objetivo, um levantamento dos padrões de projeto para agentes móveis que têm sido propostos foi realizado. A partir deste levantamento, uma classificação e uma forma de documentação para estes padrões foi proposta. Nesta fase, foram construídos os modelos independentes e dependentes, conforme proposto em [GMM03]. Em seguida, estes padrões foram formalizados, o que permitiu que os mode foi definido. Para ele, foram propostas soluções utilizando combinações dos padrões de projeto para agentes móveis catalogados. Estas soluções foram formalizadas, utilizando os modelos formalizados anteriormente, e analisadas para verificação de como cada uma delas se comporta diante de alguns cenários.

1.4 Resultados e Relevância

Nosso trabalho deverá trazer vários resultados. Um primeiro resultado é a catalogação e classificação de padrões de projeto para agentes móveis, que servirá para organizar os diversos padrões que têm sido propostos, facilitando referências aos mesmos e tornando mais fácil a identificação de que padrões há disponíveis para cada situação (comunicação, segurança, entre outras). Com isso, o uso desses padrões seria incentivado, o que facilitaria o desenvolvimento com modelos que se baseiam no uso de padrões, como [Gue02a], podendo impulsionar um maior uso deste paradigma. Além disso, essa catalogação poderá contribuir para que novos padrões que venham a ser definidos sejam classificados de acordo com as categorias propostas.

Um segundo resultado é em relação à modelagem dos padrões para várias plataformas de agentes móveis, seguindo o modelo de [Gue02a], que permite a definição de um projeto independente de plataforma, que não considera especificidades das plataformas, e projetos

dependentes específicos para cada plataforma. Como mostrado em [LMdFS04], esta divisão é motivada não só por promover o reuso do projeto, mas também porque deverá ser necessário que os padrões de projeto para agentes móveis precisem ser aplicados em plataformas diferentes. Ademais, devido à complexidade de implementar soluções baseadas em agentes móveis, a visão independente de plataforma, pode facilitar um melhor entendimento global e análise da qualidade da solução apresentada, mesmo se apenas uma plataforma é considerada [LdFG04].

Um terceiro resultado é a formalização dos padrões com redes de Petri, que nos permitirá fazer simulações e análise dos modelos, identificando e corrigindo possíveis falhas na definição dos padrões. Além disso, a formalização é uma forma de documentação dos padrões que tem a vantagem de não apresentar ambigüidade.

Um último resultado é a análise de desempenho das combinações dos padrões. Analisando as diversas soluções para o estudo de caso, usando os modelos de redes de Petri, podemos verificar que combinações se comportam melhor em cada situação, como por exemplo, diferentes larguras de banda da rede ou tamanho do agente. Assim, estas combinações e a análise poderão contribuir para que o desenvolvedor saiba que combinação utilizar de acordo com as condições de sua aplicação.

1.5 Estrutura da Dissertação

O restante deste documento foi organizado da seguinte forma:

Capítulo 2: Agentes Móveis Neste capítulo será apresentada a fundamentação teórica necessária para a compreensão de agentes móveis. Nesta fundamentação, serão apresentados conceitos e notações utilizadas em nosso trabalho, bem como as vantagens e desvantagens do uso de agentes móveis. Também mostraremos áreas de aplicação e plataformas de suporte ao funcionamento dos agentes. Por fim, os problemas no desenvolvimento de aplicações baseadas em agentes móveis serão expostos.

Capítulo 3: Padrões de Projeto para Agentes Móveis Apresentaremos os padrões de projeto de uma forma geral, definindo conceitos e motivando o uso destes. Em seguida, os padrões de projeto para agentes móveis são apresentados, e alguns problemas destes padrões

serão discutidos. Uma classificação e catalogação dos padrões é proposta, juntamente com um formato de apresentações dos padrões.

Capítulo 4: Formalização de Padrões de Projeto para Agentes Móveis Este capítulo apresentará trabalhos que têm sido realizados em relação à formalização de agentes móveis e de padrões de projeto. Tendo sido escolhido como formalismo para modelagem dos padrões para agentes móveis, as redes de Petri serão introduzidas, apresentando a fundamentação teórica sobre este formalismo. Por fim, serão mostrados os padrões de projeto para agentes móveis formalizados com redes de Petri, bem como a análise destas formalizações.

Capítulo 5: Estudo de Caso No Capítulo 5, um estudo de caso será apresentado e soluções de modelagem para este serão propostas. Estas soluções serão realizadas através da combinação de padrões de projetos para agentes móveis e serão modeladas através da combinação dos padrões formalizados anteriormente. Será mostrada a análise de desempenho das soluções definidas no capítulo anterior. A seguir, mostraremos a comparação das análises, buscando concluir que combinações se comportam melhor em cada situação (tamanho do agente, tamanho dos dados, capacidade da rede, entre outros).

Capítulo 6: Considerações Finais Baseado nos resultados obtidos, o último capítulo será destinado à apresentação das conclusões e da perspectiva de trabalhos futuros.

Apêndice A No Apêndice A, é apresentado um catálogo dos padrões de projeto para agentes móveis, de acordo com o proposto neste trabalho.

Apêndice B No Apêndice B, as formalizações de alguns padrões de projeto para agentes móveis são apresentadas.

Capítulo 2

Agentes Móveis

Este capítulo tem o objetivo de familiarizar o leitor com os principais conceitos do paradigma de agentes móveis. Nele, primeiramente serão definidos os agentes móveis, juntamente com os conceitos sobre os quais este paradigma se apóia. Também serão apresentados os sistemas de agentes móveis, explicando seus principais conceitos, funcionalidades e estado da arte, bem como os esforços que têm sido empreendidos em direção à interoperabilidade destes sistemas. Em seguida, são apresentadas possíveis aplicações para os agentes móveis, assim como vantagens e desvantagens do uso destes agentes. O desenvolvimento das aplicações baseadas em agentes móveis é tratado no final do capítulo, onde são expostos alguns problemas no desenvolvimento e soluções que têm sido propostas.

2.1 Definição

O paradigma de agentes móveis está apoiado na junção de dois conceitos: agentes e mobilidade de código. Em relação ao termo agente não existe uma definição única. Podemos dizer que agentes são entidades autônomas, incumbidas de realizar alguma tarefa, podendo reagir às mudanças no ambiente em que estão inseridos e que podem possuir características como inteligência, cooperação e mobilidade. Já por mobilidade de código, podemos entender a capacidade de alterar, dinamicamente, as ligações entre fragmentos de código e a localização onde ele é executado [FPV98]. É um conceito antigo que já vem sendo utilizado desde os anos 70 em aplicações com entrada de jobs remotos e impressoras *PostScript*, onde o código era enviado de para ser executado em uma máquina remota.

Desta forma, pode-se definir Agente Móvel como um programa autônomo de computador que atua em interesse de uma outra entidade, fornecendo-lhe o suporte necessário, e que é capaz de migrar entre diferentes localizações físicas das redes, executando suas tarefas localmente, e podendo continuar a sua execução do ponto onde parou antes da migração.

Na próxima seção, os sistemas de código móvel, no qual agentes móveis se baseiam, serão introduzidos. Em seguida, conceitos relacionados aos agentes móveis serão apresentados.

2.2 Sistemas de Código Móvel

Sistemas de código móvel são aqueles que se baseiam nas técnicas e nos mecanismos de migração originalmente utilizados em nível de sistema operacional. Esses sistemas têm importantes características como o fato de poderem ser projetados para redes de grande porte, poderem ser executados em ambientes heterogêneos, a localização dos componentes é conhecida pela aplicação e a mobilidade está sob o controle do programador.

A mobilidade de código permite re-alocar uma unidade de execução para diferentes ambientes computacionais. Assim, existem alguns padrões que definem a coordenação e realocação dos componentes necessários para realização de um serviço. Resumidamente, cada padrão determinará a localização dos componentes antes e depois do serviço, o componente responsável pela execução do serviço e o local onde o serviço será executado de fato. Estes padrões, apresentados a seguir, oferecem várias abstrações para representar as ligações entre componentes, locais de execução, e código, e suas re-configurações dinâmicas.

Avaliação Remota

Neste padrão um ambiente computacional possui o código para execução do serviço, mas não possui os recursos para tal. Então o código é enviado para um ambiente que possua os recursos, que realizará o serviço, retornando o resultado para o ambiente que enviou o código.

Código sob demanda

Aqui nós temos o inverso do caso anterior. O ambiente possui o recurso para execução do serviço, mas não possui o conhecimento (*know-how*) para execução do mesmo. Então, o código para execução do serviço é transferido de outro ambiente computacional, de forma que o primeiro ambiente execute o serviço.

Agentes Móveis

Neste caso, um computador possui um componente de computação com um determinado código que, apesar de possuir alguns recursos, necessita de recursos que estão em outro computador. Então o componente (agente móvel) é re-alocado (migra) para o outro computador, utiliza os recursos e volta para o computador de origem, sem a necessidade da interferência de um componente de computação no computador que possui os recursos. A migração do agente móvel pode ser realizada por duas formas de mobilidade:

Mobilidade Forte Na mobilidade ou migração forte, tanto o código quanto o estado de execução de uma unidade de execução é re-alocado para um ambiente computacional diferente. Assim, a unidade de execução é suspensa no ambiente de origem, transmitida para o ambiente destino, onde torna a ser executada.

Mobilidade Fraca Neste tipo de mobilidade, o agente migra levando com ele apenas o seu estado de dados. O estado de dados de um agente consiste nos valores de variáveis internas serializados antes da migração, transferidos através da rede. Após a migração, o agente é reiniciado, os valores das variáveis são restaurados, mas a execução é efetuada novamente a partir do início.

2.3 Sistema de Agentes Móveis

Nesta seção, definiremos alguns conceitos relacionados aos sistemas de agentes móveis, conforme definido em [Fok98]. Estes conceitos facilitarão o entendimento do funcionamento destes sistemas, bem como dos agentes móveis.

Sistema de agentes. Um sistema de agentes é uma **plataforma** que pode criar, executar, transferir e terminar agentes. Cada sistema de agentes funciona como uma máquina virtual para lugares e seus agentes, e é identificado de modo único por seu nome e endereço, sendo associado com um nome de autoridade que identifica a pessoa ou organização para quem o sistema de agentes age. A instanciação de um sistema de agentes é chamada de **agência**, assim, cada computador pode ter uma ou mais agências.

Lugar. Lugar é o ambiente lógico de execução dos agentes. É esse contexto que disponibiliza um conjunto de serviços (recursos) que o agente pode utilizar. Em uma migração, o agente se transfere entre diferentes lugares, que podem estar dentro de uma mesma agência ou em agências diferentes.

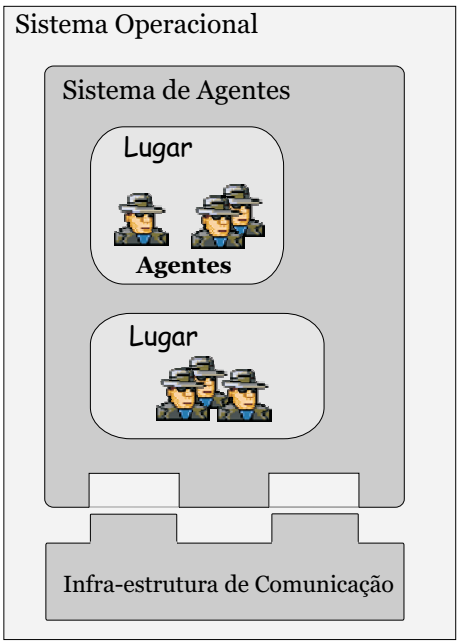


Figura 2.1: Arquitetura de um sistema de agentes

Infra-estrutura de comunicação. A infra-estrutura de comunicação (IC) é responsável por toda a comunicação entre os sistemas de agentes. Toda ela é feita através da IC, que também possui um serviço de nomes dos agentes e define as regras de segurança em nível de comunicação. A Figura 2.1 mostra a arquitetura de um sistema de agentes juntamente com infra-estrutura de comunicação.

Região. Região é um conjunto de sistemas de agentes sob a mesma autoridade. O conceito de região permite que mais de um sistema de agentes represente a mesma pessoa ou organização. O administrador da região define os serviços para a comunicação dentro de uma região e entre regiões. Os sistemas de agentes e clientes que não pertencem a região, à acessam através de sistemas de agentes, chamados pontos de acessos da região, que estão expostos ao mundo externo, como em um *firewall*.

A arquitetura de uma região é mostrada na Figura 2.2.

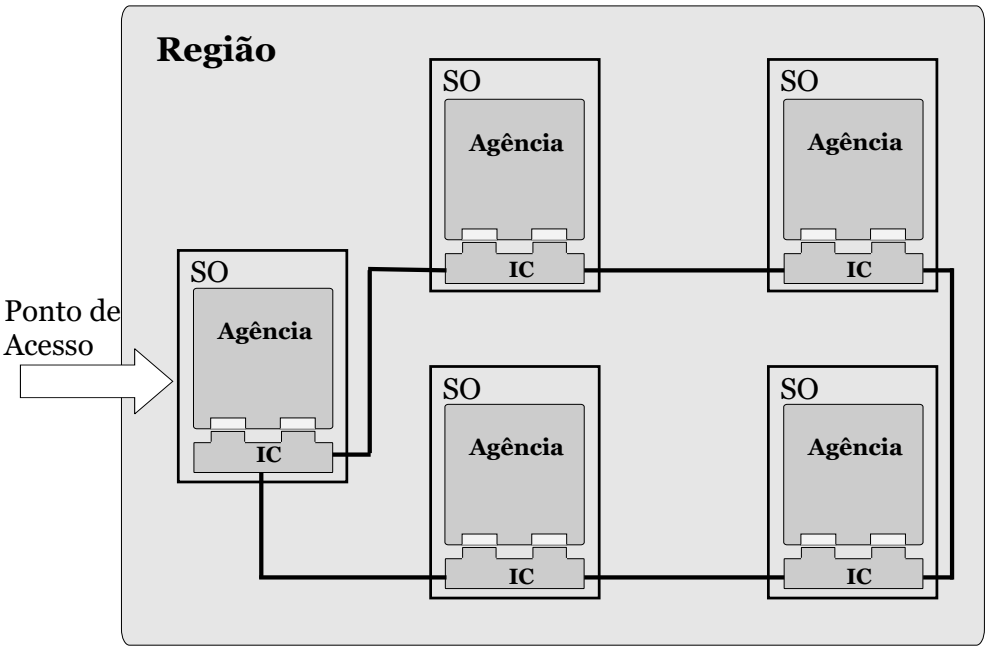


Figura 2.2: Arquitetura de uma região

2.4 Funcionalidades de um sistema de agentes móveis

As principais funcionalidades que devem estar presentes em um sistema de agente móveis são: criação, destruição, transferência e busca de agentes, provimento de nomes e localizações únicas para os agentes, suporte ao conceito de região e garantia de um ambiente seguro para as operações dos agentes. A seguir, todas estas funcionalidades são detalhadas, conforme proposto em [Fok98].

Criar agentes

Um agente é criado em um lugar por um agente do mesmo lugar ou por outro agente ou outro objeto fora do seu lugar. Antes, o criador deve ser autenticado pelo lugar onde pretende criar o agente, estabelecendo a autoridade e credenciais para o novo agente, e podendo ainda definir argumentos de inicialização para este. As definições de classe necessárias para instanciar o agente podem estar na máquina local ou remota. Abaixo nós temos os passos para a criação de um agente.

1. **Instanciação e identificação** A definição da classe é carregada, tornada executável e o objeto agente é instanciado. O lugar designa um identificador único para o agente.

2. **Inicialização** O agente se inicializa usando os argumentos fornecidos pelo criador. Quando a inicialização se completar, o agente assume que foi total e corretamente estabelecido no lugar (instanciado, identificado e inicializado).

3. **Execução autônoma** Depois de ser completamente estabelecido no lugar, o agente começa sua execução, sendo capaz de executar independente dos outros agentes no mesmo lugar.

Destruir agentes

A destruição de um agente pode ser iniciada pelo próprio agente, por outro agente do mesmo lugar, ou por outro agente ou não-agente fora do lugar. O agente pode ser destruído pelo sistema de agentes devido a uma das razões seguintes:

- o tempo de vida do agente expirou;
- ninguém se refere ou usa o agente;
- o agente violou alguma regra de segurança;
- o sistema de agentes foi desligado.

Na destruição de um agente, primeiramente o sistema permite ao agente finalizar alguma tarefa que esteja realizando, depois suspende a execução do agente.

Transferir agentes

O processo de transferência pode ser iniciado pelo próprio agente, por outro agente do mesmo lugar, ou por outro agente ou objeto de fora do lugar. Neste processo, o agente é remetido do lugar atual para um lugar especificado.

A transferência é gerenciada pelos lugares de origem e de destino. Quando a origem entrar em contato com o destino, o lugar de destino pode completar o pedido ou retornar uma indicação de falha para a origem. Se a origem não puder contatar o destino, ele deve retornar uma indicação de falha para o agente.

Remetendo um agente Primeiramente o agente deve estabelecer o lugar de destino, para então informar ao sistema de agentes que ele deve ser transferido. O sistema de agentes recebe o pedido e faz o seguinte:

1. suspende o agente: o agente é avisado da transferência e é preparado para a migração. Então o processo do agente é suspenso;

2. serializa o agente: o estado e a classe do agente são serializados pelo sistema de agentes. Serializar é o processo de criar uma representação consistente do objeto agente, incluindo o estado de execução, que pode ser transportada através de uma rede;

3. codifica o agente serializado: o sistema de agentes codifica o agente serializado para o protocolo de transporte escolhido;

4. transfere o agente: o sistema de agentes estabelece uma conexão de rede para a máquina destino e transfere o agente serializado codificado.

Recebendo um agente Antes do sistema de agentes receber um agente, ele deve verificar se o agente pode ser aceito por aquela máquina. Somente depois do remetente se autenticar a transferência se inicia e o sistemas de agente faz o seguinte:

1. recebe o agente: quando o sistema de agentes concorda com a transferência, o agente codificado é recebido;

2. decodifica o agente: o sistema de agentes decodifica o agente serializado;

3. desserializa o agente: a representação persistente do agente é desserializada. A classe do agente é instanciada e o estado do agente transferido é restaurado;

4. reinicia a execução do agente: o agente re-criado é avisado de sua chegada no lugar de destino. Ele pode agora se preparar para reiniciar sua execução.

A Figura 2.3 ilustra a transferência de um agente através da rede.

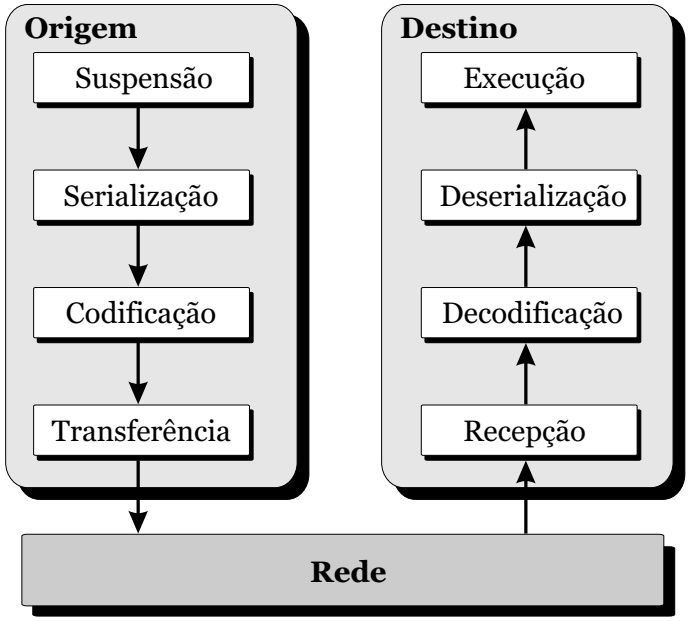


Figura 2.3: Transferência de um agente

Transferência da classe do agente O agente não pode reiniciar sua execução no destino sem que sua classe esteja presente. Existem várias formas de tornar a classe disponível para o sistema de agente destino:

- Classe no destino: Se a classe já estiver no destino não há necessidade de transferência. Apenas informações para identificar a classe, como o nome da classe e seu descritor.

- Classe na origem: Se a classe estiver localizada na origem, o que geralmente ocorre, ela pode ser facilmente transportada com o estado do agente para o sistema de agentes destino.

- Código sob demanda: A classe está disponível a partir de um servidor e o sistema de agentes destino pode recuperar a classe usando os fundamentos do código sob demanda.

  Depois de o agente ser instanciado, ele geralmente cria objetos, que precisam de suas classes para ser instanciados e continuar a execução. Se alguma dessas classes não estiver disponível no sistema de agentes destino, ela deve ser transferida da origem do agente. Pode-se escolher entre transferir todas as classes possivelmente necessárias juntamente com a classe do agente, ou transferir as classes à medida que for sendo necessária a sua instanciação.

Prover nomes e localizações únicas para os agentes

O sistema de agentes deve prover um nome único para si próprio, e para os lugares que ele cria. Também deve gerar um nome único para os agentes que ele criar. Geralmente esses nomes são baseados no IP.

Encontrar um agente móvel

A habilidade de localizar um agente móvel é importante para o gerenciamento de agentes, pois quando um agente quer se comunicar com outro, ele deve ser capaz de localizar o sistema de agentes destino e estabelecer a comunicação.

Como um agente móvel migra, o nome do agente dever ser único em todos os sistemas de agentes em uma determinada região. O sistema de agentes deve prover um serviço de nomes baseado nos nomes dos agentes.

Suportar o conceito de região

Um sistema de agentes suporta o conceito de região cooperando com outros sistemas de agentes com a mesma autoridade e suportando pontos de acesso da região. O ponto de acesso da região está encarregado de encaminhar solicitações externas para os sistemas de agentes da região.

Garantir um ambiente seguro

Como um agente móvel é um programa que migra entre sistemas de agentes, ele é bastante comparado a um vírus. Desta forma, é muito importante que os sistemas de agentes identifiquem e classifiquem agentes que estão chegando. Um sistema de agentes deve proteger seus recursos, incluindo o sistema operacional, arquivos de sistema, discos, CPU, memória, outros agentes e acessos a programas locais.

2.5 Plataformas e Interoperabilidade

Várias plataformas para sistemas de agentes móveis estão disponíveis, e muitas outras estão sendo desenvolvidas, sendo a maioria delas implementadas em Java [Sun]. Estas plataformas apresentam algumas coisas em comum, além da linguagem, como contarem com a Máquina Virtual Java em seu funcionamento e usarem o mecanismo de serialização de objetos de Java. Estas plataformas suportam, em geral, apenas a migração fraca, devido a limitação da máquina virtual java, que não permite o que a pilha de execução do agente seja transmitida, sendo a migração forte apenas simulada. Desta forma, a simulação da migração forte, fica a cargo do programador, o que não ocorreria se a mesma fosse suportada pela plataforma.

Exemplos de plataformas são TeleScript [Whi96], a primeira plataforma comercial, AgenteTCL[TCL], que foi re-implementada em Java com o nome de D'Agent [D'A], Aglets [LO98], que possui o código aberto, Grasshopper [Gmb01], primeira plataforma de agentes que está de acordo com os padrões da OMG MASIF [Fok98] e da FIPA [FIP] (apresentados a seguir), e JADE [BCTR03], que está de acordo com os padrões da FIPA e possui suporte para J2ME.

Todas estas plataformas diferem em relação à implementação e arquitetura, o que torna muito difícil a interoperabilidade entre esses sistemas. Neste sentido, existe a necessidade de padronização de alguns aspectos para tornar possível a interação entre os mais diversos sistemas. Esta padronização vem sendo discutida por duas organizações: a FIPA [FIP] e a OMG [OMG].

As especificações da FIPA representam o primeiro passo em direção à padronização de agentes. Elas não tentam descrever a estrutura interna dos agentes nem como eles devem ser implementados, mas são especificadas as interfaces necessárias para dar suporte à intero-

perabilidade entre eles. Para tanto, as formas de comunicação entre os agentes, incluindo estrutura e forma de transporte das mensagens, são padronizadas. O foco principal da FIPA é uma linguagem de comunicação dos agentes, chamada ACL (*Agent Communication Language*). A FIPA também define uma arquitetura de sistema de agentes (não necessariamente móveis) contendo três elementos fundamentais:

- *Agent Management System (AMS)* - Responsável pela gerência do sistema de agentes, controlando o acesso e o uso da mesma.
- *Directory Facilitator* - Provê o serviço de páginas amarelas para a plataformas.
- *Agent Communication Channel (ACC)* - controla a troca de mensagens em uma plataforma, incluindo as mensagens trocadas com outras plataformas.

Já a OMG, através das especificações da MASIF (Mobile Agent System Interoperability Facilities), trata da interface entre sistemas de agentes e não entre aplicações de agentes e sistemas de agentes. A preocupação da MASIF é prover interfaces de acesso a mecanismos de controle (operações de criação, destruição, transferência, suspensão e resumo), monitoramento (localização de agentes), transporte de agentes entre agências, e mecanismos de segurança. MASIF não trata da interoperabilidade de linguagens, porque seria uma questão muito difícil, estando limitada à interoperabilidade entre sistemas de agentes escritos na mesma linguagem, mas provavelmente por fornecedores diferentes. Além disso, ela não tenta padronizar operações locais dos agentes como interpretação, serialização ou execução de agentes.

2.5.1 Escolha das Plataformas

Neste trabalho, serão utilizadas três plataformas de suporte aos agentes móveis. Os critérios para escolha das plataformas e as principais características destas são apresentados a seguir.

O primeiro critério foi que a plataforma deveria ter distribuição completa e gratuita por tempo indeterminado, não oferecendo, por exemplo, períodos de alguns dias para avaliação da plataforma ou versões mais simplificadas das mesmas. Assim, gastos desnecessários seriam evitados, uma vez que vários grupos permitem o uso de suas plataformas sem cobrar por esse uso. Um outro critério foi que a plataforma deveria funcionar nos mais variados

sistemas operacionais, como variações do UNIX e versões do Windows. A linguagem de programação dos agentes na plataforma deveria ser Java, por ser uma linguagem orientada a objeto e portável, rodando nos mais diversos sistemas operacionais e tendo versões que garantem o seu funcionamento em configurações mínimas de hardware. Um último critério foi o suporte à plataforma dado pelo grupo da mesma, com documentação, API do framework, fóruns ou listas de discussão e lista de problemas conhecidos. Como o tempo para realização do trabalho era curto, esse suporte seria fundamental, podendo facilitar, ou evitar maiores dificuldades quando da modelagem dos padrões. Assim, as plataformas Grasshopper[1][Gmb01], Aglets[2][LO98] e JADE[3][BCTR03] foram escolhidas.

A plataforma Grasshopper é uma plataforma de desenvolvimento de agentes, fornecida pela IKV++, que é compatível com as definições da MASIF[Fok98]. Esta plataforma é baseada na linguagem Java e é construída sobre um ambiente de processamento distribuído para prover interoperabilidade entre plataformas de agentes móveis de diferentes fabricantes e também tecnologias do paradigma cliente/servidor. Aglets é baseada na idéia da criação de *Applets* Java especiais, chamados de aglets, que são capazes de migrar de uma máquina para outra em uma rede. Ela possui uma biblioteca de suporte chamada Aglets/API. Um protocolo chamado *Agent Transfer Protocol* (ATP) é usado para provê comunicação e mobilidade dos agentes. JADE (Java Agent DEvelopment Framework) é uma plataforma completamente implementada em Java que simplifica a implementação de sistemas multi-agentes pela existência de um *midleware* baseado nas especificações de FIPA [FIP] e de várias ferramentas de suporte ao desenvolvimento de aplicações. Recentemente JADE foi integrada com os resultados do projeto LEAP (*Lightweigth Extensible Agent Platform*), que pretende desenvolver plataformas leves e executáveis em dispositivos pequenos, como PDAs e telefones.

Estas três plataformas oferecem funcionalidade semelhantes, porém com diferentes nomenclaturas, definições e estruturas. Com relação à estrutura, por exemplo, em JADE, cada computador possui um ambiente de execução, onde os *containers* são iniciados (localmente ou remotamente). Os agentes são executados nestes *containers*. Por outro lado, em Grasshopper existem regiões, reunindo agências, que, por sua vez, possuem lugares onde os agentes são executados. Já em Aglets, existem os contextos (*AgletContext*), onde os agentes

[1]http://www.grasshopper.de

[2]http://aglets.sourceforge.net

[3]http://jade.cselt.it

são executados.

Outra diferença observa-se no fato de o Grasshopper fazer distinção entre agente móvel e estacionário, enquanto em JADE e Aglets todos os agentes herdam a característica de mobilidade. JADE apresenta o conceito de comportamentos (*behaviours*), que são classes implementando funcionalidades que são adicionadas ao agente dinamicamente, como por exemplo, um comportamento para receber mensagens ou executar uma determinada tarefa. Nas plataformas Aglets e Grasshopper, a classe do agente possui um método abstrato, que as subclasses devem implementar para definir o comportamento do agente.

2.6 Áreas de Aplicação

Vários estudos de caso têm demonstrado a aplicabilidade dos agentes móveis [BM00; GOP02]. A seguir, temos a descrição de algumas das principais áreas de aplicação:

Assistência Pessoal

A habilidade que os agentes móveis têm de executar em máquinas remotas, os torna adequados para prestar uma assistência capaz de realizar tarefas na rede em benefício do seu criador. O assistente remoto vai operar independentemente de sua conexão limitada e o criador fica livre para desligar o computador. Por exemplo, para agendar um encontro com várias pessoas, o usuário poderia mandar agentes móveis para interagir com o agente de cada pessoa convidada para o encontro. Os agentes negociariam e estabeleceriam a hora do encontro.

Recuperação de informação distribuída

Representa um agrupamento de informações satisfazendo determinados critérios a partir de fontes de informação dispersas através da rede. A informação a ser recuperada pode ser definida estaticamente ou dinamicamente durante o processo de recuperação. Sendo assim, um agente, por exemplo, pode ser encarregado de viajar pela rede, obtendo as informações a serem recuperadas.

Controle de dispositivos remotos

Na abordagem tradicional são feitos o monitoramento via coleta periódica do status dos dispositivos e a configuração através de um conjunto pré-definido de serviços. Com agentes móveis haveria co-locação de componentes de monitoramento junto aos dispositivos monitorados para reportarem a evolução do status dos dispositivos. Além disso,seriam enviados componentes de gerência para configurar dispositivos remotos localmente, permitindo maior desempenho e flexibilidade.

Gerenciamento de *Workflow*

Workflow define que atividades devem ser cumpridas em uma determinada tarefa, bem como de que maneira, onde e quando estas atividades devem ser executadas. Agentes móveis podem ser usados para prover suporte à mobilidade de atividades que encapsulem suas definições e seu estado. Por exemplo, um componente móvel pode encapsular um documento texto que suporta várias versões. O componente gerencia a informação sobre o estado do documento, que operações podem ser feitas, e qual o próximo passo para revisão.

Comércio Eletrônico

As aplicações de comércio eletrônico permitem usuários desenvolver aplicações comerciais através da rede. Uma transação pode envolver negociação com entidades remotas e requisição de informações. Portanto, é desejável que o componente móvel da aplicação forneça informações relevantes para o negócio. Aplicações deste tipo podem ser implementadas com a tecnologia de agentes móveis, já que a migração de componentes da aplicação para locais próximos às fontes de informação relevantes para a transação causará aumento de desempenho e suporte a operações desconectadas.

2.7 Vantagens e Desvantagens

O uso de agentes móveis pode trazer várias vantagens para a aplicação. Algumas dessas vantagens, elencadas por [LO99], são mostradas abaixo. Em seguida, algumas desvantagens

trazidas pela aplicação dos agentes móveis são apresentadas.

Vantagens

Redução do tráfego na rede Sistemas distribuídos demandam um grande volume de comunicação (interação) para realizar tarefas, principalmente quando há restrições de segurança envolvidas. Agentes móveis podem reduzir o tráfego da rede, pois permitem despachar tarefas que podem executar suas interações localmente. Agentes móveis podem ainda reduzir o tráfego de dados da rede, pois permitem mover o processamento para o local onde os dados estão armazenados ao invés de transferir os dados para depois processá-los. O princípio é simples: "Mover o processamento para os dados ao invés de mover os dados para o local de processamento".

Diminui a dependência da latência da rede Sistemas críticos necessitam de respostas em tempo real para mudanças no ambiente. O controle desses sistemas através de uma rede substancialmente grande ocasiona uma latência inaceitável. Agentes móveis oferece uma solução, pois podem ser despachados pelo controlador central para realizarem suas tarefas localmente.

Execução assíncrona e autônoma Tarefas podem ser embutidas em agentes móveis que podem ser despachados pela rede. Após serem despachados, os agentes são autônomos e independentes da criação de processo, podendo executar de forma assíncrona. Este recurso é útil principalmente porque um dispositivo móvel (ex. um laptop) pode se re-conectar à rede para coletar o agente mais tarde.

Adaptação dinâmica Agentes móveis possuem a habilidade de perceber mudanças no ambiente de execução e reagir autonomamente. Múltiplos agentes podem interagir entre si e se distribuir pela rede, de modo a manter uma configuração ótima para resolver um problema em particular.

Naturalmente heterogêneos Redes de computadores, geralmente são heterogêneas, tanto na perspectiva de hardware como na de software. Agentes móveis são independentes da

máquina e também da rede, sendo dependentes somente do seu ambiente de execução, não dificultando a integração de sistemas

Robustez e tolerância a faltas A habilidade dos agentes móveis de reagirem dinamicamente a situações e eventos desfavoráveis torna fácil a construção de sistemas distribuídos robustos e tolerantes a faltas. Se uma máquina está para ser desligada, todos os agentes em execução na máquina podem ser advertidos para que possam ser despachados e continuar suas tarefas em outra máquina da rede.

É verdade que essas vantagens podem ser obtidas usando outros paradigmas, que não agentes móveis. Contudo, como mostrando em [HCK95], se considerarmos a soma das vantagens é que podemos perceber o grande benefício de usar agentes móveis, uma vez que o uso deste paradigma na aplicação traria todas essas vantagens juntas, sem a necessidade de combinação de vários paradigmas.

Desvantagens

Os agentes móveis também podem trazer algumas desvantagens. Primeiramente, existe a necessidade de instalação de um sistema extra (plataforma) em cada máquina que o agente poderá visitar. O código do agente é geralmente interpretado nestas plataformas, tendendo, assim, a ser mais lento. Deve-se evitar mover muito os agentes sem necessidade, pois isso pode aumentar o tráfego da rede, e como colocado em [Pei02], pode-se ter desvantagens usando agentes móveis se não se atentar para dois pontos em relação ao tamanho do agente: o código do agente e os dados carregados. O código do agente deve ser o menor possível, portanto deve-se fatorar suas funcionalidades, deixando apenas aquelas que são indispensáveis para migrar com o agente. Os dados carregados pelo agente também devem ser bem definidos, para que o agente não carregue dados que não serão mais usados ou que podem ser facilmente reconstruídos. Um último problema, que está presente em todas as tecnologias de aplicações distribuídas, é a segurança. A segurança é um dos problemas mais difíceis de se resolver e vem sendo motivo de muita pesquisa como em [Che98] e [TOH00]. Além disso, como mostrado em [TTOH01], quanto mais segurança, mais lento fica o sistema.

2.8 Desenvolvimento de Aplicações Baseadas em Agentes Móveis

Como mostrado, o uso de agentes móveis pode trazer várias vantagens para a aplicação. Além disso, existem várias áreas potenciais para a aplicação de agentes móveis. No entanto, este paradigma ainda sofre uma certa resistência e ainda não é usado em larga escala. Isto pode se dever à dificuldade de desenvolvimento deste tipo de aplicação. Um dos motivos para esta situação é a falta de uma linguagem voltada para as aplicações móveis. Java vem sendo bastante utilizada, porém, ainda sente-se a necessidade da captura de aspectos mais específicos dos agentes móveis, como a própria abstração de agentes ou a mobilidade. Neste sentido, algumas plataformas de suporte para agentes móveis têm provido ambientes com ferramentas e *frameworks* para o desenvolvimento de aplicações baseadas em agentes móveis, como, por exemplo, JADE [BCTR03] e Grasshopper [Gmb01].

No entanto, boas ferramentas e linguagens para desenvolvimento de aplicações de agentes móveis, por si sós, não garantem a produção de software de qualidade, umas vez que as aplicações são desenvolvidas de forma *ad-hoc*, devido a falta de uma metodologia sólida. É importante que aspectos como localização, distribuição e migração dinâmica de componentes sejam considerados explicitamente no projeto do sistema, facilitando a definição da estrutura e comportamento das aplicações baseadas em agentes móveis. No entanto, as metodologias e notações existentes para desenvolvimento de aplicações orientada a objetos não contemplam aspectos importantes de mobilidade. Existem algumas metodologias e notações para desenvolvimento de sistemas multi-agentes, como [MKC01], porém estas se preocupam mais em modelar a interação e cooperação entre agentes, pouco ou não considerando a propriedade de mobilidade. Na notação AUML[Ode01], por exemplo, a mobilidade só é considerada nos diagramas de implantação.

Nesse sentido, alguns trabalhos têm sido desenvolvidos para tentar resolver esse problema da falta de notações e metodologias. O uso de componentes para modelagem de agentes móveis é explorada em [YBF98]. Em [Car99], é apresentada uma notação abstrata para modelagem de agentes móveis. Já em [KRSW01] e [GM01], são propostas extensões para os diagramas UML, para que esses contemplem a mobilidade. Em [TOH99] é proposto um método que trata da fase de projeto do desenvolvimento de aplicações. Nesta proposta,

o projeto é divido em dois níveis arquiteturais, onde são aplicados padrões de projeto.

Em [Gue02a], é apresentada uma adaptação do processo de desenvolvimento de *software* apresentado por Larman[Lar99], adicionando aspectos importantes para representar a mobilidade nas fases de análise e projeto. Na Figura 2.4 são apresentadas as principais atividades da fase de projeto desse modelo. Como pode-se verificar, a fase de projeto é dividida em três momentos: projeto arquitetural independente de plataforma, projeto detalhado independente de plataforma e projeto detalhado dependente de plataforma (onde as especificidades das plataformas devem ser consideradas). A divisão entre projeto independente e dependente de plataforma é motivada por promover o reuso do projeto, uma vez que existe a possibilidade de usar a mesma aplicação em diferentes plataformas. Além disso, o modelo independente, por ser mais simples, pode ajudar no entendimento do comportamento e relacionamento entre as entidades que compuserem o sistema.

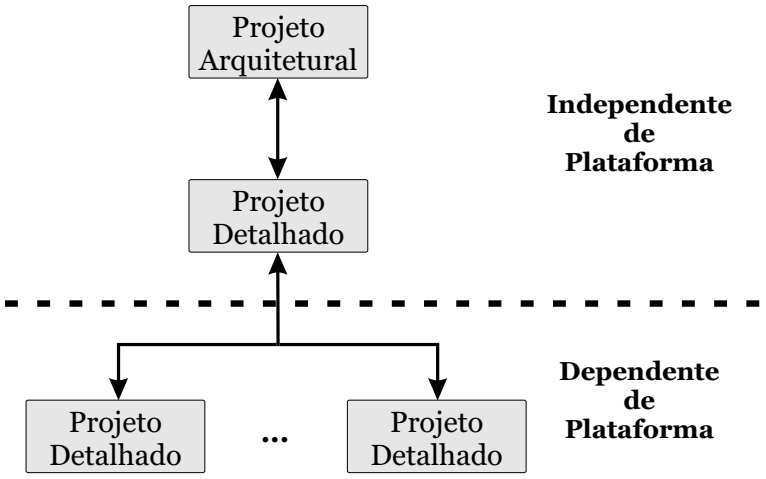


Figura 2.4: Divisão da fase de projeto

2.9 Conclusão

Neste capítulo, apresentamos os agentes móveis, juntamente com os conceitos mais importantes para o entendimento dos mesmos. A crescente utilização e pesquisa destes agentes se dá, principalmente, pelas vantagens e diversas áreas de aplicação exibidas. Além disso, a utilização de agentes como unidade de abstração tem se mostrado bastante interessante para a decomposição e organização de sistemas complexos [WC01].

No entanto, foram destacados problemas que, de certa forma, vêm dificultando o desenvolvimento de aplicações desta natureza. Como mostrado, esforços têm sido empreendidos no intuito de definir notações e modelos que possam facilitar o trabalho dos desenvolvedores. Existem diversas plataformas de suporte para os agentes móveis, contudo, estas ainda não seguem uma padronização, o que impede a utilização de diferentes plataformas em uma mesma aplicação, além de dificultar a reutilização do projeto, caso seja necessário a mudança de plataforma. Assim, é interessante que os modelos considerem a divisão do projeto nas fases independente e dependente de plataforma, a fim de facilitar o reuso do projeto, além de facilitar a compreensão sobre o funcionamento global do modelo. Esta divisão de projeto é similar à proposta pela OMG no desenvolvimento seguindo MDA (*Model-Driven Architecture* [SG01]. Em MDA, o projeto também é dividido em modelo independente de plataforma e modelo específico de plataforma. No entanto, neste caso a plataforma se refere ao *middleware* a ser utilizado pela aplicação, como, por exemplo, CORBA ou *Web-Services*. Além das vantagens apresentadas anteriormente, a OMG destaca outras vantagens que podem ser obtidas com esta divisão do projeto. Vantagens como a facilidade de integração de projetos prontos com outros projetos que venham a ser desenvolvidos, devido à flexibilidade desta abordagem mesmo face a mudanças na infra-estrutura a ser utilizada, o que também diminui os custos de manutenção.

Uma tendência que tem crescido visando facilitar o desenvolvimento de aplicações de forma geral e, conseqüentemente, também de agentes móveis, é o uso de padrões de projeto. Modelos como o de [TOH99] e [Gue02a] têm a fase de projeto baseada na utilização destes padrões. No próximo capítulo entenderemos o que são os padrões de projeto e o que tem sido feito em relação a padrões de projeto e agentes móveis.

Capítulo 3

Padrões de Projeto para Agentes Móveis

Este capítulo tem o objetivo de familiarizar o leitor com os principais conceitos referentes aos padrões de projetos, com um breve histórico do surgimento deste tipo de padrão. O estado da arte dos padrões de projeto para agentes móveis é apresentado, e uma forma de documentação e classificação, que visa facilitar o acesso e o entendimento destes, é proposta. Para exemplificar, um padrão de projeto para agentes móveis é classificado e documentado de acordo com o formato proposto.

3.1 Introdução

A idéia de um padrão é apresentar um modelo de solução para um determinado problema que se repete com alguma constância. Por exemplo, digamos que um indivíduo A deseja enviar uma carta para um indivíduo B. O indivíduo A não precisará ficar tentando inventar uma solução, pois as mesmas já foram pensadas anteriormente. Então ele poderá enviar a carta usando, por exemplo, os correios ou um *office-boy*. Nesse caso, cada forma de enviar seria um padrão que se propõe a resolver o problema.

A noção formal de padrão se originou com os padrões arquitetônicos de Christopher Alexander [AIS77]. Segundo ele, o padrão deve expressar a relação entre um contexto, um problema (inserido no contexto) que ocorre repetidamente, e uma solução para o problema. No geral, um padrão deve ter 4 elementos essenciais [GHJV95]:

1. Nome do padrão, que deve em poucas palavras dizer qual o objetivo do padrão. Esse nome servirá de referência para o padrão e facilitará a identificação do mesmo;

2. Problema, que deve explicar o problema a ser resolvido e o contexto onde o problema está inserido, indicando quando se deve usar o padrão;

3. Solução, que deve descrever um modelo para resolver o problema;

4. Conseqüências, que são os resultados e análises das vantagens e desvantagens do uso do padrão.

No exemplo dos indivíduos poderíamos ter um padrão com nome "Mensageiro", e o problema seria enviar uma carta para outro indivíduo na mesma cidade. A solução seria entregar a carta ao *office-boy*, que entregaria para o indivíduo B. As conseqüências seriam que o envio poderia ser mais rápido, porém o *office-boy* poderia abrir a carta antes de entregá-la.

3.2 Padrões de Projeto

No dia a dia da computação, durante o desenvolvimento de *software*, problemas repetidos também são encontrados em várias fases do desenvolvimento. Assim, cada desenvolvedor busca uma solução para esses problemas, algumas nem tão brilhantes, mas que de qualquer forma demandam um esforço para resolver o problema, quando outros desenvolvedores já teriam uma solução.

Em 1987, alguns desenvolvedores de *software* redescobriram o trabalho de Alexander, passando a aplicar suas idéias de padrões para documentação de decisões de projeto. Os primeiros a utilizar esta idéia foram Ward Cunninghan e Kent Beck, que criaram uma linguagem de padrões de projeto para desenvolvimento de interfaces com o usuário [WRMT98]. Desde então, vários tipos de padrão para a computação vêm sendo propostos, como padrões de análise e de teste.

Um dos padrões mais conhecidos e utilizados são os padrões de projeto (*design patterns*), aplicados durante a fase de projeto do desenvolvimento do *software*. A fase de projeto é realizada após a fase de análise, e enquanto a fase de análise se preocupa em investigar, entender e descrever o domínio do problema, a fase de projeto se preocupa em identificar uma solução lógica para o sistema, ou seja, como o sistema atende aos requisitos [Lar99]. O uso de padrões de projeto facilita o desenvolvimento de aplicações pois as torna mais flexíveis,

mais compreensíveis e reusáveis. Não são necessárias habilidades especiais, características de linguagem ou outros truques para se beneficiar do uso de padrões [AL98], já que o objetivo é reutilizar idéias, e não códigos.

Além dos elemento já citados, os padrões de projeto podem apresentar outros elementos para ajudar a descrição e o entendimento do padrão, como por exemplo padrões relacionados, onde seriam indicados outros padrões que resolvam problemas semelhantes, possivelmente apontando diferenças entre os padrões.

Apesar dos padrões já serem por si só importantes, ainda podemos, como colocado em [DW99], aumentar o poder dos padrões juntando padrões relacionados de modo a formar uma linguagem de padrões. Enquanto padrões se ocupam em resolver problemas isolados, a linguagem de padrões poderia resolver uma parte da aplicação, como, por exemplo, uma camada da arquitetura.

3.3 Padrões de Projeto para Agentes Móveis

Todas as vantagens do uso de padrões citadas na seção anterior podem ser trazidas para as aplicações de agentes móveis, facilitando, desta forma, o desenvolvimento do projeto das mesmas. Neste sentido, padrões de projeto para agentes móveis têm sido identificados, classificados e documentados em vários trabalhos [AL98; TOH99; KKPS98; DOKO99; SSJ; AD98]. No entanto, estes padrões ainda apresentam alguns problemas.

Um primeiro problema é a falta de relacionamento e repetição dos padrões, uma vez que cada autor propõe seus padrões de forma isolada, não os relacionando com outros e, algumas vezes, propondo padrões semelhantes a outros já propostos. Um maior esforço para tentar relacionar estes padrões poderia contribuir para a definição de linguagens de padrão. Como destacado em [DOKO99], uma classificação unificada para os padrões poderia facilitar a definição de relação entre eles, porém ainda não se tem essa classificação definida e cada autor classifica seus padrões como achar mais interessante.

Um cuidado que se deve ter é tentar não prender padrões a especificidades de plataformas, tentando defini-los para que eles possam ser aplicados em qualquer plataforma. Para isso, são importantes os estudos de caso, mostrando a viabilidade de implementação dos padrões nas mais variadas plataformas. Entretanto, os poucos que são apresentados são simples e

apresentam soluções presas a plataformas específicas, dificultando a aplicação dos padrões em outras plataformas.

Um último problema é a forma de descrição dos padrões, já que alguns são descritos com pequenos textos, outros com figuras e outros com diagramas UML. Estas descrições nem sempre facilitam o entendimento do padrão, levando até a entendimentos equivocados. Seria interessante definir uma forma de documentação para os padrões. Além disto, poderia-se, também, descrever os padrões usando alguma linguagem formal, gerando modelos sem ambigüidade e possibilitando realizar análises sobre estes.

3.4 Catálogo de Padrões de Projeto para Agentes Móveis

Nesta seção apresentaremos o catálogo de padrões de projeto para agentes móveis proposto em nosso trabalho. A existência de um catálogo detalhado e unificado pode trazer várias vantagens para a comunidade de agentes móveis, uma vez que o acesso aos padrões seria facilitado.

A aplicação de modelos que têm a fase de projeto baseada na utilização de padrões para agentes móveis, como os propostos em [Gue02a] e [TOH99], se torna praticamente inviável sem essa catalogação. Então, o catálogo auxiliaria na identificação e entendimento dos padrões, bem como na escolha de quais padrões aplicar.

A seguir, apresentaremos a classificação definida para os padrões, e uma forma de documentação para estes é proposta. É interessante que novos padrões que venham a ser propostos estejam no formato do catálogo apresentado, evitando a ocorrências dos problemas detectados na seção anterior.

3.4.1 Categorias

A definição de categorias para classificação dos padrões de projeto para agentes móveis visa facilitar a identificação de que padrão aplicar de acordo com o tipo do problema para o qual se busca solução. Desta forma, não é necessário que o desenvolvedor procure dentre todos os padrões, aquele que pode resolver seu problema, buscando, então, apenas na categoria na qual o mesmo está inserido. Assim, por exemplo, se o problema se refere à interação de agentes, pode-se recorrer aos padrões de comunicação.

Estas categorias foram definidas de acordo com os padrões de projeto que tem sido propostos. Desta forma, a partir do levantamento destes padrões, buscou-se englobar as questões mais comuns referentes aos agentes móveis, como migração e realização de tarefa. Como pode ser visto no catálogo (Apêndice A), todos os padrões puderam ser classificados de acordo com as categorias definidas.

A seguir, temos as categorias para estes padrões de projeto:

- **Arquitetura**: padrões que se preocupam com arquiteturas de aplicações baseadas em agentes móveis. São padrões mais complexos, que definem toda a estrutura de uma aplicação, podendo serem compostos por outros padrões mais simples;
- **Migração**: padrões que se preocupam com o controle da migração do agente para uma ou mais plataformas. Estes padrões definem como os agentes devem migrar e que ordem devem seguir durante a migração entre diversas plataformas;
- **Tarefa**: padrões que se preocupam com a realização de tarefas pelos agentes móveis. Nestes padrões são definidas questões como a delegação e organização da distribuição das tarefas;
- **Segurança**: padrões que se preocupam com a segurança tanto do agente, quanto da plataforma de suporte ao funcionamento destes. São padrões muito específicos das plataformas, e seu uso depende do suporte provido pela mesma;
- **Recursos**: padrões que se preocupam com o controle do compartilhamento e acesso de recursos;
- **Comunicação**: padrões que se preocupam com a comunicação entre dois ou mais agentes. Estes padrões devem definir formas de controle da interação entre os agentes, indicando como a mesma se realizará (por compartilhamento de espaços ou troca de mensagens, por exemplo).

A Tabela 3.1 mostra alguns dos padrões propostos por vários autores, classificados de acordo com as categorias apresentadas anteriormente.

Categoria	Arquitetura	Migração	Tarefa	Segurança	Recurso	Comunicação
Padrão	*Agent Pattern;*	*Itinerary*; *Branching*; *Star-Shaped*; *Forwarding*; *Ticket*	*Master-Slave*; *Plan*	*Authentication*; *Mutual Itinerary Recording*; *Cryptography*;	*MoProxy*; *Broker*	*Meeting*; *Messenger*; *Facilitator*; *Group Communication*

Tabela 3.1: Alguns padrões classificados de acordo com as categorias propostas.

3.4.2 Forma de Documentação

Além da classificação dos padrões de projeto para agentes móveis, importante para identificação de que padrões existem para um dado problema, é importante que o padrão apresente uma boa documentação. É nesta documentação que estarão detalhes sobre o objetivo, estrutura e funcionamento do padrão, que facilitarão o entendimento do mesmo pelo desenvolvedor.

A forma de documentação proposta neste trabalho é baseada em [GHJV95], sendo composta pelos elementos explicados a seguir. É importante que padrões propostos apresentem documentação com os vários elementos preenchidos.

- **Descrição**: deve apresentar uma descrição sucinta sobre o funcionamento e objetivo do padrão. Maiores detalhes sobre o padrão devem ser evitados, pois serão expostos mais à frente;

- **Motivação**: deve motivar o desenvolvedor para o uso do padrão, indicando porque o mesmo deve ser aplicado e que vantagens o padrão pode trazer para a aplicação;

- **Aplicabilidade**: deve indicar possíveis tipos de aplicações onde o padrão pode ser utilizado. É importante mostrar como o padrão pode ajudar no desenvolvimento do tipo de aplicação;

- **Estrutura**: a estrutura do padrão refere-se aos modelos independentes de plataforma, ou seja, que não dependem das especificidades das plataformas. Esta deve indicar os elementos que fazem parte do padrão, bem como seus relacionamentos e funcionamento. Sugerimos os diagramas UML de classes e de seqüências para representação da estrutura do padrão, uma vez que podem, de uma forma simples e abstrata, capturar as necessidades da mesma. Estes modelos são mais simples que os dependentes de

plataforma, facilitando o entendimento sobre o funcionamento do padrão e podendo ser detalhado para qualquer plataforma de agentes móveis;

- **Implementação**: refere-se aos modelos dependentes de plataforma. É o detalhamento dos modelos da estrutura para uma ou mais plataformas específicas, apresentando como o padrão deve ser projetado para determinada plataforma. Estes modelos podem facilitar a aplicação do padrão em uma determinada plataforma. Além disto, a idéia da modelagem de acordo com especificidades da plataforma pode ser utilizada para modelagem em outras plataformas que apresentem características semelhantes;

- **Conseqüências**: deve indicar as conseqüências boas e/ou ruins que podem ser trazidas pela aplicação do padrão;

- **Usos conhecidos**: deve apresentar casos reais de uso do padrão, onde o desenvolvedor poderá verificar o funcionamento do padrão dentro do contexto de uma aplicação real;

- **Padrões relacionados**: deve apresentar padrões relacionados ao padrão documentando, que possam ser aplicados em substituição ao mesmo.

A seguir, temos a documentação do padrão de tarefa *Master-Slave*. A classificação e documentação dos demais padrões são apresentadas no Apêndice A.

3.4.3 Exemplo: *Master-Slave*

O padrão *Master-Slave* tem sido um dos mais usados padrões de projeto para agentes móveis, tendo se mostrado bastante eficiente nas aplicações onde tem sido utilizado.

A Figura 3.1 ilustra o funcionamento do padrão, onde temos um agente mestre criando um escravo (1), que migra para execução remota da tarefa (2). Após a execução da tarefa (3), o agente escravo retorna para a origem (4), onde entrega os resultados para o mestre (5).

Este padrão, que documentamos a seguir, foi proposto em [AL98], e é classificado na categoria de **Tarefa**.

- **Descrição**: Um agente mestre delega e controla tarefas a serem executadas remotamente por agentes escravos.

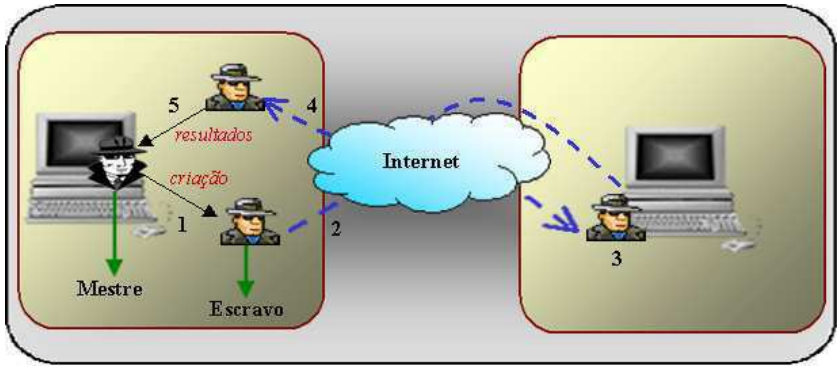


Figura 3.1: Padrão *Master-Slave*

- **Motivação**: Utilizando este padrão, será possível que a máquina onde está o agente mestre se desconecte durante a realização da tarefa, só reconectando para recuperação do agente escravo. Outra vantagem, é que o agente mestre exercerá um controle central sobre a execução da tarefa, sabendo que escravo está executando que tarefa e aonde. Outro fator interessante deste padrão é que o agente mestre poderá ser responsável pela comunicação com aplicações *desktop*, podendo até funcionar como interface com o usuário, uma vez que quem migra para realização da tarefa é o agente escravo, ficando o mestre sempre na máquina de origem.

- **Aplicabilidade**: Três tipos de aplicação se mostram bastante interessantes para o uso deste padrão:

  - aplicações com interfaces com usuário, podendo, como dito, o agente mestre fazer o papel da interface;
  - aplicações que necessitem de um controle central, que será exercido pelo agente mestre;
  - aplicações de computadores portáteis com conexão de alto-custo e/ou intermitente, uma vez que o computador poderá ser desconectado durante a execução da tarefa, que é realizada de forma autônoma pelo agente escravo.

- **Conseqüências**: O padrão provê uma forma de delegação de tarefa, porém, para várias tarefas é necessário que exista uma gerência das tarefas que serão executadas paralelamente, bem como o controle de muitos agentes escravos. Isto pode não ser uma tarefa

trivial, podendo até ser necessária a combinação com um outro padrão de tarefa, como o *Plan*.

- **Usos conhecidos**: Sistema de apoio às atividades de comitês de programa em conferências [Gue02a].

- **Padrões relacionados**: *Plan*

Estrutura

Em nosso trabalho, é utilizada uma notação equivalente à apresentada em [Gue02a][1]. Nesta notação, o diagrama de seqüência possui um objeto representando a entidade da plataforma que controla a execução do agente (criação, destruição, migração) e indica sua localização atual. As migrações são representadas com setas, de uma entidade agência para outra, rotuladas com um *MIGRATING AGENT*. Ela é requisitada pelo envio de uma mensagem *move()*. Antes da migração, a execução do agente é interrompida (seta rotulada com *destroy()*). A execução é continuada no destino após a migração ser efetivada (seta rotulada com *initialize()*).

No diagrama de seqüências da Figura 3.2 temos a representação do funcionamento do padrão. Inicialmente, o agente mestre (*MasterAgent*) solicita à agência (*SourceAgency*) que crie um agente escravo (*SlaveAgent*) para a realização de uma dada tarefa. Após sua criação, o escravo executa as inicializações necessárias à tarefa, chamando o método *initializeJob*, e então migra para a agência (*DestinyAgency*), indicada pelo agente mestre. Ao chegar no destino, o escravo invoca o método *doJob*, que implementa a tarefa a ser executada. Finalizando seu trabalho, ele retorna para a agência de origem, onde entrega os resultados para o agente mestre, invocando o método *addResult*. Por fim, o escravo se destrói através do método *dispose*, que indica para a agência que o agente deve ser destruído.

No diagrama de classes da Figura 3.3 temos a estrutura do padrão, onde temos os dois agentes, mestre e escravo, com seus respectivos métodos. Podemos verificar a restrição de que o agente escravo deve ser um agente móvel, uma vez que ele executará tarefas remotamente.

[1] Alguns elementos gráficos diferem devido a limitações da ferramenta de edição.

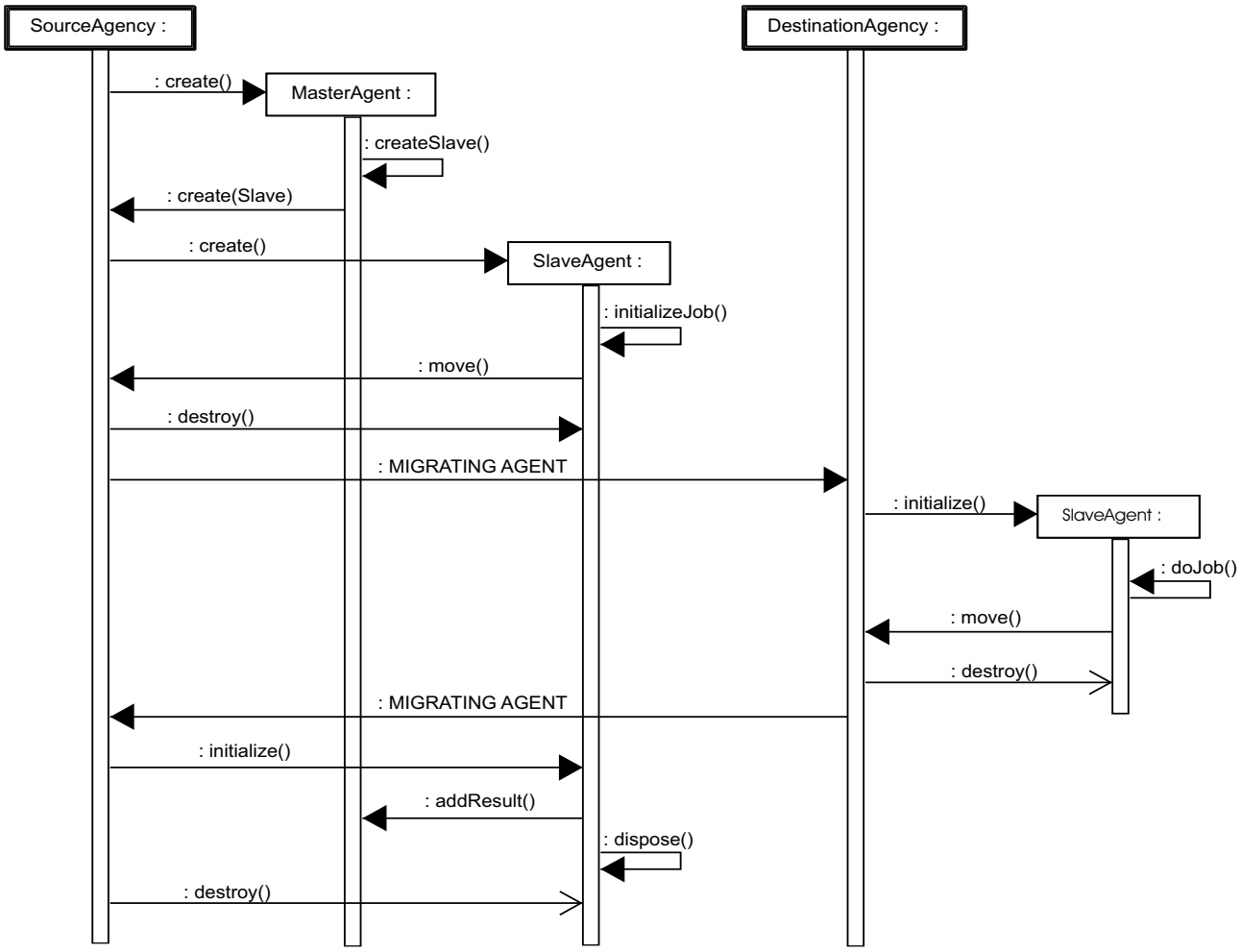


Figura 3.2: Diagrama de seqüências do *Master-Slave*

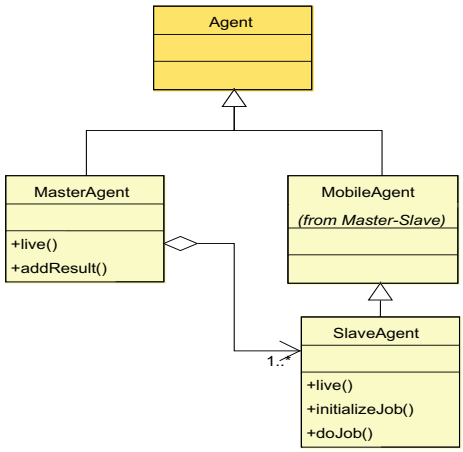


Figura 3.3: Diagrama de classes do *Master-Slave*

Implementação

Aglets

A plataforma Aglets facilita a utilização do padrão *Master-Slave* uma vez que parte do trabalho já está contido no pacote *com.ibm.agletx.patterns*. No diagrama de classes da Figura 3.4 podemos verificar a existência da classe *Slave*, que encontra-se parcialmente implementada. É necessário apenas que o desenvolvedor especifique a tarefa a ser executada nos métodos abstratos *initializeJob* (chamado *initializeTask* na classe Slave), que inicializa o escravo para satisfazer as pré-condições da tarefa, e *doJob* (chamado *doTask* na classe Slave), que executa a tarefa. No método *run()*, que define o ciclo-de-vida do agente, estão definidos os passos básicos da execução de um agente escravo, inclusive com as devidas chamadas aos métodos *initializeTask* e *doTask*.

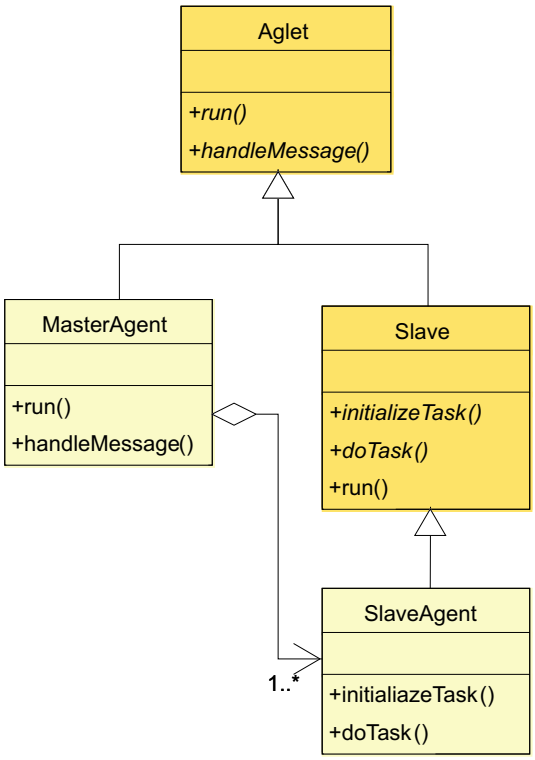


Figura 3.4: Diagrama de classes do *Master-Slave* na plataforma Aglets

Como podemos observar no diagrama de seqüência (Figura 3.5), quando o agente Master é inicializado, ele cria os agentes escravos que deverão realizar tarefas remotas enquanto ele continua sua execução localmente. Os escravos, por sua vez, executam o método *initializeTask* e migram para o local remoto. Ao chegar no local onde a tarefa será executada,

eles invocam o método *doTask*, que implementa o algoritmo do trabalho a ser realizado, migram de volta e enviam uma mensagem com o resultado da tarefa ao agente mestre e, por fim, se destroem.

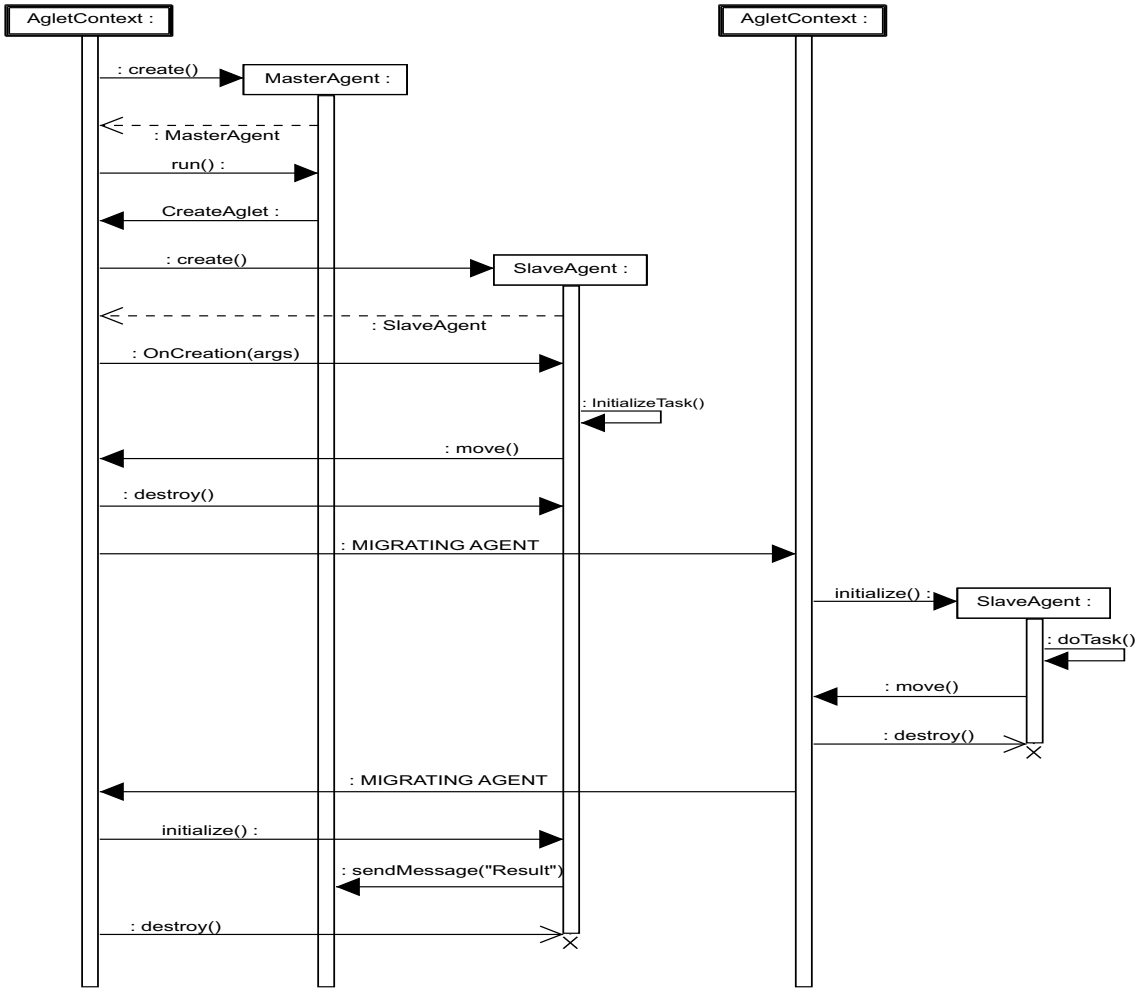


Figura 3.5: Diagrama de seqüências do *Master-Slave* na plataforma Aglets

Grasshopper

No Grasshopper (Figuras 3.6 e 3.7) ocorre tudo também de forma bastante semelhante ao sugerido pelo modelo independente de plataforma. Entretanto, neste caso, o agente mestre é definido como uma extensão de um agente estacionário, classe *StationaryAgent*, por questões de conveniência (conceitualmente o agente mestre não deve se mover). Já o agente escravo é, obviamente, um agente móvel, classe *MobileAgent*. O método *live* determina o ciclo-de-vida do agente.

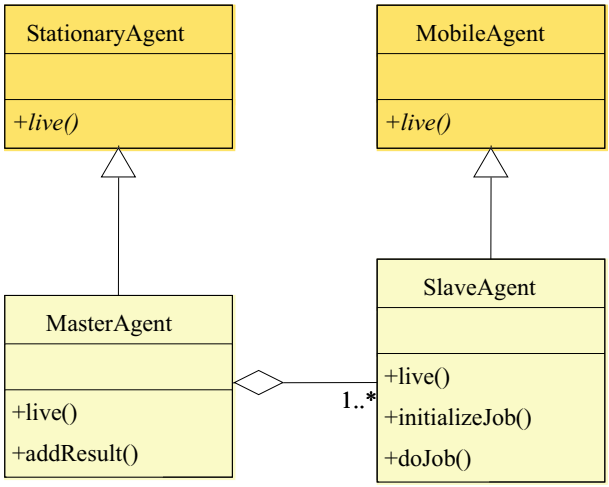


Figura 3.6: Diagrama de classes do *Master-Slave* na plataforma Grasshopper

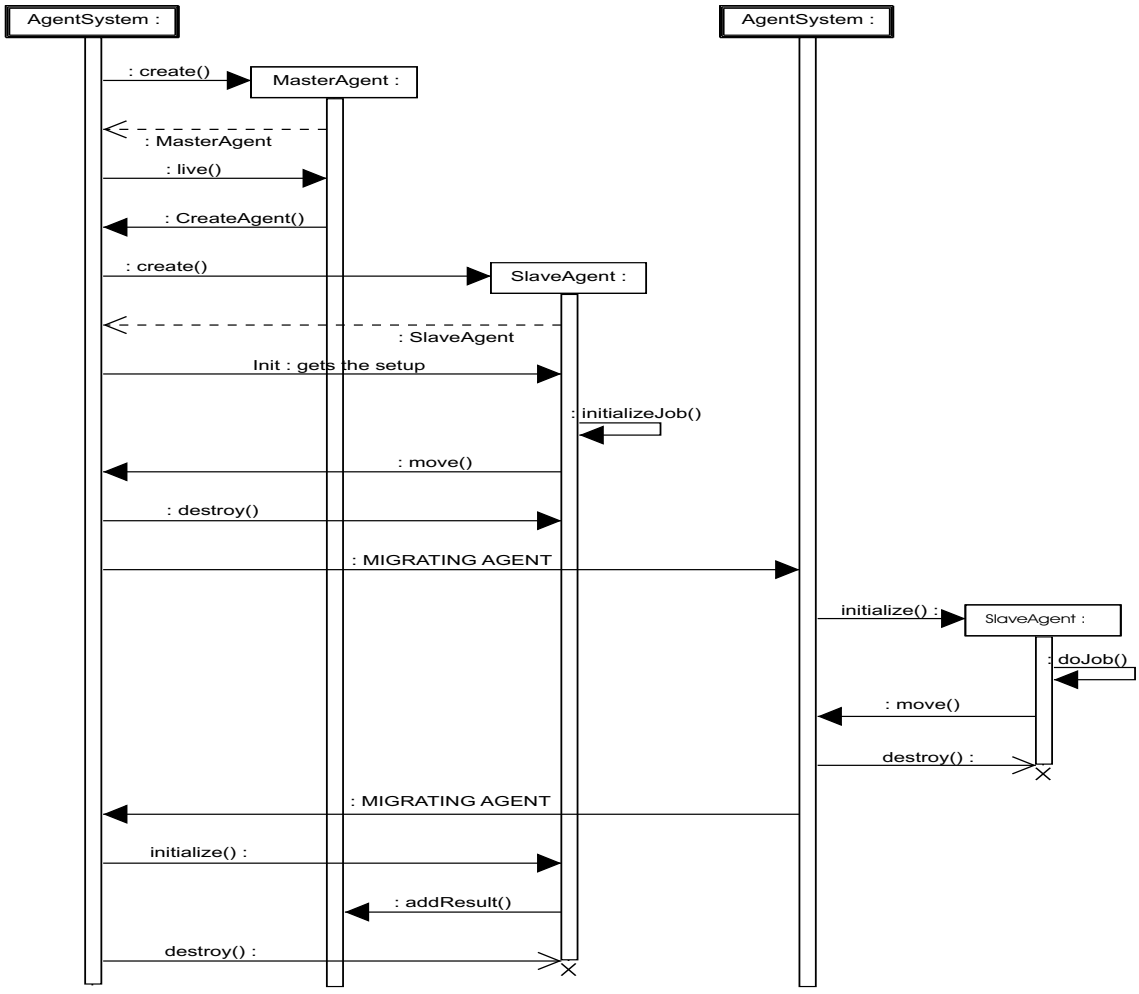


Figura 3.7: Diagrama de seqüências do *Master-Slave* na plataforma Grasshopper

JADE

Em JADE, verificamos no diagrama de classes da Figura 3.8 que, além das classes presentes no modelo independente, existem classes de comportamento. A classe *ReceiveResultBehaviour* é um comportamento cíclico responsável por receber as mensagens destinadas ao agente mestre. Já a classe *JobBehaviour* é um comportamento do agente escravo onde está implementada a tarefa que ele irá executar. Este encapsulamento da tarefa, torna simples o uso de um mesmo escravo para várias tarefas, sendo necessária somente a mudança do comportamento, não alterando o código do agente.

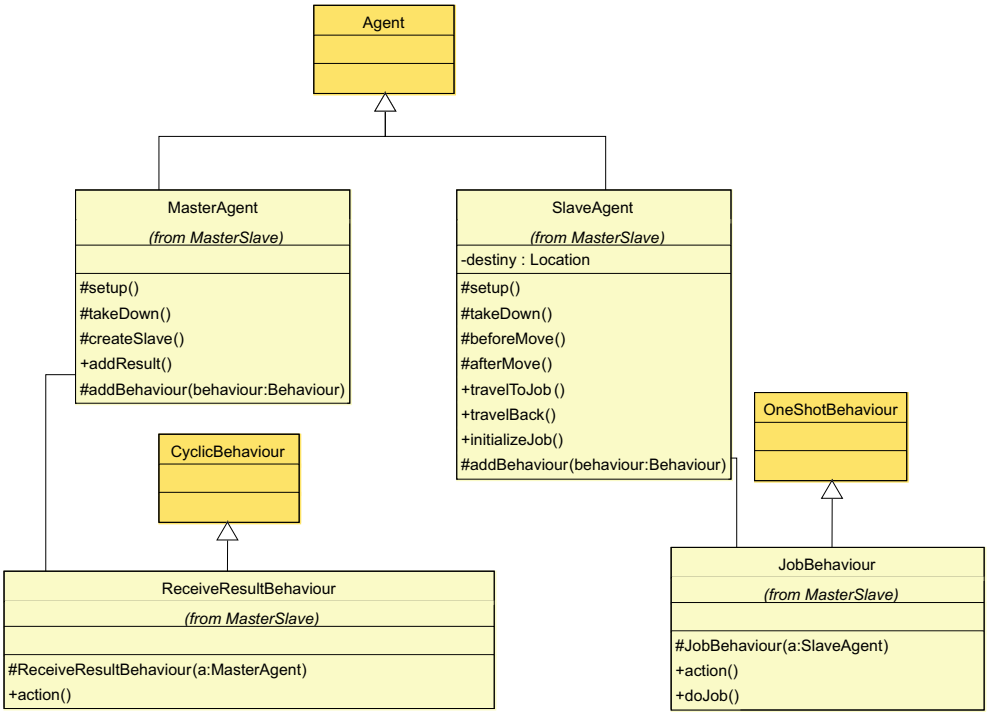


Figura 3.8: Diagrama de classes do *Master-Slave* na plataforma JADE

Em relação à dinâmica do padrão percebemos, no diagrama de seqüências (Figura 3.9), que a diferença está na presença dos comportamentos. O comportamento *ReceiveResultBehaviour* é adicionado ao agente mestre e fica esperando uma mensagem do agente escravo. Quando recebe a mensagem com os resultados, ele chama o método *addResult*, para encaminhada-los para o mestre. Já o comportamento *JobBehaviour*, é adicionado ao agente escravo quando esse chega ao destino. Como explicado, esse comportamento executa a tarefa destinada ao agente.

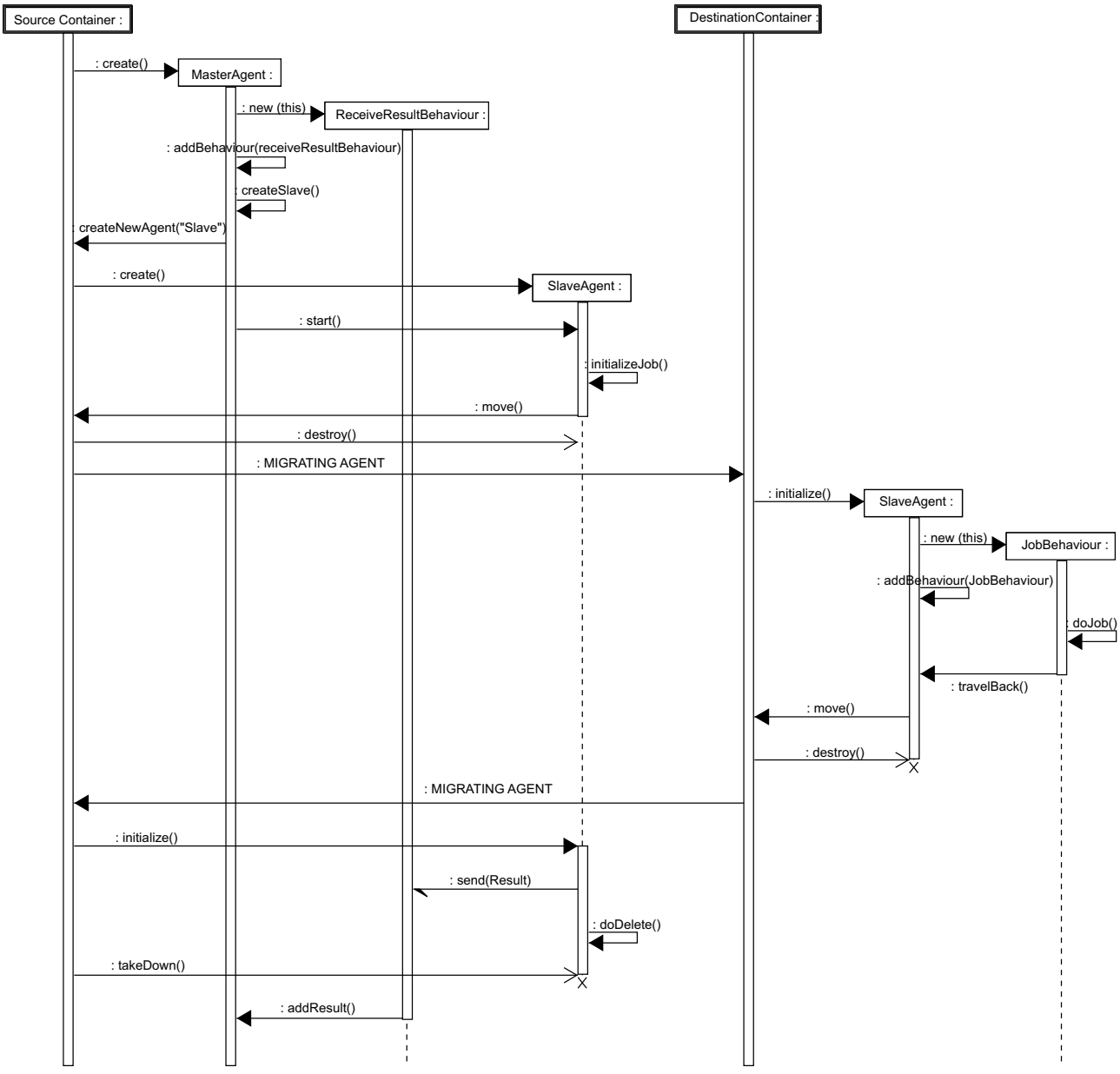


Figura 3.9: Diagrama de seqüências do *Master-Slave* na plataforma JADE

3.5 Conclusão

Neste capítulo, foram apresentados os conceitos de padrões de projeto, juntamente com os conceitos de padrões de uma forma geral. Vimos as vantagens da aplicação destes, bem como os esforços que se tem realizados em direção à definição de padrões de projetos para agentes móveis.

Como exposto, os padrões para agentes móveis ainda apresentam alguns problemas, principalmente no que diz respeito à forma de documentação. Sendo, desta forma, necessária uma boa descrição do padrão, evitando dificuldades na compreensão do mesmo, o que pode levar a um entendimento equivocado sobre o funcionamento do padrão.

Neste trabalho, foi proposta uma forma de documentação, incluindo elementos como descrição, aplicabilidade e estrutura do padrão. No entanto, uma outra forma de documentação, que pode servir como adicional à documentação proposta, é a modelagem do padrão com algum método formal. Esta formalização apresenta a vantagem de ser uma descrição sem ambigüidade, além de permitir análises sobre a estrutura e funcionamento do mesmo.

Na próximo capítulo apresentaremos alguns dos trabalhos que têm tratado da formalização de padrões de projeto de uma forma geral, bem como da formalização de agentes móveis. A formalização de padrões de projeto para agentes móveis também será apresentada.

Capítulo 4

Formalização de Padrões de Projeto para Agentes Móveis

Neste capítulo será apresentada a formalização de padrões de projeto para agentes móveis. Para tanto, inicialmente são apresentados trabalhos que tratam da formalização de agentes móveis e de padrões. Em seguida, as redes de Petri, formalismo escolhido para modelagem dos padrões, são introduzidas, juntamente com seus principais conceitos. Por fim, a formalização dos padrões de projeto para agentes móveis é apresentada, juntamente com as técnicas de validação aplicadas.

4.1 Formalização de Agentes Móveis e Padrões de Projeto

A formalização de agentes móveis e padrões tem sido abordada em muitos trabalhos, tendo objetivos distintos e utilizando vários métodos formais. Nesta seção, apresentaremos alguns destes trabalhos, destacando que formalismo foi aplicado, e com que fim a modelagem foi realizada. Em seguida, apresentaremos o formalismo escolhido para formalização dos padrões de projeto para agentes móveis, destacando os critérios considerados para esta escolha.

4.1.1 Trabalhos Relacionados

Formalização de Agentes Móveis

Os trabalhos que tratam da formalização de agentes móveis podem ser divididos em 3 categorias: trabalhos que propõem formalismos para modelagem, trabalhos que utilizam formalismos para análise, e trabalhos que apresentam estudos comparativos entre vários formalismos.

Na primeira categoria, três trabalhos podem ser destacados. Em [VBML99], um método para especificação formal baseada em máquinas de estado abstratas é proposto. Este método mapeia as abstrações de computação móvel apresentadas em *Ambient Calculus*. Já em [XD00] são mostrados 3 modelos de redes de Petri Predicado-Transição para sistemas de agentes móveis. No primeiro é modelado um agente, no segundo é modelada a mobilidade do agente e no terceiro é modelado o sistema de agentes conforme proposto pela MASIF[Fok98]. A modelagem de agentes móveis com *Actors* é proposta em [AMM01], onde são destacadas as vantagens de utilização deste formalismo. Esta modelagem é realizada com a finalidade de facilitar o desenvolvimento de aplicações utilizando uma API Java que também é proposta.

Dentre os trabalhos que tratam da utilização dos modelos formais podemos destacar [RB99], [RW00] e [MCM01]. Em [RB99] e [RW00], os padrões de projeto para agentes móveis *Meeting* e *Master-Slave* são modelados com redes de Petri estocásticas, tendo como principal finalidade a análise de propriedades e de desempenho. Também com fins de análise de desempenho, em [MCM01], dois sistemas de recuperação de informação são modelados com redes coloridas estocásticas bem formadas (SWNs). Um destes sistemas utiliza agentes móveis. Os resultados desta análise são comparados para determinar que modelo tem melhor desempenho.

Um estudo comparativo de formalismos para modelagem de agentes móveis é apresentado em [SMT98]. Nele, os autores modelam os ambientes de código móvel *Obliq* e *Messenger*. Assim, é verificado como cada formalismo modela aspectos como migração, comunicação, e localização de agentes. Os formalismos usados são: variações de π*-calculus*, *Ambient-Calculus*, variações de redes de Petri, *Actors* e linguagens de coordenação.

Formalização de Padrões

Os métodos formais também têm sido bastante utilizados para modelagem de vários tipos de padrão, como, por exemplo, padrões de projeto, padrões de interação com o usuário e padrões de *workflow*. Em [Mik98], os padrões de projeto *Observer* e *Mediator* são modelados com o método de especificação DisCo, com o objetivo de representar aspectos temporais dos padrões, sendo também mostrada a possibilidade da combinação destas formalizações. A representação gráfica da seção de intenção dos padrões de projeto de [GHJV95] é mostrada em [GE99]. Esta representação gráfica utilizada tenta interpretar palavras e sentenças da intenção dos padrões como símbolos. Já em [TF02], os autores formalizam padrões de interação com o usuário, usando as redes de Petri lugar/transição, e tendo como objetivo principal a combinação dessas formalizações. Finalmente em [vdAtH], os autores formalizam padrões de *workflow*, usando, inicialmente, redes de Petri coloridas, hierárquicas e temporizadas. Porém, algumas limitações na formalização de alguns padrões são detectadas. Desta forma, é proposta uma extensão de redes de Petri, chamada de YAWL (*Yet Another Workflow Language*), que adiciona elementos gráficos para facilitar a modelagem dos padrões de workflow.

Conclusões

Nesta seção, foram mencionados alguns formalismos para modelagem de agentes móveis, e todos estes formalismos tiveram sucesso em sua modelagem. No entanto, é importante destacar que alguns modelam os agentes móveis mais naturalmente que outros, e que a facilidade de modelagem depende muito do nível de abstração que se deseja capturar da aplicação e do objetivo com a modelagem, como análise ou especificação formal.

A partir destes trabalhos, pode-se perceber e importância da formalização tanto dos padrões, como forma de descrição e especificação, quanto dos agentes móveis, principalmente para fins de análise. No entanto, sente-se a falta de trabalhos que unam a formalização de agentes móveis e de padrões, permitindo a utilização dos modelos para vários fins. As modelagens apresentadas em [RB99; RW00], apesar de serem utilizadas para análises, são complicadas para o entendimento, sendo difícil identificar os elementos dos padrões, o que prejudica a utilização como forma de documentação.

Em nosso trabalho, estas duas abordagens são unidas com o intuito de termos uma formalização dos padrões de projeto para agentes móveis. Nesta formalização, buscamos construir modelos abstratos destes padrões, de forma que os mesmos pudessem ser utilizados tanto como forma de documentação, como para análises (estrutural e de desempenho). Além disto, os modelos formais construídos poderão ser combinados de forma a construir especificações maiores. Os trabalhos citados anteriormente, são importantes não só por destacar a necessidade da formalização, mas também por dá idéias de fatores que podem ser analisados ([MCM01; RB99; RW00]), e de estratégias para combinar as formalizações dos padrões [TF02].

4.1.2 Escolha do Formalismo

Para formalização dos padrões de projeto para agentes móveis, foram utilizadas as redes de Petri Coloridas Hierárquicas Temporizadas (THCPN). Este formalismo foi escolhido pois possui aspectos que consideramos importantes para um melhor aproveitamento dos modelos gerados.

Um primeiro aspecto diz respeito à facilidade no entendimento do modelo, evitando que se restrinja a usuários experientes do formalismo. Este entendimento é facilitado com THCPN, uma vez que este possui uma representação gráfica e permite a divisão da formalização em módulos. A ferramenta Design/CPN[1] é fundamental no desenvolvimento dos modelos, uma vez que, além de validá-los, permite também a simulação, o que pode ajudar no entendimento da dinâmica destes. Outro fator importante é o amadurecimento das técnicas de análise estáticas, dinâmicas e de desempenho deste método formal. As técnicas de análise são importantes para verificar propriedades dos padrões e detectar possíveis problemas já durante a fase de projeto.

Na seção seguinte, este formalismo é introduzindo, apresentando suas características e principais conceitos. Um aprofundamento maior sobre este formalismo pode ser visto em [Mur89] e [Jen92].

[1] http://daimi.aau.dk/designCPN

4.2 Redes de Petri

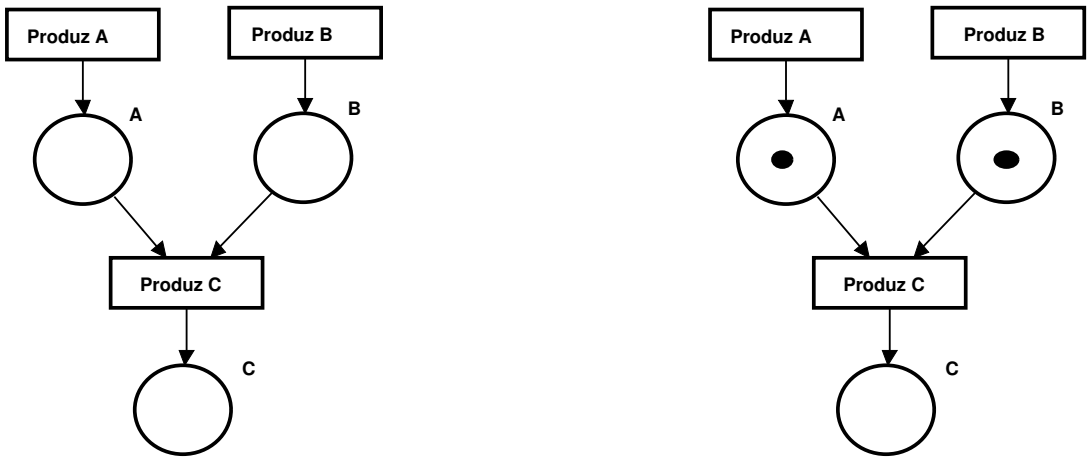
As redes de Petri são uma ferramenta matemática para a modelagem e análise de sistemas, especialmente os que possuem características concorrentes, distribuídas, paralelas, assíncronas e/ou estocásticas[Mur89], que surgiu no começo dos anos 60 e desde então vem se desenvolvendo muito, tanto em relação a teorias quanto aplicações. As redes de Petri possuem uma notação gráfica bastante amigável e seus modelos são ditos executáveis, por permitirem simulações.

Uma rede de Petri é composta de uma estrutura de rede, inscrições associadas a essa estrutura e uma marcação. A estrutura de rede é um grafo bipartido direcionado, com dois tipos de nós: lugares, representados graficamente por círculos, e transições, representadas graficamente por retângulos. A marcação de um lugar é definida pela presença de zero ou mais fichas neste lugar. Já a marcação de uma rede de Petri é definida pelo conjunto da marcação de todos os lugares desta rede. O estado do sistema é definido pela marcação da rede, enquanto as ações são representadas pelas transições. A marcação inicial de uma rede representa o estado inicial do sistema.

Na Figura 4.1(a), temos um exemplo de uma rede de Petri lugar/transição. Nela, existem 3 lugares (A, B e C) e 3 transições (Produz A, Produz B e Produz C), ligados por arcos. Na marcação inicial deste modelo não existem fichas em seus 3 lugares. Desta forma, pode-se afirmar que, no estado inicial do sistema modelado pela rede de Petri, não existem produtos do tipo A, B ou C disponíveis.

As transições de uma rede possuem um certo número de lugares de entrada, que são os lugares de origem de onde partem os arcos que atingem a transição, e de lugares de saída, que são os lugares para onde partem os arcos da transição. Para simular o comportamento dinâmico de um sistema, uma marcação em uma rede de Petri é alterada de acordo com uma regra de disparo. A regra de disparo, portanto, determina quais são as condições para uma transição estar habilitada a disparar e quais as conseqüências do seu disparo. Como apresentado em [Mur89], a regra de disparo de uma transição pode ser dividida em três partes:

1. Uma transição está habilitada a disparar, se cada lugar de entrada contém pelo menos o número de fichas indicado pelo peso do arco do respectivo lugar de entrada.



(a) Marcação inicial

(b) Após os disparos

Figura 4.1: Exemplo de uma rede de Petri lugar/transição: marcação inicial da rede (a) e marcação após o disparo das transições *Produz A* e *Produz B* (b).

2. Uma transição habilitada pode ou não disparar.

3. O disparo de uma transição habilitada remove o número de fichas indicado pelo peso do arco de cada lugar de entrada, e adiciona no lugar de saída o número de fichas indicado pelo peso do arco do referido lugar.

Voltando ao modelo da Figura 4.1(a), a transição *Produz C* possui dois lugares de entrada (A e B) e um lugar de saída (C). Já as transições *Produz A* e *Produz B* não possuem lugares de entrada, mas possuem um lugar de saída (A e B, respectivamente). De acordo com a regra de disparo, pode-se verificar que as transições *Produz A* e *Produz B* estão habilitadas para o disparo (parte 1 da regra de disparo). Já a transição *Produz C* não está habilitada, uma vez que não existem fichas em seus lugares de entrada. Após o disparo das transições *Produz A* e *Produz B*, temos a rede com uma nova marcação (Figura 4.1(b)). Nesta marcação, a transição *Produz C* estará habilitada para o disparo, uma vez que existem fichas nos lugares A e B. As transições *Produz A* e *Produz B* continuam habilitadas para o disparo.

As redes de Petri de alto-nível caracterizam-se por permitirem associar tipos de dados às fichas. As fichas podem carregar informações complexas, que são manipuladas conforme as inscrições dos arcos. O fato das fichas expressarem informações complexas aumenta o poder de descrição dessas redes, permitindo a construção de modelos mais compactos. A passagem

de redes de Petri de baixo nível para redes de alto nível pode ser comparada à passagem da linguagem de máquina para linguagens de programação de alto nível [Jen92].

As redes de Petri de alto-nível mais utilizada são as redes de Petri coloridas (CPN). Nestas redes, cada ficha tem associada a ela um valor de dado chamado cor da ficha. Associado a cada lugar há um tipo de dados chamado de conjunto de cores (*colour set*), que determina os possíveis valores das fichas ali contidas. A declaração dos conjuntos de cores associados aos lugares, variáveis e funções utilizadas nas redes dos modelos são feitas no nó de declaração. Em CPN, a linguagem de programação utilizada nas inscrições de arco e nas declarações é denominada CPN-ML, que é uma extensão da linguagem de programação SML. Em relação à regra de disparo, pode existir mais uma condição para que a transição esteja habilitada a disparar. Esta condição é definida pela guarda, que é uma expressão booleana associada a transição. Desta forma, a transição só dispara se a guarda for avaliada como verdadeira.

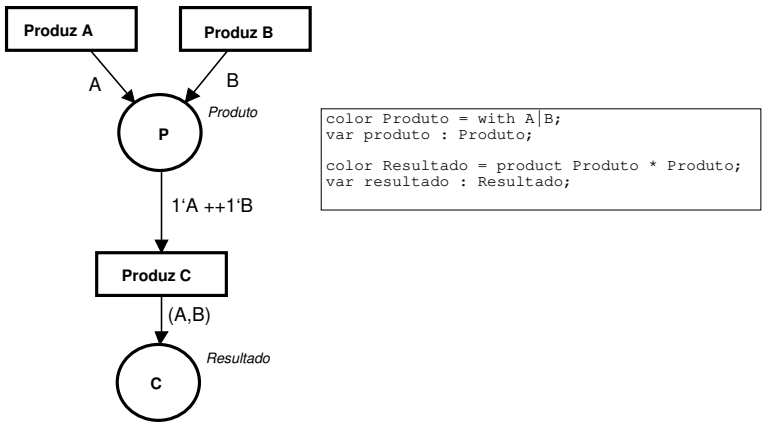
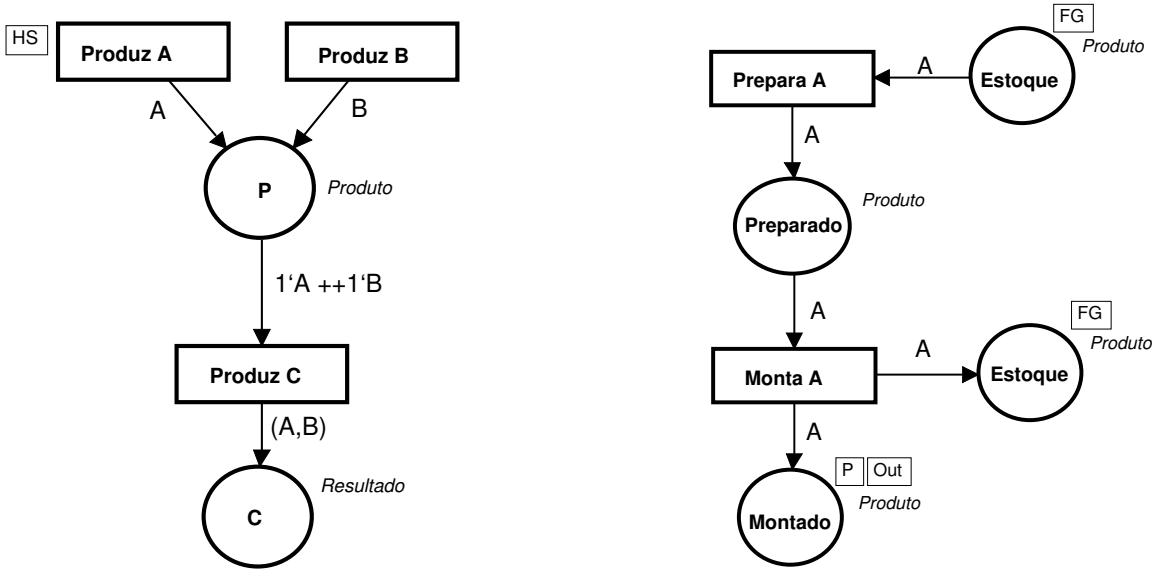


Figura 4.2: Exemplo de rede de Petri colorida

Na Figura 4.2, é apresentado um modelo de rede de Petri colorida. O sistema modelado é o mesmo apresentado para as redes lugar/transição. Os lugares P e C possuem uma cor associada (Produto e Resultado, respectivamente) indicando os possíveis valores das fichas que estarão no lugar. Cada cor é definida no nó de declaração (retângulo ao lado do modelo). Nele, pode-se verificar que a cor Produto pode ter o valor A ou B, enquanto a cor Resultado é uma tupla onde cada campo da tupla é um Produto. Neste nó também podem ser declaradas funções e constantes, caso estas precisem ser utilizadas no modelo. No modelo podemos verificar as inscrições dos arcos. O arco que liga a transição *Produz A* ao lugar P, por exemplo, coloca uma ficha do tipo Produto, com valor A, no lugar P. O arco de entrada da transição

Produz C consome uma ficha com valor A e uma ficha com valor B do lugar P, enquanto o arco de saída coloca uma ficha do tipo Resultado no lugar C.

Algumas extensões de redes de Petri têm sido propostas para facilitar a modelagem com CPN, dentre as quais podemos destacar as hierárquicas e as temporizadas. A redes de Petri coloridas hierárquicas (HCPN)[Jen92], permitem ao modelador a construção de modelos grandes combinando um número de pequenas redes de Petri coloridas em uma grande rede. A modelagem de sistemas usando HCPN pode ser desenvolvida usando uma abordagem *top-down* ou *bottom-up*. A HCPN possui dois mecanismos de estruturação: transições de substituição e lugares de fusão. As transições de substituição representam uma abstração de uma determinada parte do modelo, ou seja, cada transição de substituição é detalhada por um outro modelo CPN. Lugares de fusão permitem ao usuário especificar um conjunto de lugares que são idênticos. Quando uma ficha é adicionada/removida de um lugar de fusão, uma ficha idêntica será adicionada/removida em todos os outros lugares do modelo relacionados ao lugar de fusão. As transições de substituição são rotuladas com um "HS", enquanto os lugares de fusão são rotulados com um "FG".



(a) Modelo de alto-nível

(b) Detalhamento

Figura 4.3: Exemplo de uma rede de Petri colorida hierárquica: modelo de alto-nível com uma transição de substituição (a) e rede que detalha a transição de substituição *Produz A* (b).

Na Figura 4.3(a), temos o mesmo modelo apresentado anteriormente, com a diferença

de que a transição *Produz A* (rotulada com um "HS") é uma transição de substituição. Ela é detalhada pelo modelo apresentado na Figura 4.3(b). O lugar Montado é a porta de saída de seu modelos (rotulados "P Out"). Este lugar é relacionado ao lugar P, do modelo principal, que é o lugar de saída da transição de substituição. Desta forma, sempre que uma ficha for acrescentada ou retirada de Montado, o mesmo ocorre com o lugar P, e vice-versa. No modelo da Figura 4.3(b), temos ainda dois lugares de fusão (Estoque). Estes lugares são rotulados com um "FG"e funcionam como se fossem apenas um.

As redes de Petri temporizadas adicionam o conceito de tempo aos modelos CPN. Desta forma, existe um tempo global para o modelo, que pode ser alterado quando uma transição é disparada. As fichas também podem ter um tempo associado, que não é necessariamente o mesmo tempo global do modelo. No entanto, uma ficha só poderá ser utilizada para o disparo de uma transição, se o tempo associado à ela for menor ou igual ao tempo global do modelo.

4.3 Formalização de Padrões de Projeto para Agentes Móveis

Nesta seção, apresentaremos a formalização dos padrões de projeto para agentes móveis usando redes de Petri. Como exposto anteriormente, estes modelos podem ser utilizados como documentação do padrão, tendo a vantagem de não apresentarem ambigüidade. Além disto, a partir da modelagem com THCPN, poderão ser realizadas análises para validação do modelo e do padrão e, sendo estes modelos executáveis, simulações podem ser realizadas, facilitando o entendimento sobre a dinâmica do padrão.

A seguir apresentamos a formalização de três padrões de migração: *Itinerary*, *Star-Shaped* e *Branching*. Primeiramente serão apresentados os três padrões, em seguida os modelos em THCPN serão detalhados e, por fim, o processo de validação será explicado. A formalização dos demais padrões é apresentada no Apêndice B.

4.3.1 Padrões de Migração

Nesta subseção, introduziremos os três padrões de migração. Maiores detalhes sobre estes podem ser encontrados no Apêndice A.

Itinerary

Este padrão provê uma maneira de executar a migração de um agente, o qual será responsável por realizar uma dada tarefa em máquinas remotas. O agente recebe um itinerário na agência de origem, indicando a seqüência de agências que deve visitar. Uma vez em uma agência, o agente executa a tarefa localmente e então continua em seu itinerário. Após visitar a última agência, o agente retorna para a agência de origem. Este padrão é uma boa solução para agentes que precisam executar tarefas seqüenciais. Em [GMM03] e [Med03], estudos de caso que aplicam este padrão são mostrados.

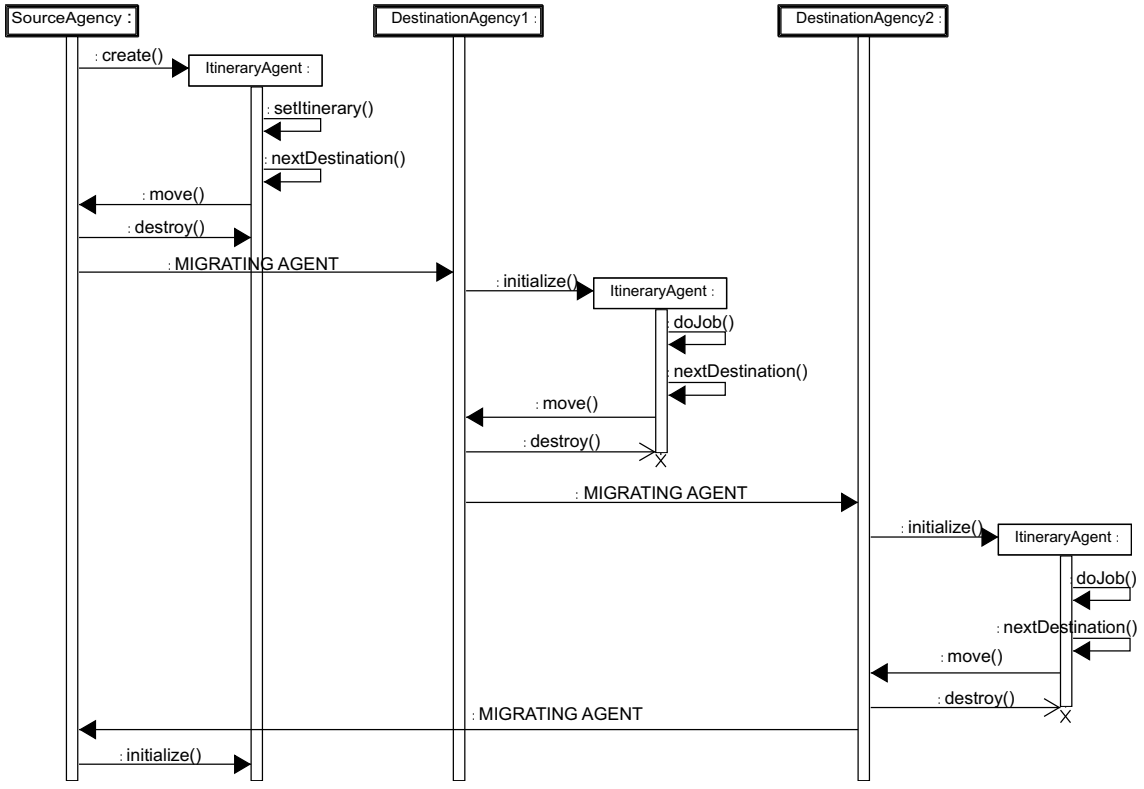


Figura 4.4: Diagrama de seqüências para o padrão *Itinerary*

Na Figura 4.4, está representada uma possível seqüência de execução para este padrão. Como pode-se ver, existem três agências: uma agência de origem (*SourceAgency*) e duas

agências onde realizará a tarefa (*DestinationAgency1* and *DestinationAgency2*). Seguindo o diagrama, percebe-se que existe um agente (*ItineraryAgent*) que define seu itinerário, migra para a primeira agência de destino, onde executa a tarefa. Então, o agente migra para a segunda agência de destino, executa a tarefa, e retorna para a agência de origem.

Star-Shaped

No padrão *Star-Shaped*, o agente recebe uma lista de agências para onde ele deve migrar. Primeiramente, o agente migra para a primeira agência em sua lista. Após completar a migração, ele executa a tarefa e torna a migrar, retornando para a agência de origem. O agente repete este ciclo até que a última agência de sua lista seja visitada. A vantagem deste padrão é que o agente armazena os resultados da tarefa na agência de origem, não precisando migrar para as demais agências com eles. Dependendo da aplicação, os resultados podem ser mostrados para o usuário assim que o agente os armazena. Desta forma, o usuário já pode saber os resultados parciais antes do agente terminar sua migração por todas as agências de destino.

Na Figura 4.5, é apresentada a seqüência de execução para o padrão *Star-Shaped*. Neste diagrama, temos a mesma configuração do diagrama de seqüências apresentado para o padrão *Itinerary*: 3 agências e um agente. Seguindo o diagrama, observa-se que o agente define seu itinerário, e então migra para a primeira agência de destino. Após executar a tarefa, o agente retorna para a agência de origem, onde armazena os resultados. Após isto, o agente migra para a segunda agência de destino, executa a tarefa, e retorna para a agência de origem, onde armazena os resultados.

Branching

No padrão *Branching*, o agente recebe uma lista de agências que deve visitar, e se clona de acordo com o número de agências em seu itinerário. Cada clone recebe uma agência da lista do agente original, para onde migra e executa a tarefa. Após executar a tarefa, cada agente retorna para a agência de origem. A importância deste padrão, é que ele pode dividir a tarefa que pode ser executada em paralelo. O tratamento final dos resultados é um detalhe não considerado por este padrão. Por exemplo, os agentes podem colocar o resultado da tarefa

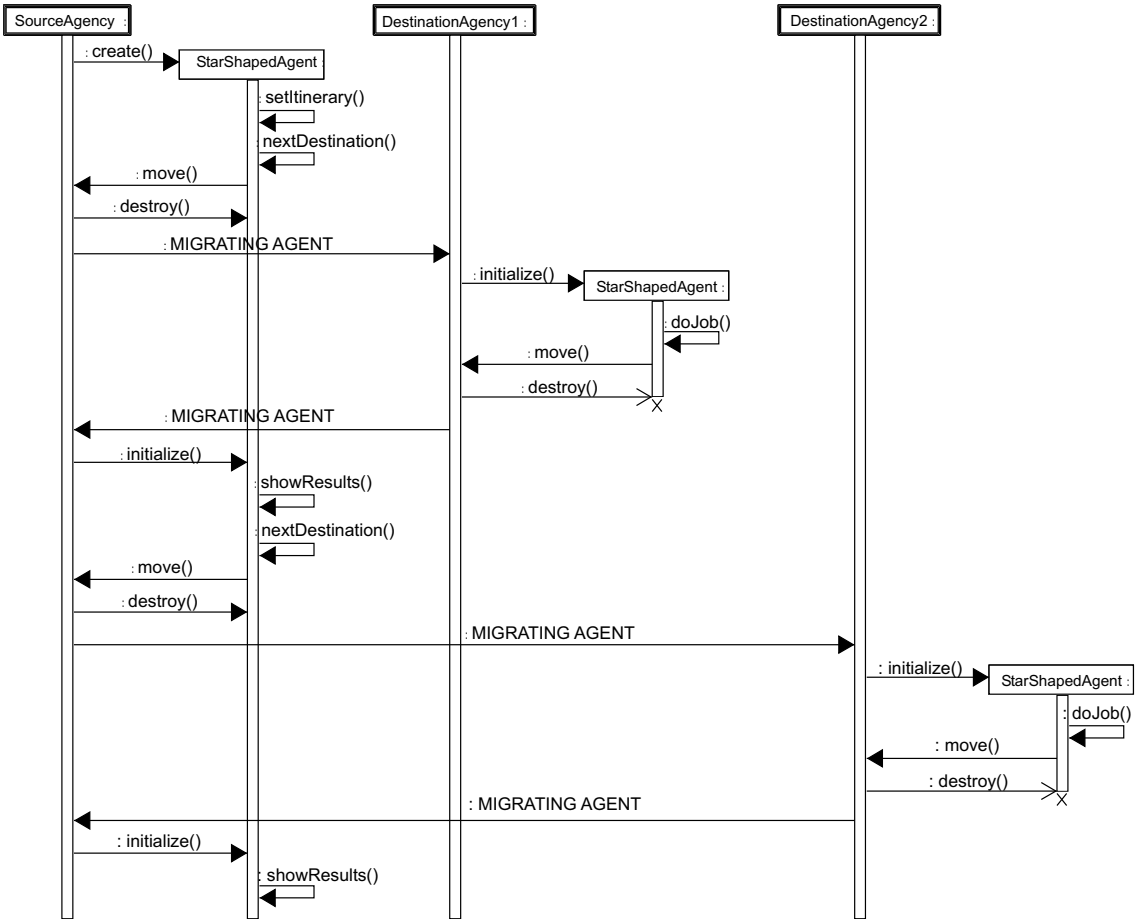


Figura 4.5: Diagrama de seqüências para o padrão *Star-Shaped*

em uma interface com o usuário ou enviar para outro agente.

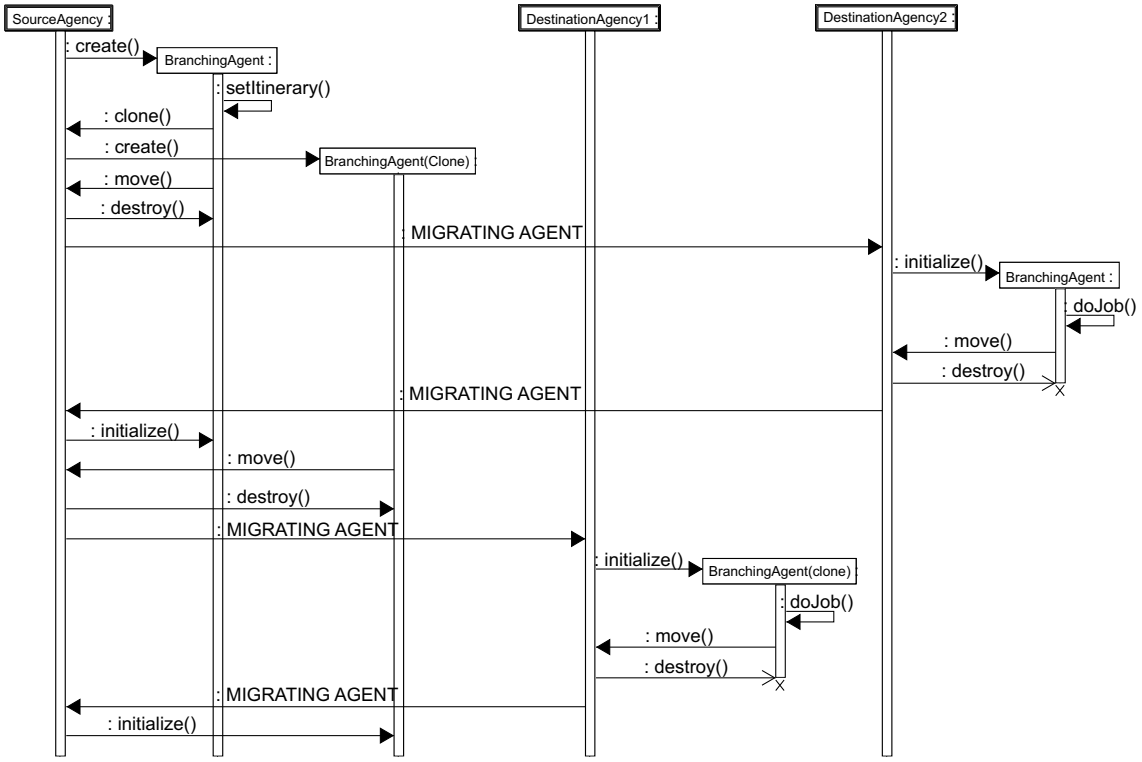


Figura 4.6: Diagrama de seqüências para o padrão *Branching*

Na Figura 4.6, é apresentada uma seqüência de execução para este padrão, em um cenário onde existem três agência e um agente. Seguindo a figura, pode-se ver que o agente define seu itinerário, e então se clona. Após isto, cada agente (o original e o clone) migra para uma agência de destino, onde executa a tarefa, e então retorna para a agência de origem.

4.3.2 Formalização dos Padrões de Migração

Na Figura 4.7, é mostrado o modelo de alto-nível que é comum aos três padrões de projeto. Este modelo consiste de dois lugares e três transições de substituição. Neste modelo, pode-se perceber três importantes elementos componentes dos padrões de migração: a agência de origem (de onde o agente é enviado para executar a tarefa), as agências de destino e a rede (através da qual os agentes irão migrar). Estes elementos são representados, respectivamente, pelas transições de substituição *SourceAgency*, *DestinantionAgency* e *Net*. O lugar *SA* (*Source Agency*) é usada para fins de inicialização. O rótulo "AGENT"indica a cor do lugar, significando que agentes poderão estar neste lugar. Já o rótulo "Agent", por sua vez,

indica a ficha que está, inicialmente, neste lugar, ou seja, o agente em seu estado inicial. O lugar *MA* (*Migrating Agent*) centraliza o fluxo dos agentes entre as agências. Este fluxo é controlado pelas agências e pela rede, como será detalhado mais a frente.

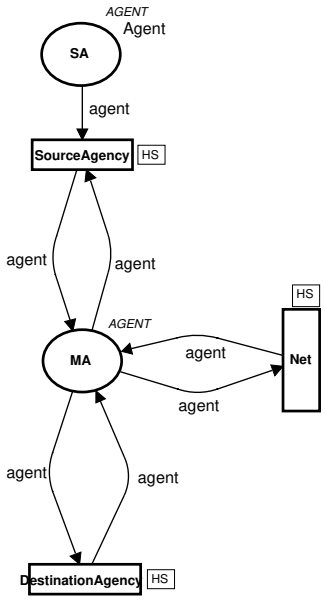


Figura 4.7: Modelo de alto-nível

A principal diferença na modelagem do padrões *Itinerary*, *Star-Shaped* e *Branching* está no modelo que detalha a transição de substituição *SourceAgency*. Lembre-se que, na agência de origem ocorre a definição de que agência o agente deve visitar, e então um procedimento diferente é adotado para os três padrões de projeto. Os modelos detalhados para as transições de substituição *Net* e *DestinationAgency* são similares para os três padrões.

A modelagem da rede é mostrada na Figura 4.8. A transição *Transmit* está associada a uma função de tempo. Sempre que esta transição dispara, o tempo global do modelo é incrementado de acordo com a função *migrateTime(agent)*. Esta função pega o tamanho do agente (código + dados) para calcular, de acordo com a capacidade média da rede, o tempo médio de duração da migração. A ficha no lugar *MA* é atualizada da seguinte maneira: o itinerário do agente e sua localização sao atualizados (função *updateAgency(agent)*), e seu estado é alterado de *migrating* para *executing*[2] (função *changeState(agent)*). Note que o agente só será transmitido, isto é, a transição *Transmit* dispara, se o estado do agente é

[2]Um agente possui dois estados: *executing*, que indica que o agente está executando ou pronto para executar; e *migrating*, que indica que o agente está migrando ou pronto para migrar.

migrating. Esta condição é garantida pela guarda *[not(isExecuting(agent))]*.

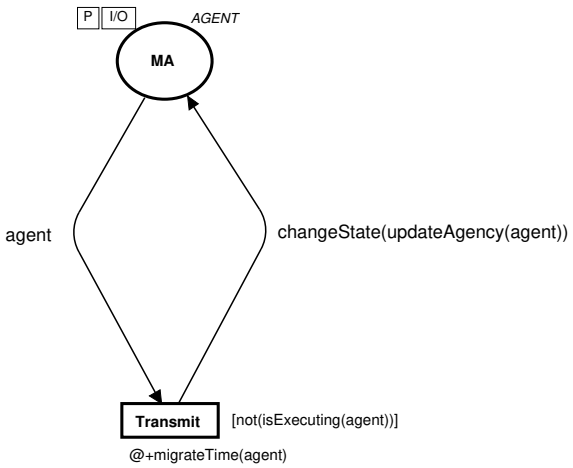


Figura 4.8: Modelo da rede

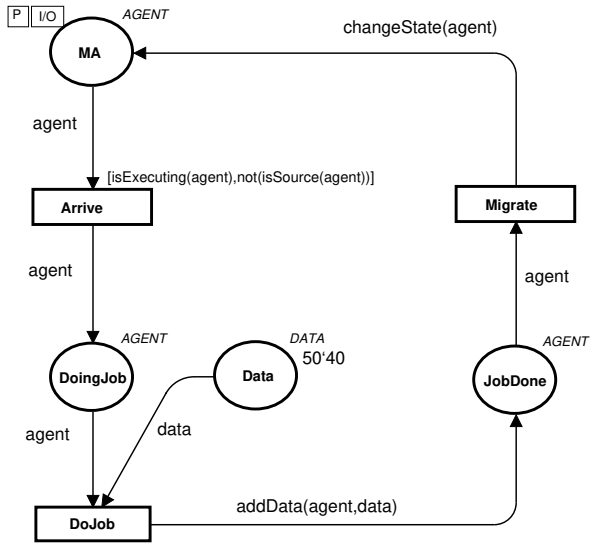


Figura 4.9: Modelo da agência de destino

Sempre que um agente chega na *DestinationAgency* (Figura 4.9), o agente é recebido pela transição *Arrive* e executa sua tarefa (transição *doJob*). Esta transição recupera um dos dados no lugar *Data*. Este dado é adicionado ao agente pela função *addData(agent, data)*. Depois disto, o agente migra para a próxima agência em seu itinerário (transição *Migrate*), e seu estado é alterado de *executing* para *migrating* (função *changeState(agent)*). Note que o agente só será recebido pela agência, isto é, a transição *Arrive* apenas dispara, se seu estado é *executing* e sua localização atual não for a agência de origem (guarda *[(isExecuting(agent)), not(isSource(agent))]*). Como as agências de destino são representadas pela mesmo modelo, não é feita distinção entre elas, apenas precisando verificar se um agente está na agência de origem ou não.

Modelo da Agência de Origem para o Padrão *Itinerary*

Na Figura 4.10, o modelo da agência de origem para o padrão *Itinerary* é detalhado. Inicialmente, o agente que irá realizar a tarefa é representado por uma ficha no lugar *SA*. O agente, primeiramente, define seu itinerário (transição *SetItinerary*), isto é, recebe a lista de agências onde deverá executar a tarefa. Neste modelo, o itinerário é armazenado no lugar *Itinerary*, e a função *setItinerary(agent,itinerary)* informa o itinerário ao agente. Então o agente migra para a primeira agência de sua lista (transição *Migrate*), e a função *changeState(agent)*

altera o estado do agente para *migrating*. Tão logo termine a migração através de todas as agências em sua lista, o agente retorna para a agência de origem (transição *ReceiveAgent*). Então, os resultados da tarefa são colocados no lugar *Result* (transição *GetResult* e função *getData(agent)*). Note que o agente só é recebido pela agência, isto é, a transição *ReceiveAgent* somente dispara, se o estado do agente for *executing* e sua localização atual for a agência de origem (guarda *[(isExecuting(agent)), isSource(agent)]*).

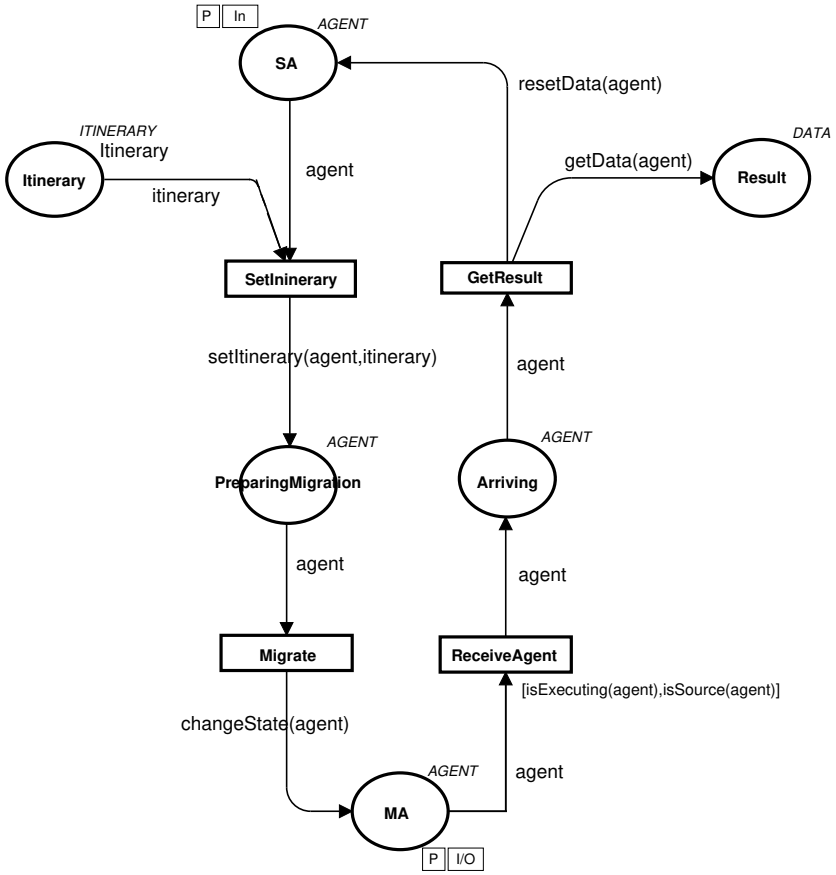


Figura 4.10: Modelo da agência de origem para o padrão *Itinerary*

Modelo da Agência de Origem para o Padrão *Star-Shaped*

A diferença deste padrão é que, como explicado, o agente sempre retorna para a agência de origem após visitar cada agência em seu itinerário. No modelo da agência de origem para o padrão *Star-Shaped* (Figura 4.11), a transição *ReceiveResult* coleta os resultados da tarefa (função *getData(agent)*), e remove estes resultados do agente (*resetData(agent)*). Isto significa que o agente não precisa migrar com os resultados já armazenados. Esta transição

também altera o estado do agente para *migrating* (*changeState*), e o envia para a próxima agência em seu itinerário. Note que o agente só irá ser recebido pela agência para a coleta dos resultados, isto é, a transição *ReceiveResult* dispara, se o estado do agente for *executing* e sua localização atual for a agência de origem (guarda *[(isExecuting(agent)) , (isSource(agent))]*). Para finalizar sua migração, isto é, para a transição *ReceiveAgent* disparar, é necessário que o estado do agente seja *executing* e seu itinerário esteja finalizado (guard *[(isExecuting(agent)), (isEnd(agent))]*).

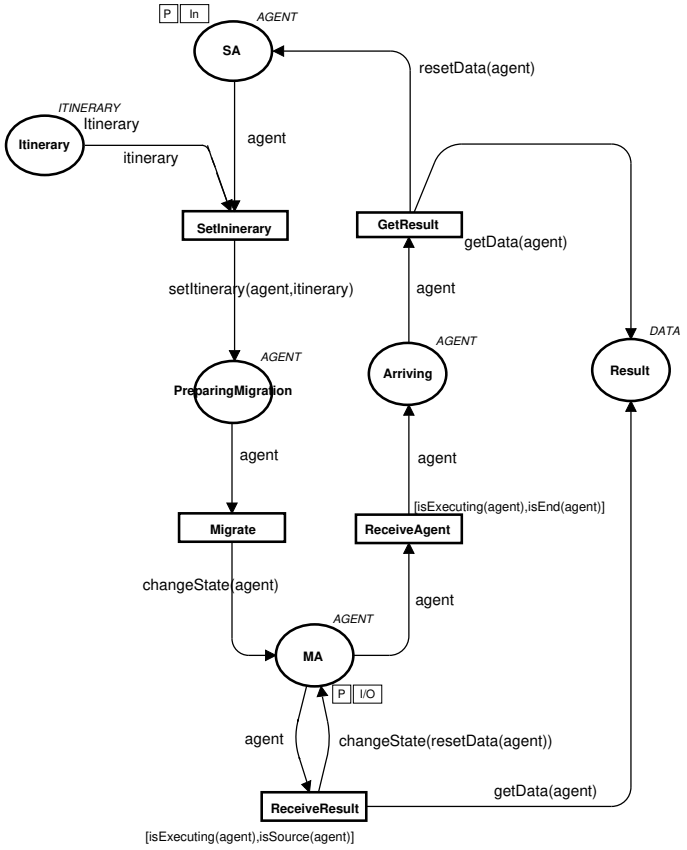


Figura 4.11: Modelo da agência de origem para o padrão *Star-Shaped*

Modelo da Agência de Origem para o Padrão *Branching*

Como explicado, neste padrão, o agente original se clona de acordo com o número de agências em seu itinerário. O processo de clonagem, cuja rede de Petri é mostrada na Figura 4.12, é modelado pela transição temporizada *Clone*, que incrementa o tempo global de acordo com o valor da constante *cloningDuration*. A função *removeFirstAgency(agent)* remove a

primeira agência da lista, de modo que nenhum outro agente receba esta agência como destino. Por outro lado, a função *doClone(agent)* cria um novo agente, clone do agente original, entretanto com o itinerário contendo apenas a primeira agência do itinerário do agente original. A guarda *[itinerarySize(agent)>1]*, da transição *Clone*, garante que apenas o número necessário de clones será criado. O clone migra através da transição *Migrate_Clones*, enquanto o original migra através da transição *Migrate*, quando todos os clones necessários forem criados (guard *[itinerarySize(agent)=1]*).

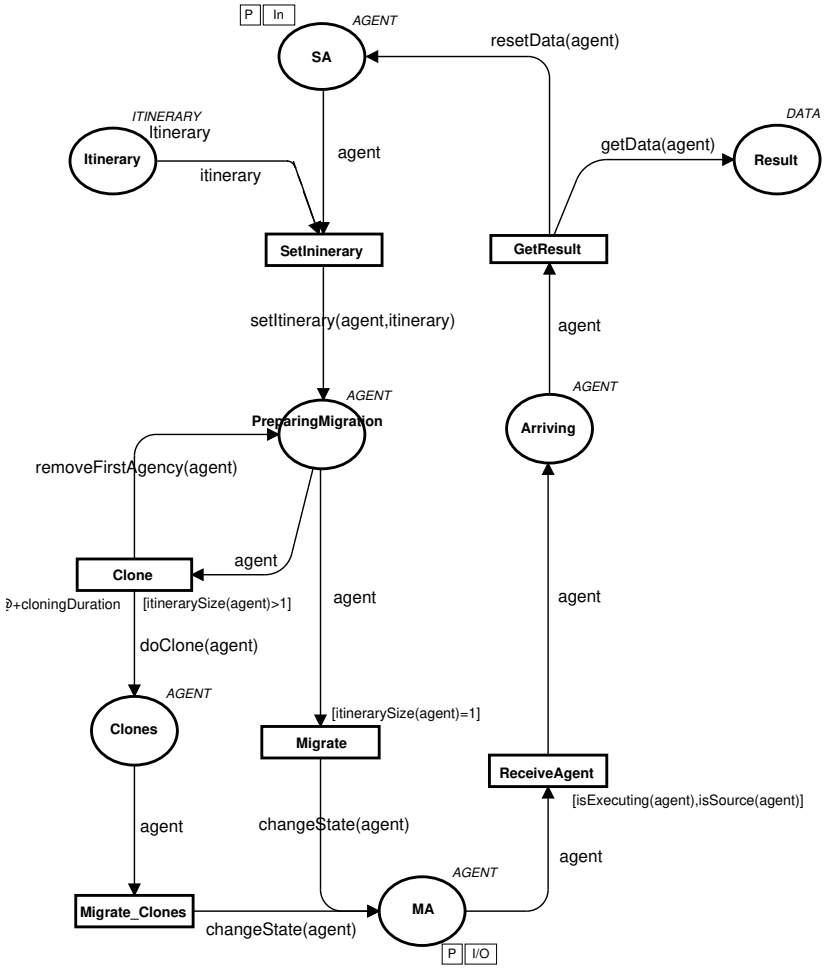


Figura 4.12: Modelo da agência de origem para o padrão *Branching*

4.3.3 Validação dos Modelos

Para analisar os modelos foram utilizadas técnicas formais e informais. A análise informal foi realizada através da simulação do modelo. A simulação é uma maneira muito eficiente

Padrão	*Itinerary*	*Star-Shaped*	*Branching*
Num. de Agências	**5**		
Nós	26	34	44994
Arcos	25	33	196673
Num. de Agências	**15**		
Nós	66	94	45120 (parcial)
Arcos	65	93	174902 (parcial)
Num. de Agências	**45**		
Nós	186	270	63965 (parcial)
Arcos	185	269	231703 (parcial)

Tabela 4.1: Resumo do grafo de ocorrência para o padrão *Branching*

de depurar um modelo de THCPN. Foram combinadas simulações interativas e automáticas durante a fase de validação. Através destas simulações foi possível investigar detalhes dos modelos e, pela escolha de cenários apropriados, foi possível obter uma melhor compreensão sobre o comportamento dos padrões modelados, corrigindo erros na modelagem e aumentando a confiança que os modelos estão corretos. Vários cenários de itinerários, incluindo diferente número de agências a serem visitadas, foram considerados.

A análise formal foi realizada usando o método de grafo de ocorrência (OG)[3]. Para cada modelo dos padrões de projeto foi gerando um grafo de ocorrência. A Tabela 4.1 resume os principais resultados desta análise. Considerando 5 agências, foi possível gerar o grafo completo para os três modelos de padrões de projeto. O grafo de ocorrência para os modelos dos padrões *Itinerary* e *Star-Shaped* têm 26 e 34 nós, respectivamente. O grafo para o padrão *Branching* tem 44.994 nós. Essa grande diferença no número de nós do OG dos modelos é devida ao fato de que os padrões *Itinerary* e *Star-Shaped* têm execução seqüencial, enquanto o *Branching* tem execução concorrente, gerando um maior número de marcações alcançáveis. Considerando a configuração com 15 e 45 agências, não foi possível gerar o grafo completo para o modelo do padrão *Branching*. Também foi utilizado o relatório padrão gerado pela ferramenta de grafo de ocorrência, para analisar algumas propriedades

[3]A idéia básica por trás do grafo de ocorrência é construir um grafo direcionado com um nó para cada marcação alcançável e um arco para cada elemento de ligação [JCK96].

do modelo. Focou-se na prova de que o modelo pára de forma consistente quando todos os resultados forem retornados para a agência de origem.

Na Figura 4.13 é apresentada parte do relatório gerado pela ferramenta Design/CPN, considerando o padrão *Branching* com 5 agências para serem visitadas. A partir do relatório, pode-se observar que existe apenas um estado de passagem (*home marking*)[4] e uma marcação morta[5]. Este estado de passagem corresponde ao estado final, quando os resultados são enviados de volta para a agência de origem. Uma vez que o estado final é um estado de passagem, é garantido que o estado final é sempre alcançável. A partir das propriedades de limitação (*Boundeness Properties*) foi possível observar que apenas 4 agentes foram clonados, como definido pelo padrão de projeto *Branching*.

```
Statistics
------------------------

 Occurrence Graph
    Nodes:  44994
    Arcs:   196673
    Secs:   125
    Status: Full

 Scc Graph
    Nodes:  44994
    Arcs:   196673
    Secs:   17
```

```
Boundedness Properties
-----------------------------------------

 Best Integers Bounds    Upper      Lower
 SourceAgency'Arriving     5          0

 SourceAgency'Clones       4          0
 SourceAgency'Itinerary    1          0

 SourceAgency'PreparingMigration
                           1          0
 SourceAgency'Result       5          0

Home Properties
-----------------------------------------

 Home Markings:  [44994]

Liveness Properties
-----------------------------------------

 Dead Markings:  [44994]
 Dead Transitions Instances: None
 Live Transitions Instances: None
```

Figura 4.13: Relatório para o padrão *Branching*

4.4 Conclusão

Neste capítulo foi apresentada a formalização de três padrões de projeto para agentes móveis. Como exposto, a formalização de padrões e agentes móveis vêm sendo abordada em diversos

[4]Estado de passagem é uma marcação alcançável a partir de qualquer marcação alcançável do modelo.

[5]Marcação morta é uma marcação onde nenhuma transição está habilitada.

trabalhos devido as várias vantagens que pode trazer.

As redes de Petri coloridas hierárquicas temporizadas se mostraram adequadas para modelar os padrões, bem como os agentes móveis, capturando seus principais conceitos, como migração e clonagem. Desta forma, os modelos formais dos padrões de projeto para agentes móveis podem facilitar bastante o entendimento da dinâmica do funcionamento destes, através da realização de simulação.

Além disto, pôde-se tirar proveito das técnicas de análise estruturais, como mostrado, validando tanto o modelo quanto o padrão. Além das análises apresentadas neste trabalho, a partir dos modelos gerados, também puderam ser realizadas análises de desempenho, comparando o comportamento dos diversos padrões, como mostrado em [LdFG04]. A partir destas análises de desempenho, puderam ser verificadas características importantes sobre o comportamento dos padrões, como por exemplo, a grande vantagem em relação ao desempenho do padrão *Branching* na realização de tarefas paralelas, mesmo com um número reduzido de agências, e apesar do tráfego gerado pela existência de vários agentes clones na rede.

O grande diferencial de nosso trabalho em relação a outros já realizados, é que, enquanto estes se concentram na utilização do modelo formal para um único fim, seja ele descrição ou análise, neste trabalho, buscamos definir modelos formais que pudessem ser utilizados para vários fins: descrição, simulação, análises estruturais e análises de desempenho. Alguns problemas em relação à modelagem dos agentes móveis usando o método formal escolhido podem ser destacados. A forma de modularização dos modelos (em hierarquia) poderia ser mais intuitiva se seguisse a modularização já definida na modelagem UML, evitando, desta maneira, a sua redefinição. Extensões de redes de Petri, como as redes de Petri orientadas a objeto [Gue02b], possibilitam este tipo de modularização, no entanto, ainda não apresentam ferramentas adequadas para sua devida utilização. Uma limitação das redes de Petri, é a necessidade de se definir uma estrutura estática para o modelo, impossibilitando a criação dinâmica de novas redes e ligações entre agentes e agências. Por fim, em relação à analise de desempenho, apesar de ser possível a sua realização com redes de Petri coloridas, outras extensões, como redes de Petri estocásticas, têm se mostrado mais adequadas.

Uma abordagem que pode tornar mais interessante estas formalizações é a combinação das mesmas. Através desta combinação pode-se verificar como vários padrões de projeto

para agentes móveis se comportam juntos. Assim, sistemas reais que utilizem agentes móveis, podem ser formalizados pelas combinações dos padrões. Isto permite que o projeto do sistema seja validado e seu comportamento seja melhor entendido e analisado.

No próximo capítulo, é apresentado um estudo de caso de um sistema baseado em agentes móveis. Este estudo de caso tem seu projeto definido pela combinação de padrões de projeto para agentes móveis, e é formalizado para um estudo do seu comportamento.

Capítulo 5

Estudo de Caso

Neste capítulo, um estudo de caso será apresentado para a aplicação dos padrões de projeto para agentes móveis. Este estudo de caso é um sistema de recuperação de URLs de imagens e, para este sistema, duas soluções são propostas. Estas soluções são modeladas a partir da combinação de padrões de projeto para agentes com a ajuda do catálogo destes padrões. Em seguida, estas soluções são formalizadas, validadas, e um estudo comparativo é realizado para verificação de como as soluções se comportam diante de alguns cenários.

5.1 Descrição do problema

O estudo de caso utilizado neste trabalho consiste de um sistema de recuperação de URL (*Unified Resource Location*) de imagens na *Internet*, a partir de arquivos HTML, apresentado em [BG03]. Na Figura 5.1, um possível cenário para este sistema é apresentado. Nele, estão presentes os principais elementos que compõem o sistema:

- **Banco de Dados (DB)**: Responsável por armazenar URLs de imagens recuperadas pelo sistema e URLs bases, a partir das quais serão feitas as buscas por URLs de imagens;
- **Servidor (*Server*)**: Responsável pelo acesso ao banco de dados. Realiza a recuperação de URLs bases para pesquisa, e o armazenamento de URLs de imagens recuperadas. Podem existir mais de um servidor acessando o banco de dados;

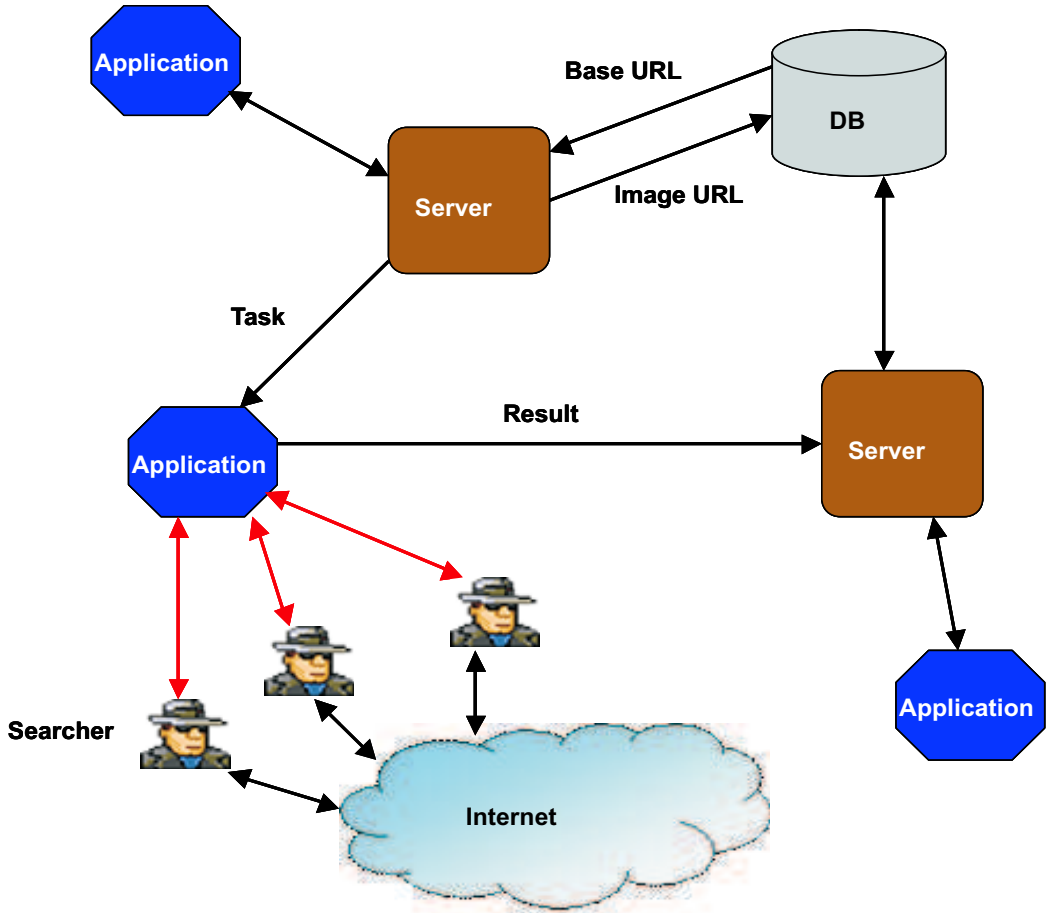


Figura 5.1: Cenário para o sistema de recuperação de URLs de imagens

- **Aplicação (*Application*)**: Solicita URLs bases ao servidor para realizar buscas por URLs de imagens. Essas URLs bases são passadas para os buscadores.
- **Buscador (*Searcher*)**: Realiza a busca por URLs de imagens de acordo com as URLs bases recebidas. Após a busca, entrega os resultados para a aplicação.

Uma possível seqüência de execução do sistema pode ser vista no diagrama da Figura 5.2. Neste diagrama estão presentes os quatro elementos apresentados anteriormente. Inicialmente, a aplicação solicita uma URL base para o servidor através do método *getTask()*. O servidor, por sua vez, acessa o banco de dados para recuperação da URL (método *getURL()*), e a retorna para a aplicação. Então, a aplicação cria um buscador, repassando a URL para ele (método *create(URL)*). Durante a realização da busca, o buscador procura URLs de imagens, bem como novas URLs base, a partir das quais novas buscas poderão ser realizadas. Ao término da mesma, o buscador envia os resultados para a aplicação, que repassa para o

servidor atualizar o banco de dados.

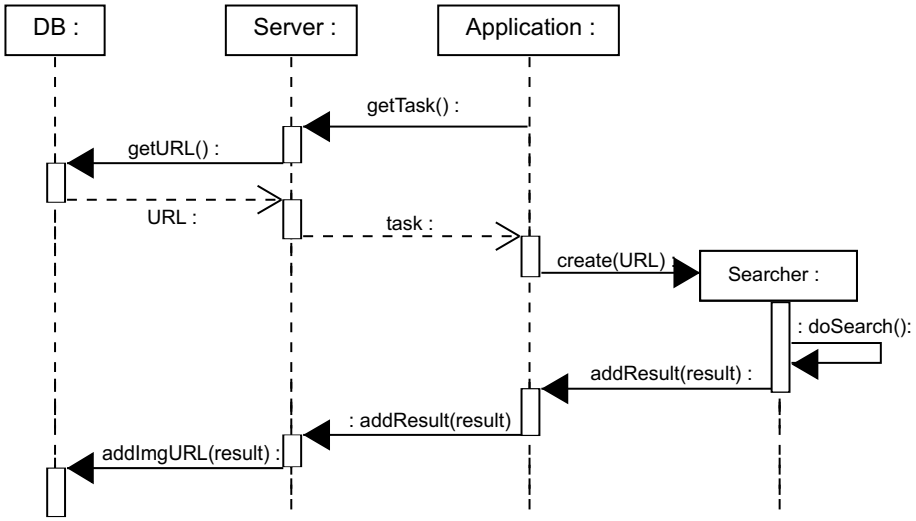


Figura 5.2: Diagrama de seqüências para uma possível execução do sistema do estudo de caso

É importante destacar que, juntamente com as URLs base solicitadas, a aplicação recebe uma lista de servidores disponíveis no momento. Assim, caso o contato com o servidor esteja indisponível por algum motivo (problemas de conexão, por exemplo), os resultados podem ser enviados para um outro servidor.

As URLs base armazenadas no banco de dados possuem a informação do seu estado (não pesquisada, pesquisando ou pesquisada). Assim, evita-se que duas aplicações recebam a mesma URL base para busca.

A seguir, temos duas possíveis soluções para o estudo de caso usando agentes móveis e padrões de projeto para agentes móveis.

5.2 Soluções Propostas

Para definição das soluções, foram utilizados padrões de projeto para agentes móveis retirados do catálogo. Para identificação dos padrões a serem utilizados, o catálogo foi de fundamental importância. O primeiro passo foi identificar o(s) problema(s) a ser(em) solucionado(s). A partir do problema, verifica-se que categoria de padrões que se relaciona com o tipo do problema (problema de interação entre agentes, por exemplo), e de acordo com

a documentação, determina-se que padrão ou padrões propõem solução para o problema encontrado.

Após a escolha dos padrões, os modelos independentes de plataformas providos pela documentação dos padrões no catálogo foram combinados, juntamente com outras classes específicas da aplicação e que não estavam presentes nos padrões, de forma a compor uma possível solução para o estudo de caso. Isto basicamente consistiu em montar os modelos dos diagramas de classe providos pela documentação com classes específicas do domínio da aplicação. Diagramas de seqüências foram criados para ilustrar os cenários, baseados na combinação de seqüências de cada padrão utilizado, juntamente com interações e classes específicas do sistema [LMdFS04].

Durante o desenvolvimento de soluções, novos problemas são encontrados, podendo levar à identificação de novos padrões ou variantes de padrões catalogados. É importante que estes sejam documentados no catálogo.

5.2.1 Primeira Solução

No sistema do estudo de caso, três grandes problemas podem ser identificados: acesso ao banco de dados, gerenciamento da realização da busca e forma de realização da busca. A partir destes problemas, as categorias de padrões que podem ser utilizados na solução é definida. No caso, são necessários padrões das categorias de recurso, tarefa e migração. A partir do catálogo e da documentação dos padrões, um padrão de cada categoria identificada é escolhido. Na primeira solução, são utilizados os padrões *Master-Slave* (tarefa) e *Branching* (migração), já explicados anteriormente, e o padrão *MoProxy*[1] (recurso). A identificação destes padrões no sistema pode ser vista no cenário da Figura 5.3. Neste cenário, pode-se verificar que o padrão *MoProxy* é utilizado para acesso ao banco de dados, o padrão *Master-Slave* é utilizado para controle, pela aplicação, da execução da tarefa pelo buscador, e o padrão *Branching* é utilizado pelo buscador para realização da tarefa.

Como explicado anteriormente, o diagrama de classes é definido de acordo com os mode-

[1]No padrão *MoProxy*, quando um agente precisa de um recurso, ele requisita para o concessor de recursos, indicando as permissões desejadas. Então, o concessor de recursos retorna um proxy móvel para o agente, de forma que ele acesse o recurso com as permissões desejadas, de acordo das restrições do recurso (detalhes no Apêndice A).

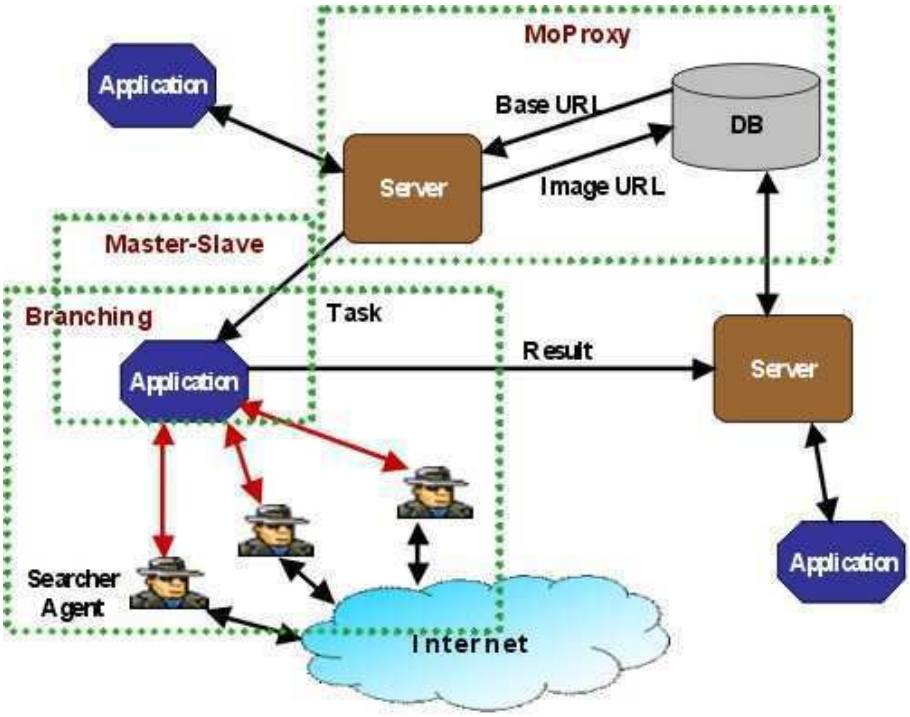


Figura 5.3: Identificação dos padrões da primeira solução para um cenário do sistema

los independentes presentes no catálogo juntamente com novas classes relativas ao domínio da aplicação. Pode-se observar no diagrama de classes da Figura 5.4 que, nesta solução, o servidor possui um agente *ServerAgent*, responsável pelo acesso ao banco de dados. Também estão presentes as classes definidas pelo padrão *MoProxy* (*ResourceGranter* e *MoProxy*). Já na aplicação, temos um agente da aplicação (*ApplicationAgent*), e um agente de busca (*SearcherAgent*). A relação entre o agente de aplicação e o agente de busca é definida pelo padrão *Master-Slave*, sendo o agente da aplicação o mestre e o agente de busca, o escravo. Como podemos ver no diagrama de classes, além de ser um agente escravo, o agente de busca também é um *BranchingAgent*.

A seguir temos a apresentação de uma possível seqüência de execução para este sistema, onde o funcionamento da primeira solução poderá ser melhor entendido.

Seqüência de execução para a primeira solução

Na Figura 5.5, temos o diagrama que representa a inicialização do servidor, onde o agente concessor de recursos (*ResourceGranterAgent*) e o agente servidor (*ServerAgent*) são criados na agência servidor (*ServerAgency*). Após esta criação, o agente servidor solicita, ao

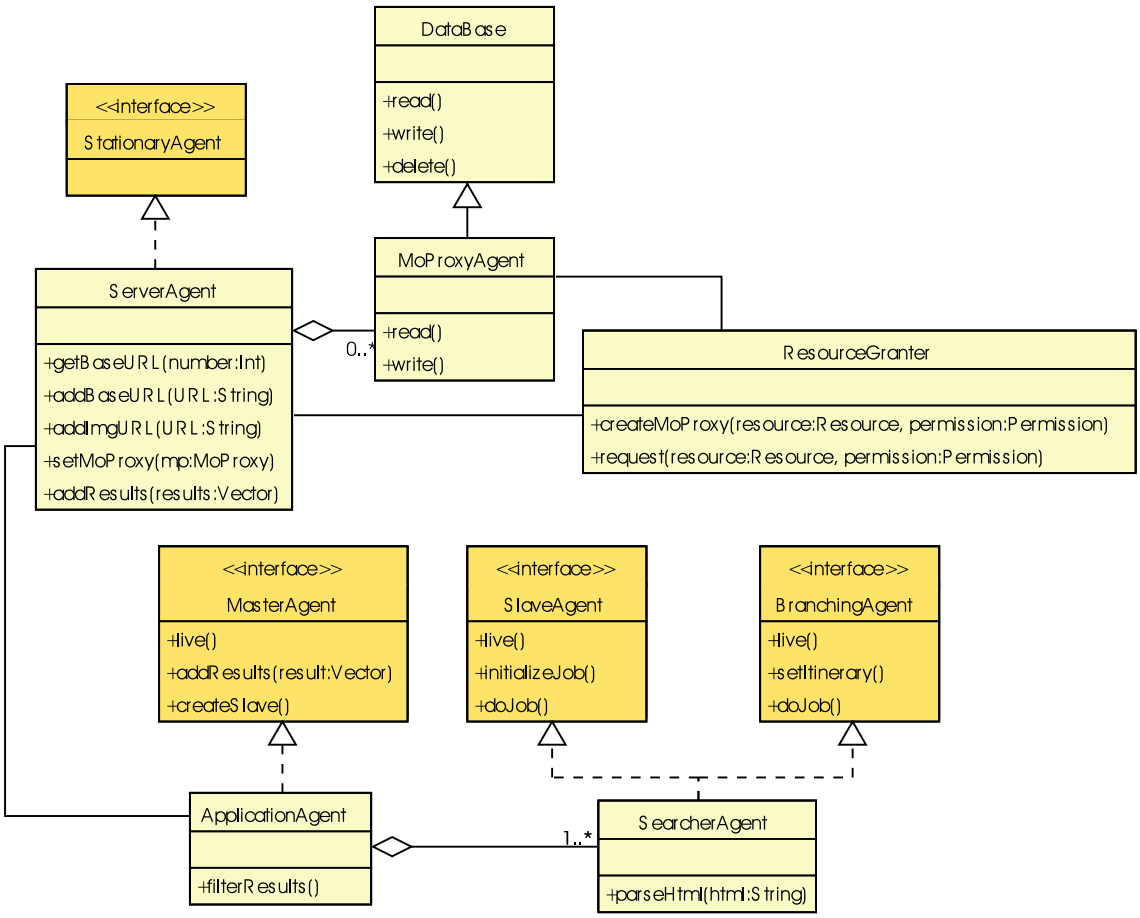


Figura 5.4: Primeira solução: diagrama de classes para o sistema

concessor de recursos, a criação do agente *MoProxy* (*MoProxyAgent*), definindo o tipo de recurso que necessita e as permissões que deseja (método *request(DB,"rw")*). Após a criação do *MoProxy* (método *createMoProxy(DB,"rw")*), o agente servidor define seu *MoProxy* de acordo com o definido pelo concessor de recursos (método *setMoProxy(moProxy)*), e está pronto para acessar o banco de dados.

Após a inicialização do servidor, o agente da aplicação (*ApplicationAgent*), que está na agência da aplicação (*ApplicationAgency*), já pode solicitar URLs base. Como mostrado do diagrama da Figura 5.6, esta solicitação é realizada através do envio de uma mensagem ao agente servidor indicando a quantidade de URLs desejadas. Ao receber esta mensagem, o agente servidor acessa o banco de dados, através do agente *MoProxy* e recupera as URLs base solicitadas, as enviando para o agente da aplicação.

Ao receber as URLs base solicitadas (Figura 5.7), o agente da aplicação cria um agente de busca (*SearcherAgent*), através do método *createSlave()*. Ao ser criado, o agente de busca

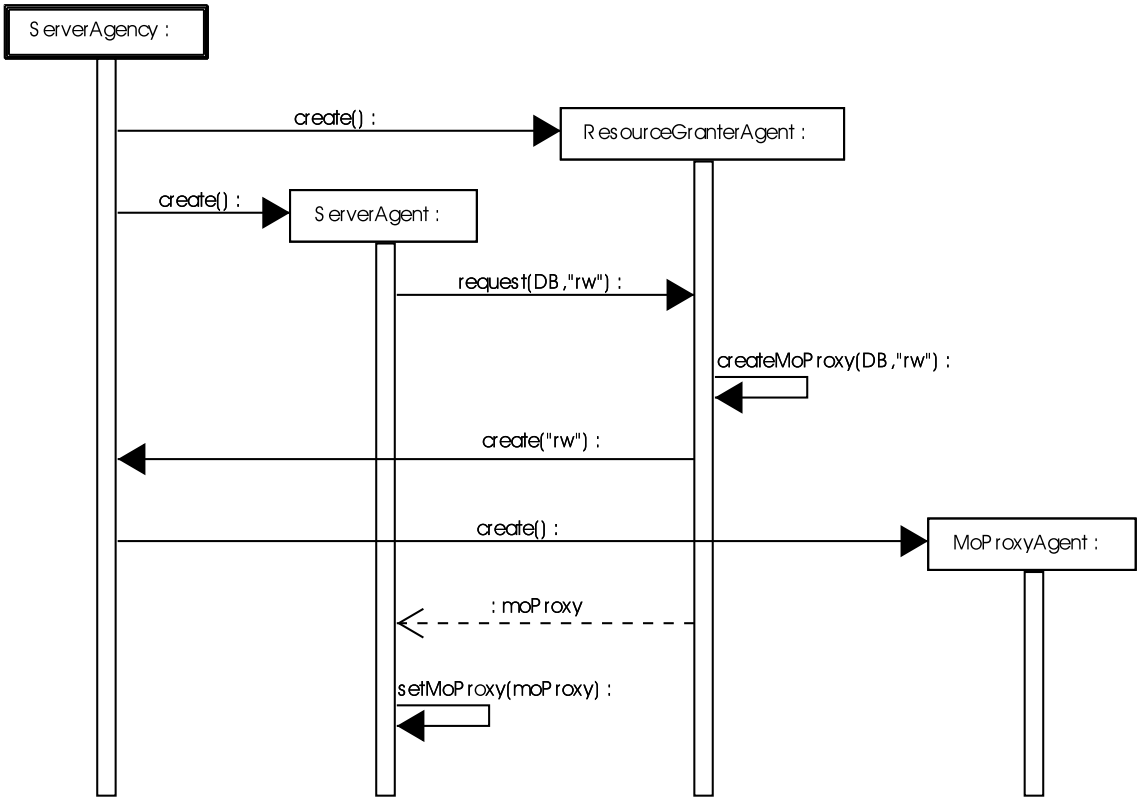


Figura 5.5: Primeira solução: diagrama de seqüências para a inicialização do servidor

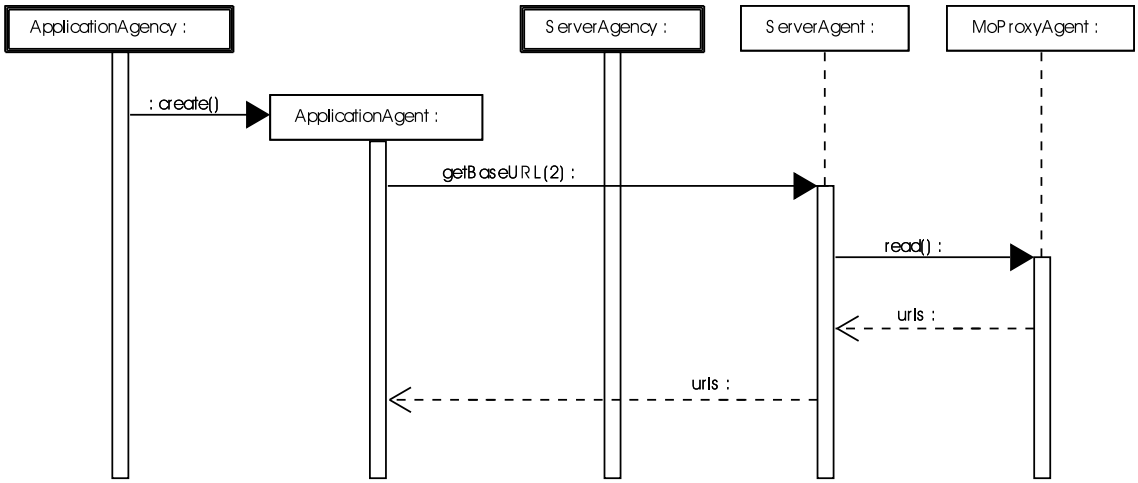


Figura 5.6: Primeira solução: diagrama de seqüências para a requisição de URLs base

define seu itinerário de acordo com as URLs recebidas (método *setItinerary*), e se clona, passando para o agente clone, uma das duas ULRs de seu itinerário.

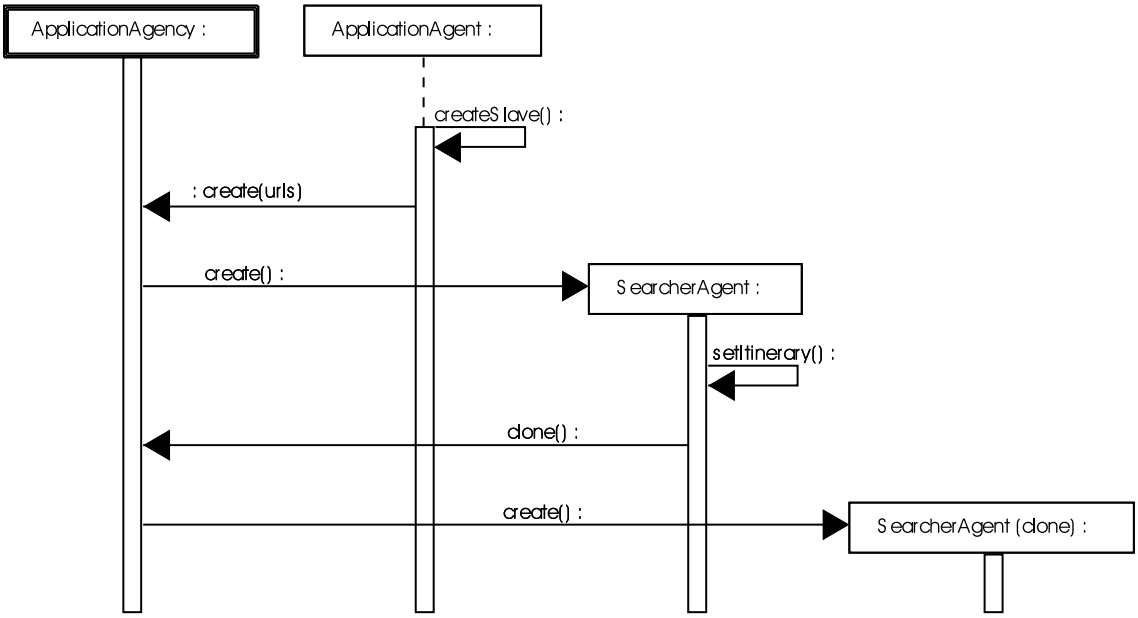


Figura 5.7: Primeira solução: diagrama de seqüências para a criação dos agentes buscadores

A partir de então, cada agente de busca (o original e o clone) irá migrar pra a agência da URL que possui (*RemoteAgency1* e *RemoteAgency2*), como definido pelo padrão *Branching* e mostrado nas Figuras 5.8 e 5.9. Chegando na agência remota, cada agente irá realizar a tarefa de busca de URLs de imagens (métodos *doJob()* e *parseHtml()*). Após realizar a tarefa, os agentes retornam para a agência da aplicação, entregam o resultado para o agente da aplicação (método *addResult(result)*), e se destroem. Ao receber os resultados dos agentes, o agente da aplicação filtra estes resultados (método *filterResults()*), eliminando possíveis redundâncias.

Tendo os resultados finais da busca filtrados, o agente da aplicação envia os resultados para o agente servidor. Ao receber estes resultados, o agente servidor armazena as URLs de imagens recuperadas (método *addImgURL(url)*), e também armazena as novas URLs base (método *addBaseURL(url)*).

5.2.2 Segunda Solução

A escolha dos padrões para a segunda solução é realizada como mostrado para a primeira solução. A diferença desta solução está na escolha do padrão de migração. Nesta solução,

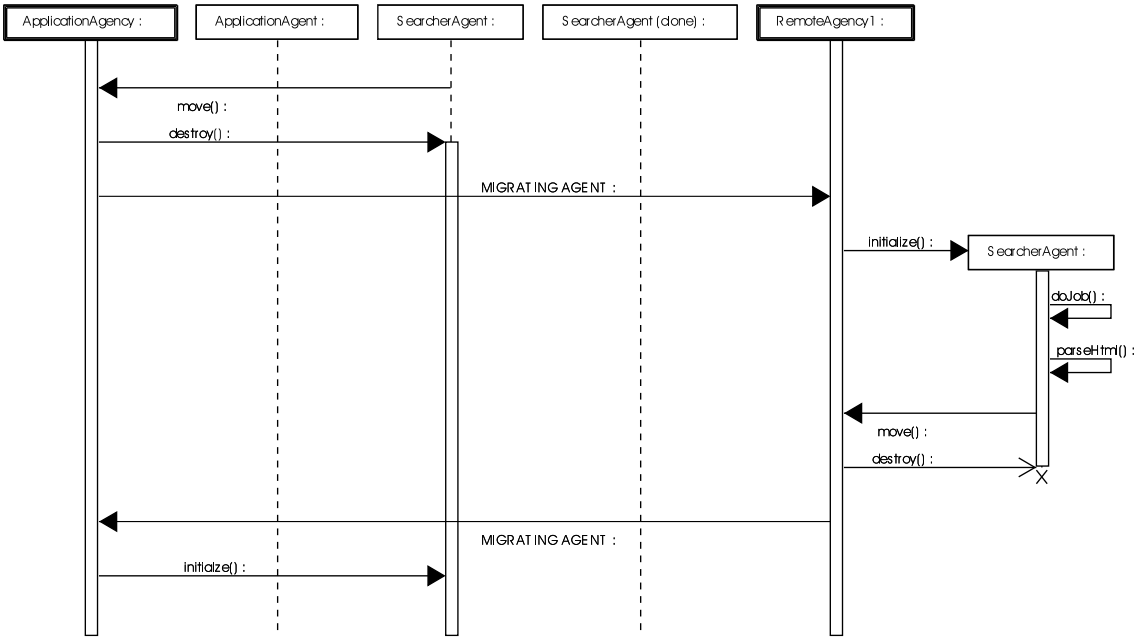


Figura 5.8: Primeira solução: primeira parte do diagrama de seqüências para recuperação de URLs de imagens

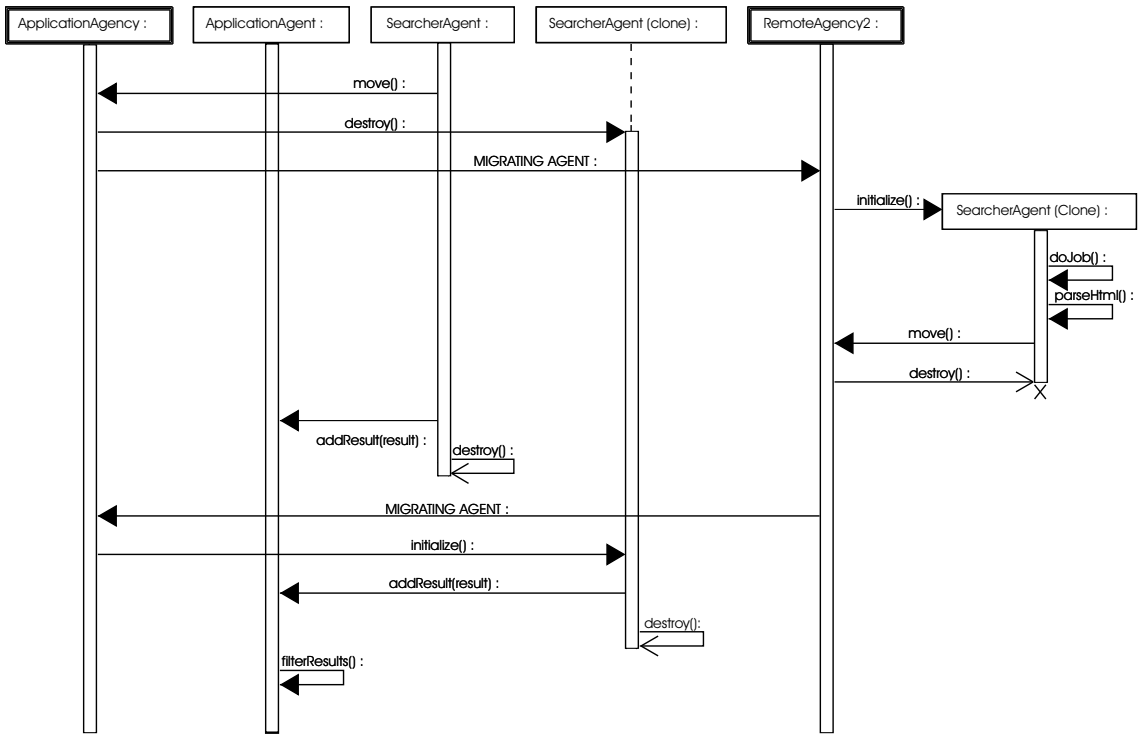


Figura 5.9: Primeira Solução: Segunda parte do diagrama de seqüências para recuperação de URLs de imagens

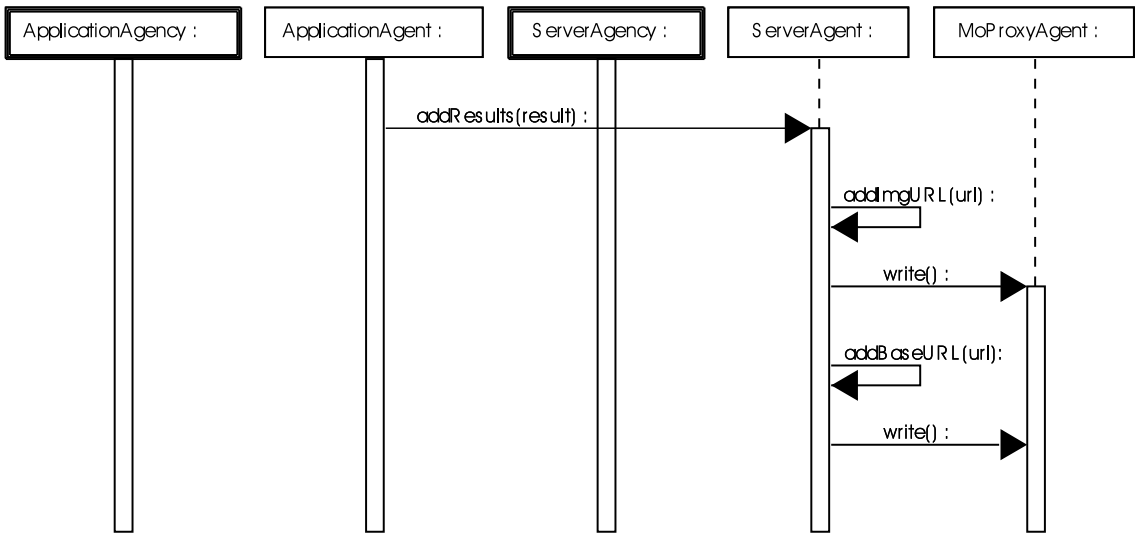


Figura 5.10: Primeira solução: diagrama de seqüências para o armazenamento das URLs de imagens recuperadas

temos a utilização dos padrões *MoProxy*, *Master-Slave* e *Star-Shaped*. A identificação destes padrões no sistema pode ser vista no cenário da Figura 5.11. Nela, pode-se verificar que o padrão *MoProxy* é utilizado para acesso ao banco de dados, o padrão *Master-Slave* é utilizado para controle, pelo servidor, da execução da tarefa pelo buscador, e o padrão *Star-Shaped* é utilizado pelo buscador para realização da tarefa. O padrão *Master-Slave* aplicado nesta solução é, na verdade, uma adaptação do padrão original, como será explicado mais a frente.

Como pode-se observar no diagrama de classes da Figura 5.12, nesta solução, assim como na anterior, o servidor possui um agente *ServerAgent*, responsável pelo acesso ao banco de dados. Para este acesso, o padrão *MoProxy* é utilizado. Já na aplicação, temos uma entidade não-agente , chamada *Application*, e um agente de busca (*SearcherAgent*). A relação entre o agente servidor e o agente de busca é definida pelo padrão *Master-Slave*, sendo o agente servidor o mestre e o agente de busca, o escravo. Neste diagrama, diferentemente do diagrama da primeira solução, o agente de busca se utiliza do padrão *Star-Shaped* para realização da tarefa de busca. Este diagrama de classes foi construído com auxílio dos diagramas independentes de cada padrão, apresentados na documentação.

A seguir temos a apresentação de uma possível seqüência de execução para este sistema, onde o funcionamento da primeira solução poderá ser melhor entendido.

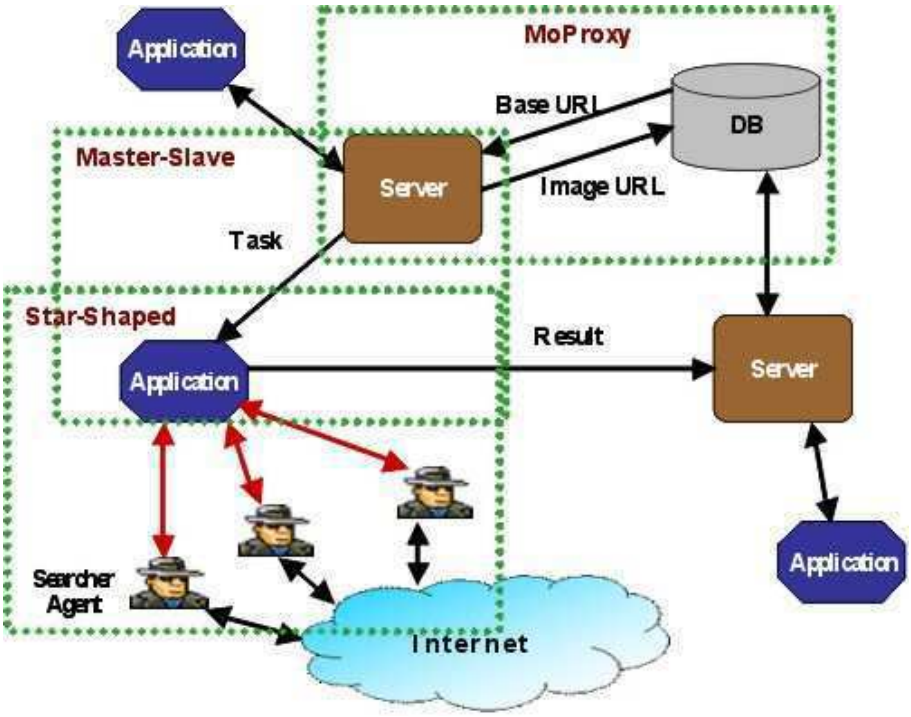


Figura 5.11: Identificação dos padrões da segunda solução para um cenário do sistema

Seqüência de execução para a segunda solução

A inicialização do servidor é semelhante à apresentada para a primeira solução (Figura 5.5). Após esta inicialização, a aplicação poderá solicitar URLs base para o servidor. No diagrama da Figura 5.13, a aplicação (*Application*), que está na agência da aplicação, envia uma mensagem para o agente servidor, indicando a quantidade de URLs base que deseja. O agente servidor, então, recupera as URLs no banco de dados através do *MoProxy* e cria um agente de busca, passando as URLs recuperadas (método *createSlave(urls)*). É nesta criação que está a adaptação do padrão *Master-Slave*. Na definição original do padrão, o agente escravo (no caso o de busca) seria criado na agência do mestre (agência do servidor) e então migraria para a agência de destino. No entanto, nesta solução, o agente de busca é criado diretamente na agência da aplicação. Assim, ao invés do agente de busca ter que migrar para a agência da aplicação, apenas uma mensagem é enviada, contendo as URLs a serem pesquisadas.

Após ser criado, o agente de busca inicia a migração pelas agências remotas (Figuras 5.14 e 5.15). Como definido pelo padrão *Star-Shaped*, após visitar cada agência e realizar a tarefa, o agente retorna para a agência da aplicação, onde entrega os resultados para a aplicação (método *addResults(results)*). Então, ele os apaga (método *removeResults()*), não

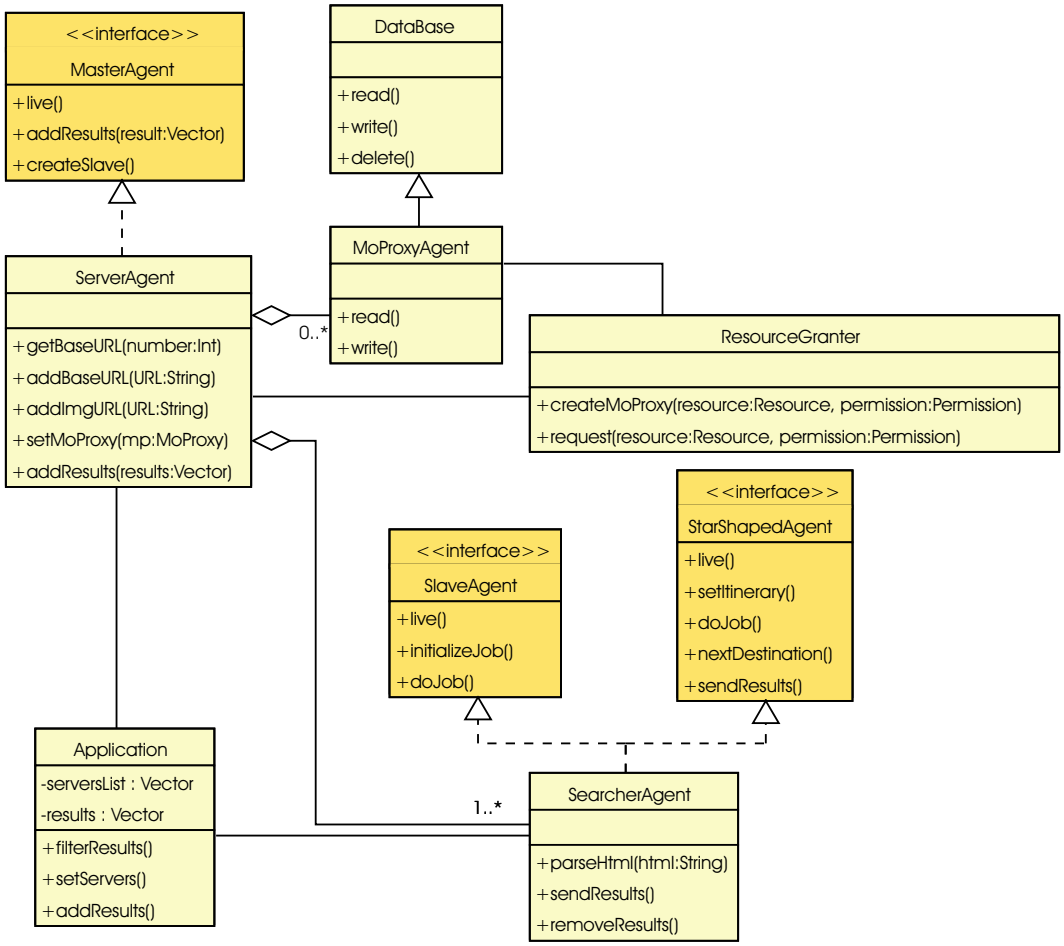


Figura 5.12: Segunda solução: diagrama de classes para o sistema

precisando migrar com os resultados já entregues para a aplicação, e migra para a próxima agência remota de seu itinerário. Após cumprir seu itinerário, o agente de busca pede para a aplicação filtrar os resultados (método *filterResults()*).

Tendo os resultados filtrados, o agente de busca os envia para o agente servidor através de uma mensagem. O agente servidor, por sua vez, armazena as URLs de imagens e as novas URLs base recuperadas na busca, no banco de dados através do *MoProxy* (Figura 5.16).

5.3 Formalização das Soluções

Nesta seção, as duas soluções propostas anteriormente para o estudo de caso serão formalizadas. Como os projetos deste estudo de caso se baseiam no uso de padrões, a formalização dos mesmos ficou bastante facilitada, uma vez que os padrões já foram formalizados individ-

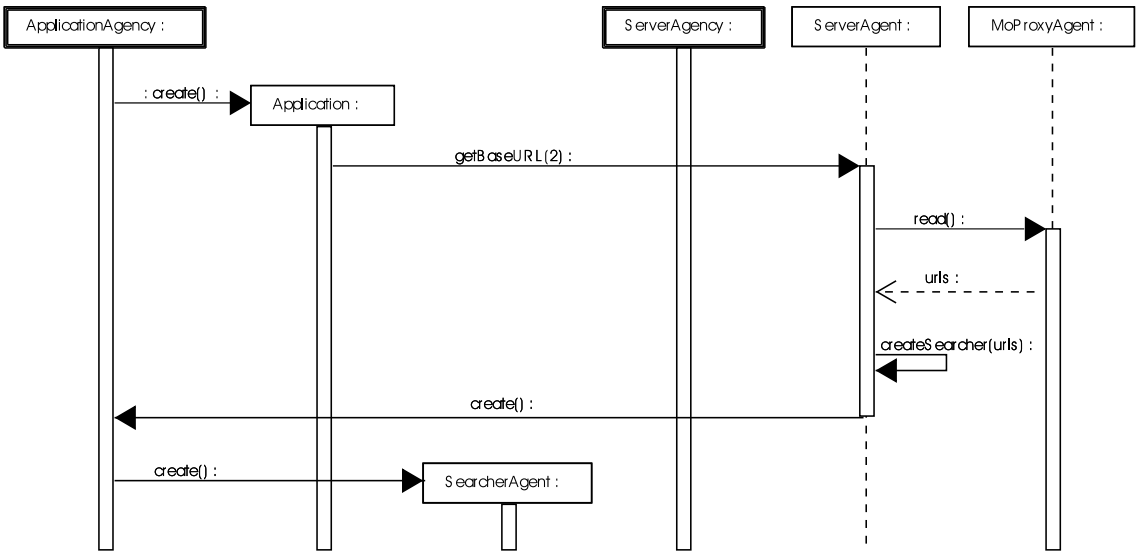


Figura 5.13: Segunda solução: diagrama de seqüências para a requisição de URLs base e criação do agente de busca

ualmente. Assim, foi necessário, apenas, acrescentar na modelagem formal, a modelagem do projeto não definida por padrões, bem como os detalhes específicos do estudo de caso no que diz respeito a aplicação dos padrões, uma vez que os mesmos haviam sido formalizados de maneira abstrata. A estrutura dos modelos dos padrões é mantida, como poderá ser percebido.

5.3.1 Primeira Solução

Na Figura 5.17, a visão de mais alto nível para a primeira solução é apresentada. Nesta visão, os 5 elementos principais do sistema podem ser percebidos: a agência servidor, detalhada pela transição de substituição *ServerAgency*; a agência da aplicação, detalhada pela transição de substituição *ApplicationAgency*; a agência remota, detalhada pela transição de substituição *RemoteAgency*; e as redes entre o servidor e a aplicação (transição de substituição *ASNet*), e entre a aplicação e a agência remota (transição de substituição *ARNet*). Perceba que esta rede entre o servidor e a aplicação é utilizada para a troca de mensagens, enquanto a outra rede é utilizada para migração dos agentes.

A seguir, os modelos que detalha cada uma destas transições de substituição são apresentados.

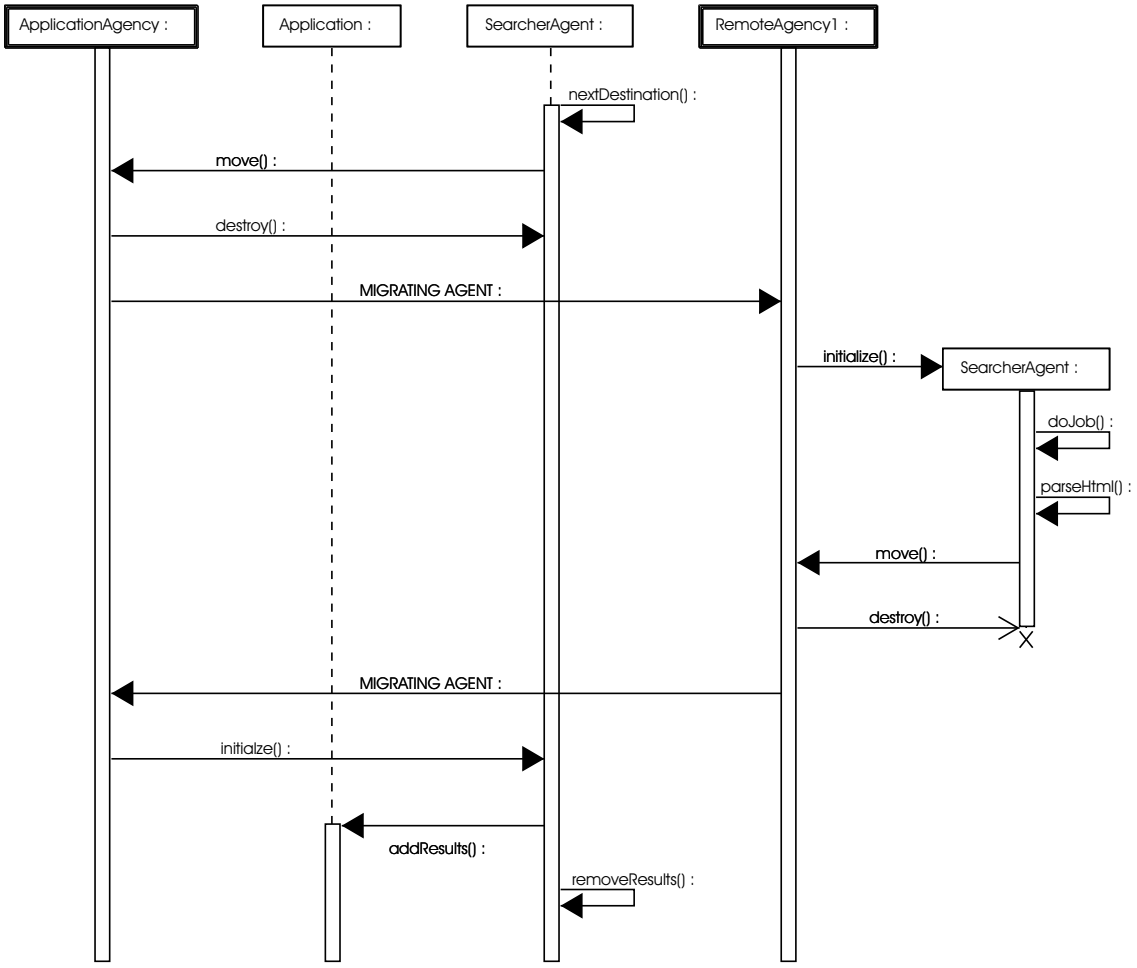


Figura 5.14: Segunda solução: primeira parte do diagrama de seqüências para recuperação de URLs de imagens

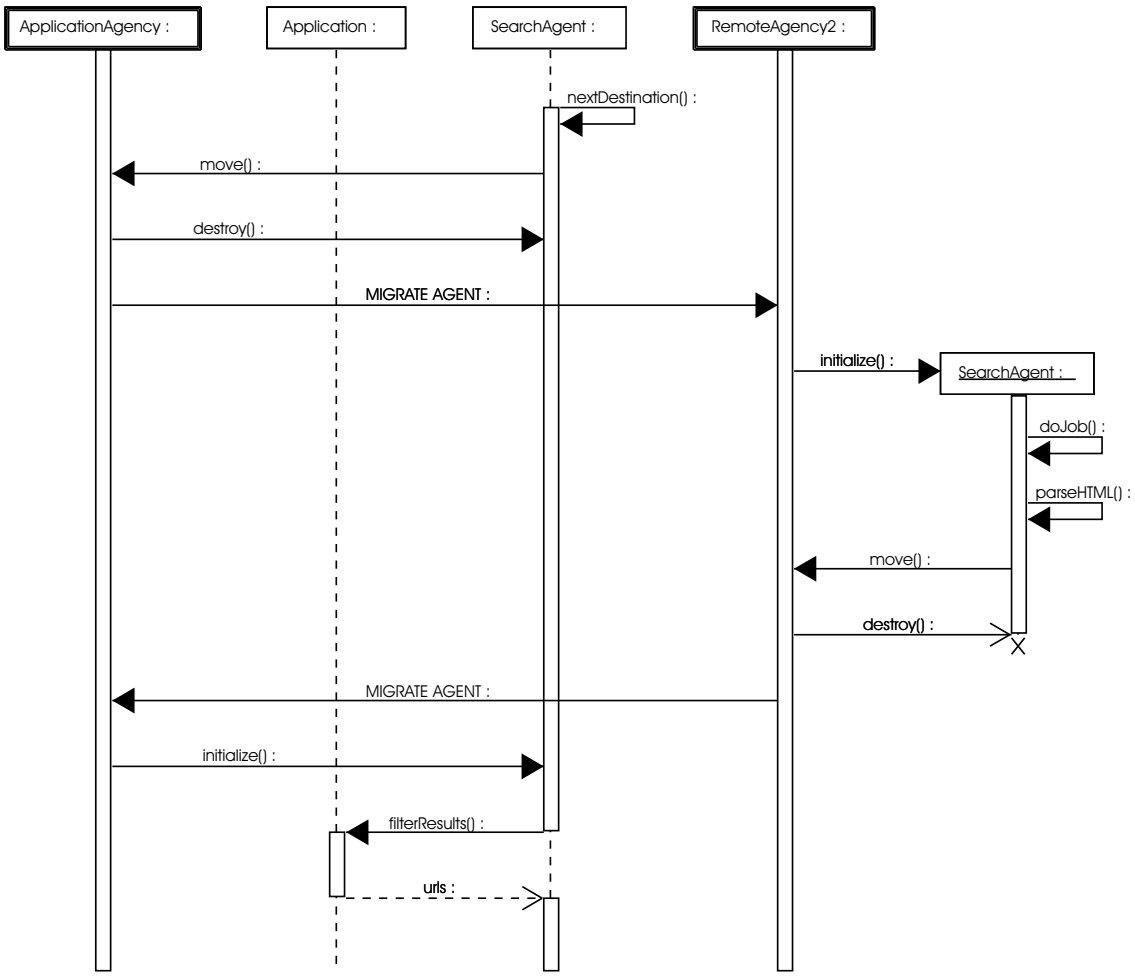


Figura 5.15: Segunda solução: segunda parte do diagrama de seqüências para recuperação de URLs de imagens

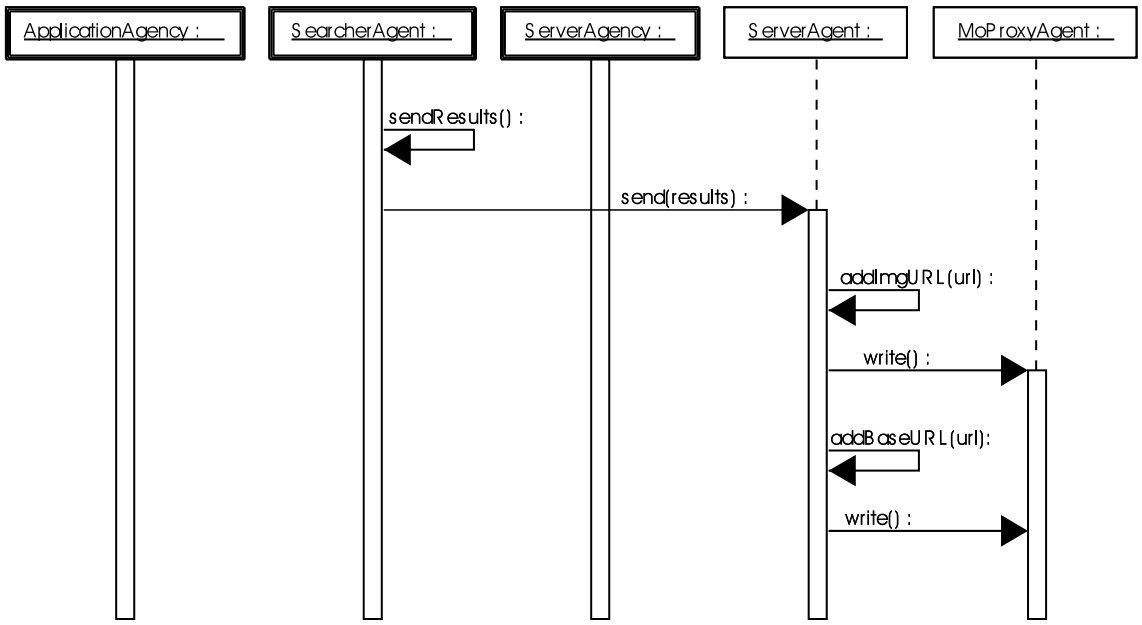


Figura 5.16: Segunda solução: diagrama de seqüências para o armazenamento das URLs de imagens recuperadas

Agência Servidor

A agência servidor (Figura 5.18) é detalhada pelas transições de substituição *ServerMsgControl*, responsável pelo envio e recebimento de mensagens no servidor, e *BDAccess*, responsável pela pesquisa e armazenamento de URLs no banco de dados.

O modelo da Figura 5.19 (detalhamento da transição de substituição *BDAccess*) é composto, basicamente pelo padrão *MoProxy*, um pouco mais detalhado para funcionar de acordo com a solução proposta. A modelagem da agência servidor ficou facilitada, pois, como pode-se perceber, esta é bastante semelhante ao modelo apresentado para o padrão *MoProxy* (Apêndice B), acrescentando-se apenas os detalhes específicos da aplicação. Neste modelo, estão presentes 4 entidades principais: agente servidor (*ServerAgent*), concessor de recursos (*ResourceGranter*), *MoProxy* e banco de dados (*DataBase*). O agente servidor está inicialmente parado (lugar *ServerAgent Stopped*, e solicita ao concessor de recursos, a criação de um *MoProxy* para acesso ao banco de dados (transição *CreateMoProxy*).

Após a criação do *MoProxy*, o agente servidor está pronto para processar as requisições que poderá receber da agência da aplicação. Quando possui uma requisição, o agente servidor a repassa para o *MoProxy* através da transição *ProcessRequest*. A requisição pode ser

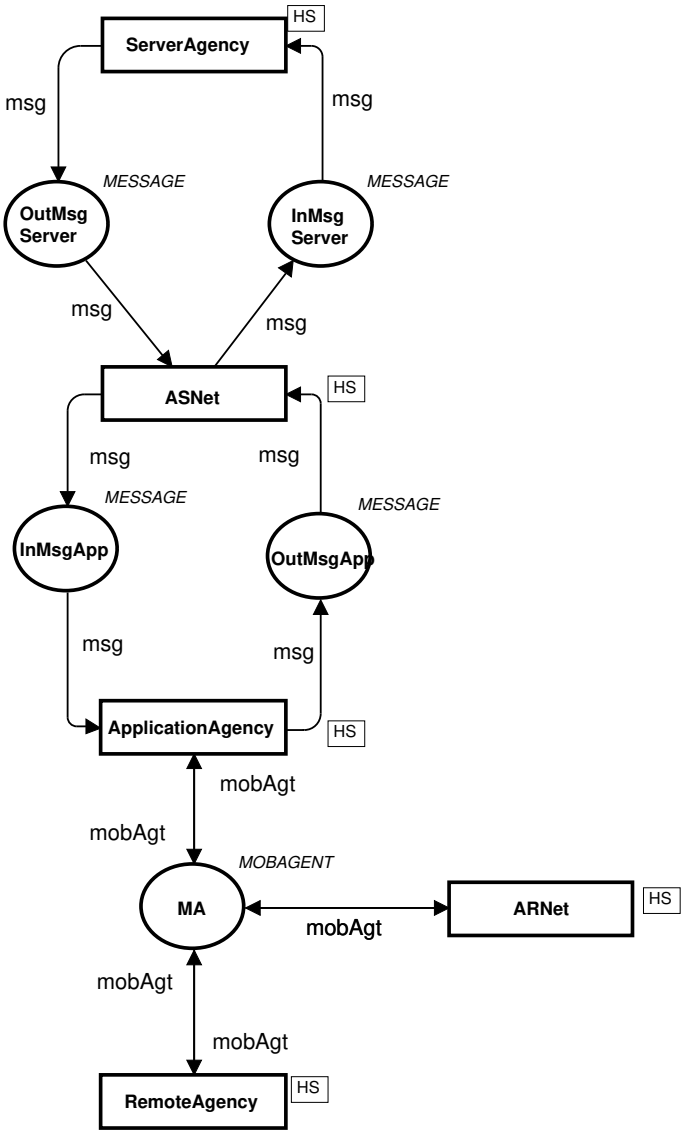


Figura 5.17: Primeira solução: modelo de alto nível

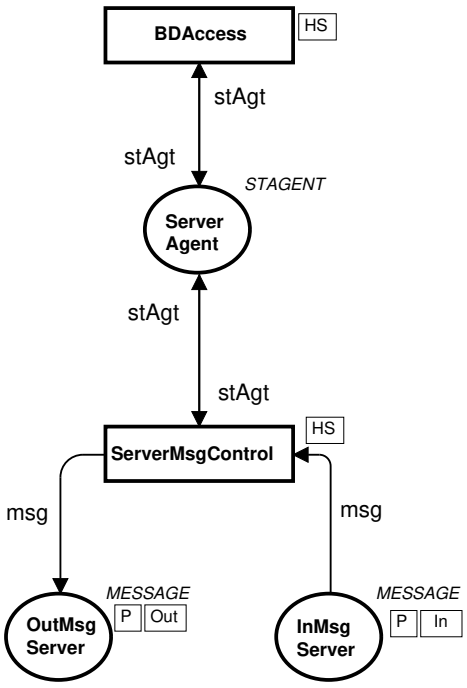


Figura 5.18: Primeira solução: modelo da agência servidor

de dois tipos: leitura e escrita. Sendo de leitura, o agente servidor fica esperando, do *MoProxy*, uma resposta com os resultados da leitura (lugar *Server_AgentRequest*). Caso contrário (sendo de escrita), o agente servidor continua esperando por novas requisições. Para receber uma resposta do *MoProxy*, a transição *GetResponse*.

No banco de dados estão armazenadas as URLs base e as URLs de imagens recuperadas. Para acessar este banco de dados, o *MoProxy* utiliza uma das duas transições disponíveis, dependendo do tipo de requisição recebida do agente servidor (transições *Read* e *Write*).

Na Figura 5.20 (detalhamento da transição de substituição *ServerMsgControl*), é modelado o envio e recebimento de mensagens pelo agente servidor. Desta forma, um agente servidor (lugar *ServerAgent*) pode enviar ou receber mensagens pelas transições *SendMessage* e *ReceiveMessage*, respectivamente.

Quando uma mensagem é enviada, ela é colocada no lugar *OutMsg* e, ao mesmo tempo, a mesma mensagem é colocada no lugar *Buffer* e uma ficha é colocada no lugar *Timeout*. Isto ocorre para que a mensagem possa ser re-enviada caso ocorra algum erro em sua transmissão. Assim, se o tempo de espera por uma confirmação do recebimento da mensagem for ultrapassado, a mensagem é re-enviada (*ReSend*). Esta confirmação deve ser colocada no lugar de fusão *Ack*, pela agência que receber a mensagem. O tempo de espera pela confirmação

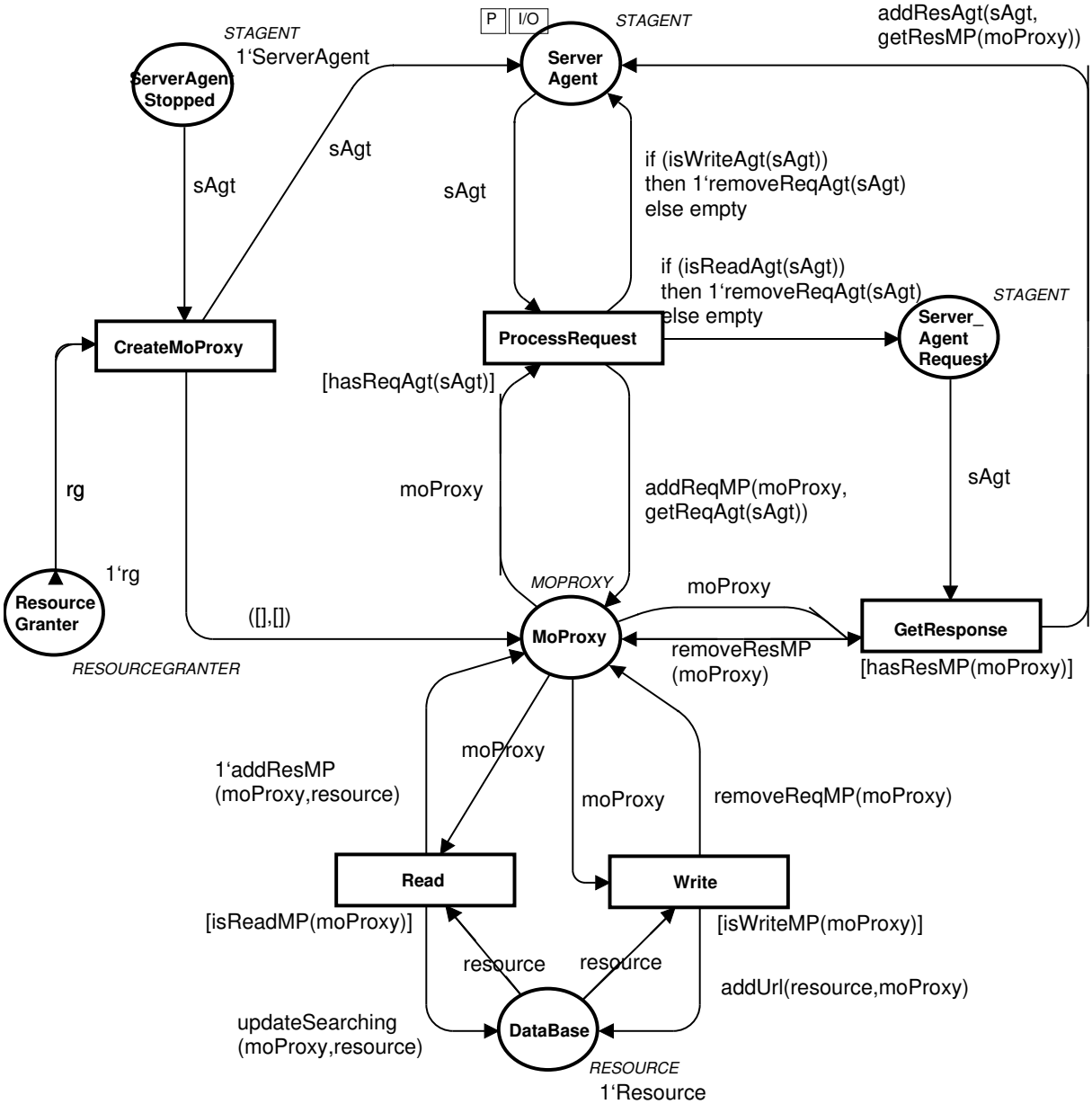


Figura 5.19: Primeira solução: modelo do acesso ao banco de dados

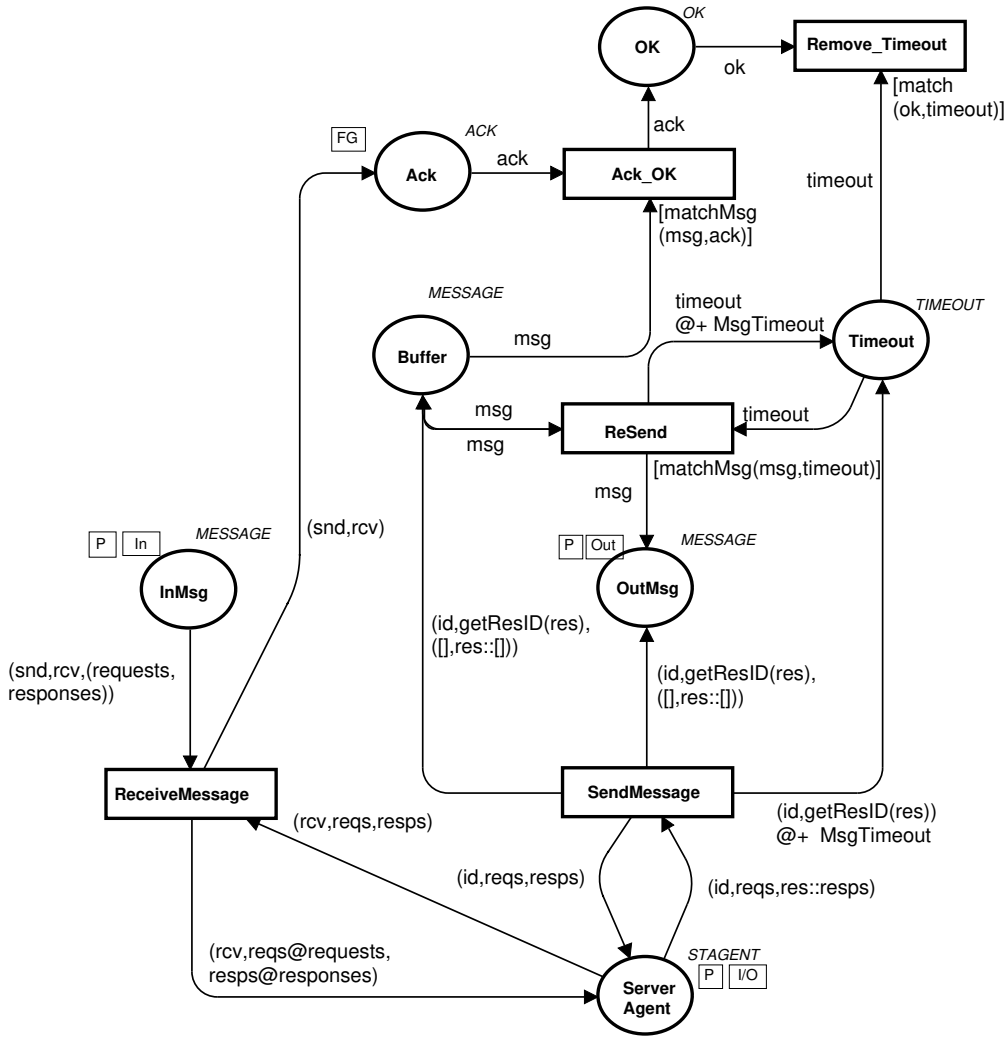


Figura 5.20: Primeira solução: modelo do controle de mensagens da agência servidor

é definido pelo valor da constante *MsgTimeout*, adicionado ao tempo da ficha timeout, o que leva essa ficha a ficar indisponível para o disparo até que o tempo global seja igual ao tempo da ficha. A confirmação do recebimento da mensagem é consumada pela transição *Ack_OK*, que consome a mensagem que estava no lugar *Buffer*, evitando que esta seja re-enviada. O timeout é consumido pela transição *Remove_Timeout*. A função *matchMsg* é utilizada como guarda para garantir que a mensagem será consumida apenas com a confirmação ou *timeout* que se refere à ela. Perceba que ao receber uma mensagem, o servidor coloca uma confirmação de recebimento no lugar *Ack*, de forma que o remetente saiba que a mensagem chegou ao destino com sucesso.

Rede Agência da Aplicação/Agência Servidor

Na Figura 5.21 (detalhamento da transição de substituição *ASNet*), é modelada a transferência de mensagem da agência servidor para agência da aplicação (lugar *OutMsgServer* para *InMsgApp*) e da agência da aplicação para a agência servidor (lugar *OutMsgAppp* para *InMsgServer*). Quando um agente é transferido, o tempo global é incrementado de acordo com a constante *MsgTransfer*. No lugar *Reliability*, existe uma ficha que define o nível de confiabilidade desta rede. É este nível de confiabilidade que define as chances de uma mensagem ser transmitida com sucesso. Assim, um erro de transmissão é possível, o que causaria a perda da mensagem. Para definir se a transmissão terá sucesso ou não, a função *netFail(netfail,reliability)* recebe dois valores como entrada e retorna verdadeiro se o primeiro valor for maior do que o segundo. O primeiro valor é um número randômico entre 0 e 100, e o segundo é a confiabilidade da rede (de 0 a 100%). Desta forma, se o valor da ficha no lugar *Reliability* for 90, a mensagem terá 90% de chance de ser transmitida com sucesso.

Agência da Aplicação

A agência da aplicação (Figura 5.22) é detalhada por 3 transições de substituição: *AppMsgControl*, responsável pelo controle de mensagens; *CloneControl*, responsável pela criação dos agentes de busca e tratamento dos resultados finais; e *AppAgentControl*, responsável pelo controle da migração dos agentes.

Na Figura 5.23 (detalhamento da transição de substituição *AppMsgControl*), é modelado

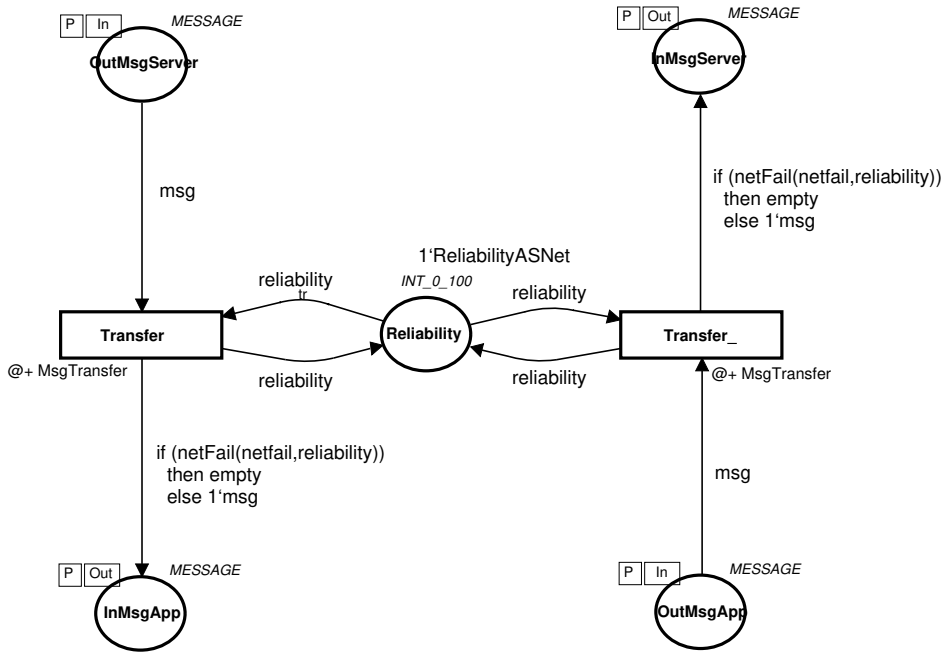


Figura 5.21: Primeira solução: modelo da rede entre a agência da aplicação e a agência servidor

o recebimento e envio de mensagens pelo agente da aplicação. Este processo é semelhante ao controle de mensagens do servidor, com envio e recebimento das mensagens, e re-envio de mensagens em caso de erro na transmissão.

O modelo da criação e controle dos clones é mostrado na Figura 5.24. Inicialmente, o agente da aplicação (lugar *ApplicationAgent*) cria um agente de busca, passando a lista de agências que deverá visitar (transição *CreateSearcher*), e vai para o lugar *AppAgt Waiting*, onde espera pelo retorno dos resultados, conforme definido pelo padrão *Master-Slave*. O agente de busca criado é colocado no lugar *SearcherAgent*, e se clona de acordo com o número de agências de destino que recebeu (transição *Clone*), como definido pelo padrão *Branching*. A criação do agente de busca, bem como a clonagem, incrementa o tempo global de acordo com a constante *CreationTime*.

Ao retornar para a agência da aplicação, cada agente envia seus resultados para o agente da aplicação e se destrói (transição *ReceiveResult*). Ao receber todos os resultados, o agente da aplicação finaliza o trabalho de busca (transição *FinishJob*) e filtra os resultados recebidos (função *filterResults(appAgent)*). O lugar de fusão *Agents Searching*, indica quantos agente estão realizando a busca. Já o lugar *Agent_IDs* é utilizado para definição da identificação dos

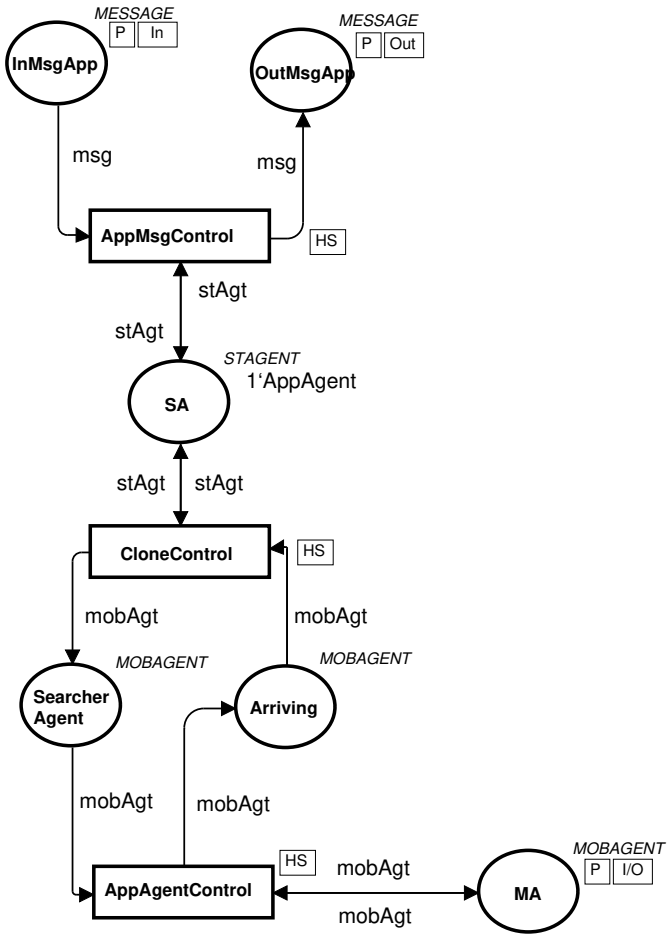


Figura 5.22: Primeira solução: modelo da agência da aplicação

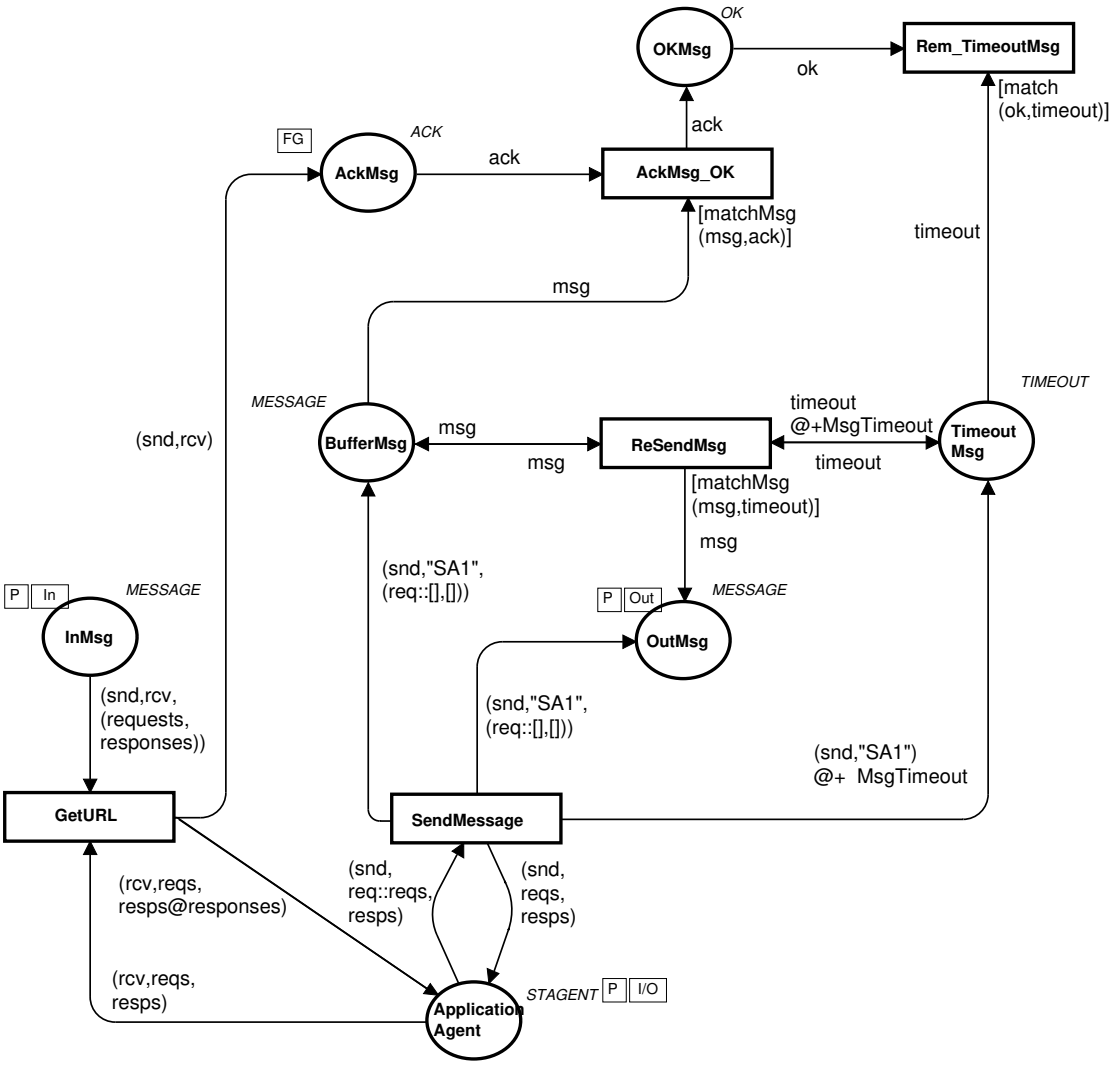


Figura 5.23: Primeira solução: modelo do controle de mensagens da agência da aplicação

agentes clones.

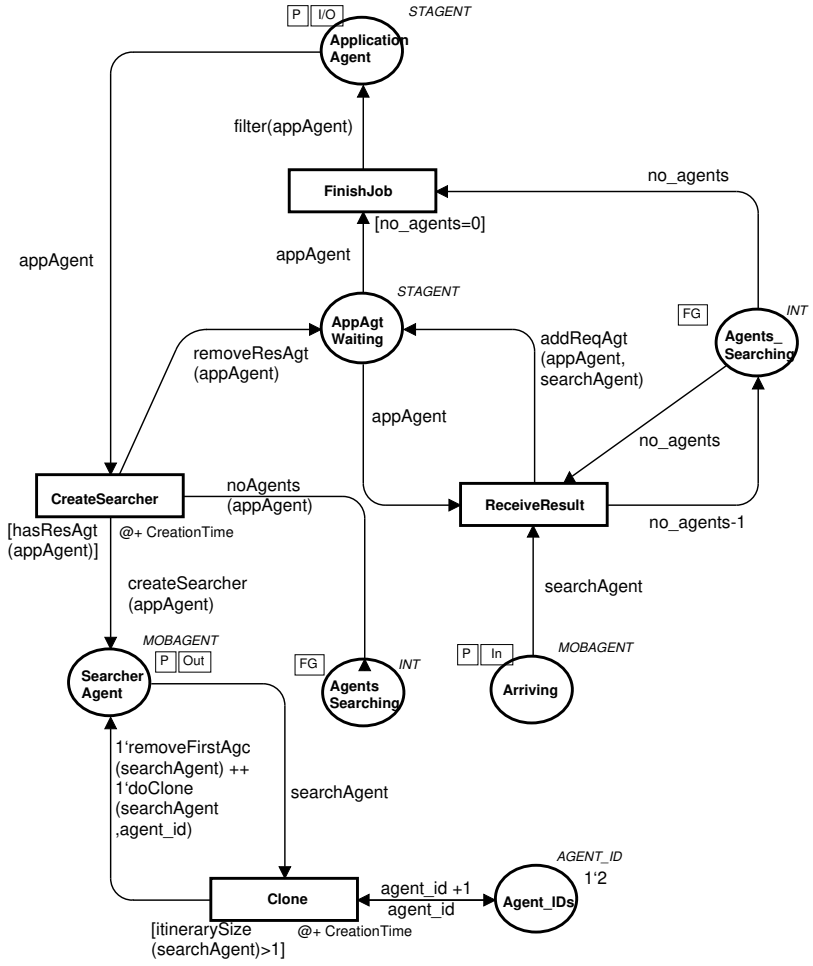


Figura 5.24: Primeira solução: modelo da criação e controle dos clones

O controle da migração dos agentes de busca é detalhado pelo modelo da Figura 5.25. Neste modelo temos a migração e o recebimento dos agentes de busca pelas transições *Migrate* e *ReceiveAgent*, respectivamente. Os outros detalhes deste modelo se referem ao reenvio dos agentes em caso de erro de transmissão, funcionando de forma semelhante ao modelo do controle das mensagens.

Rede Agência da Aplicação/Agência Remota

No modelo da rede entre a agência da aplicação e agência remota (Figura 5.26), é representado o processo de migração dos agentes. Inicialmente, o agente é colocado no lugar *Transfering* pela transição *InitMigration*. Quando esta transição dispara, o agente passa a contar

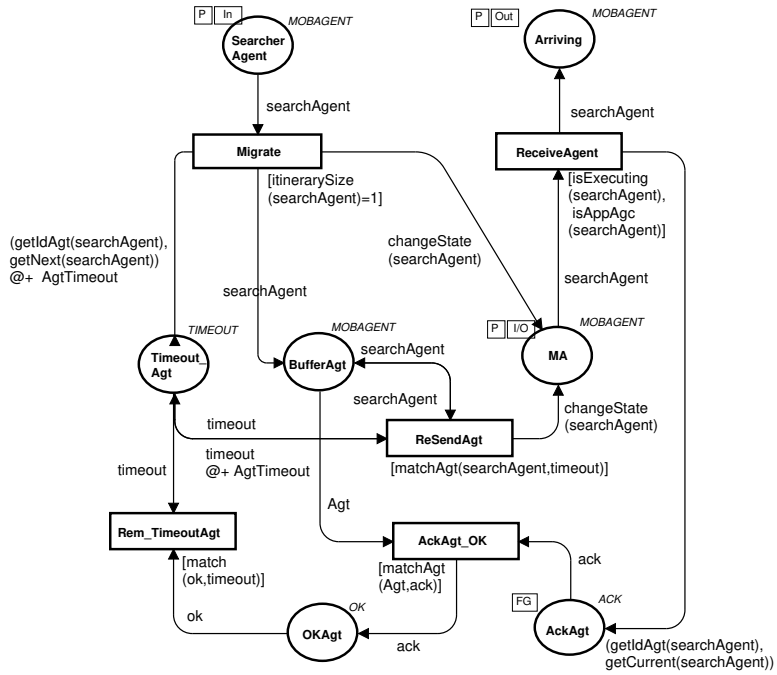


Figura 5.25: Primeira solução: modelo do controle da migração dos agentes da agência da aplicação

no nível de tráfego da rede. Este nível é representado pelo lugar *NetLoad*, que indica quantos agentes estão migrando no momento. A migração é completada pela transição *Transfer*. Nesta transição é adicionado um tempo à ficha do agente, de acordo com o tamanho do agente, o tráfego e a capacidade da rede (função *migrateTime(searchAgent,netload)*). Assim como na rede de troca de mensagens, esta rede também possui um nível de confiabilidade (lugar *Reliability*), que indica as chances de ocorre um erro na transmissão de um agente.

Agência Remota

No modelo da agência remota (Figura 5.27), temos a representação da realização da tarefa pelo agente de busca. Novamente, pode-se perceber que este modelo é semelhante ao modelo já apresentado para o padrão *Branching* isoladamente. Inicialmente, o agente é recebido pela transição *Arrive*. Então ele executa a tarefa de busca, e recolhe URLs do lugar *URLS* (transição *DoSearch*). Esta transição incrementa o tempo global de acordo com a constante *SearchTime*. Após concluir a tarefa, o agente pode migrar para outra agência (transição *Migrate*). A parte superior do modelo se refere ao re-envio do agente, no caso de ocorrer

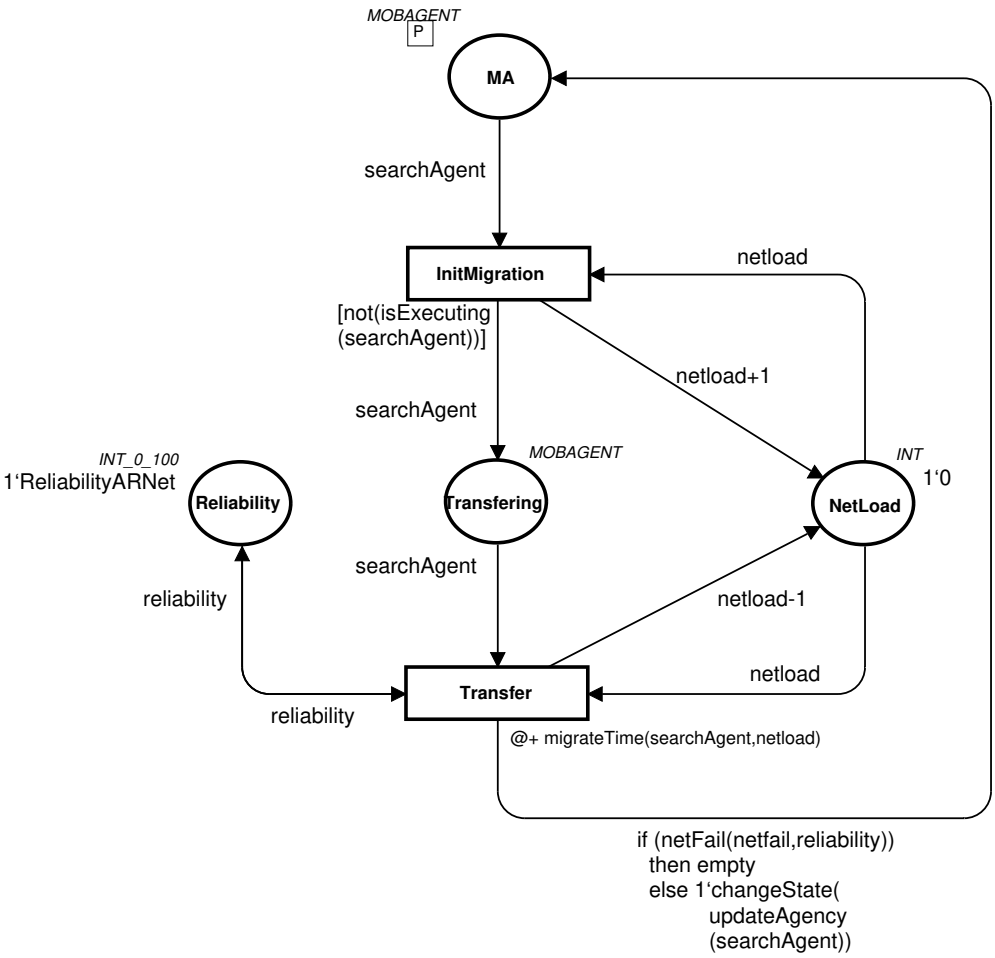


Figura 5.26: Primeira solução: modelo da rede entre a agência da aplicação e as agências remotas

algum erro na transmissão.

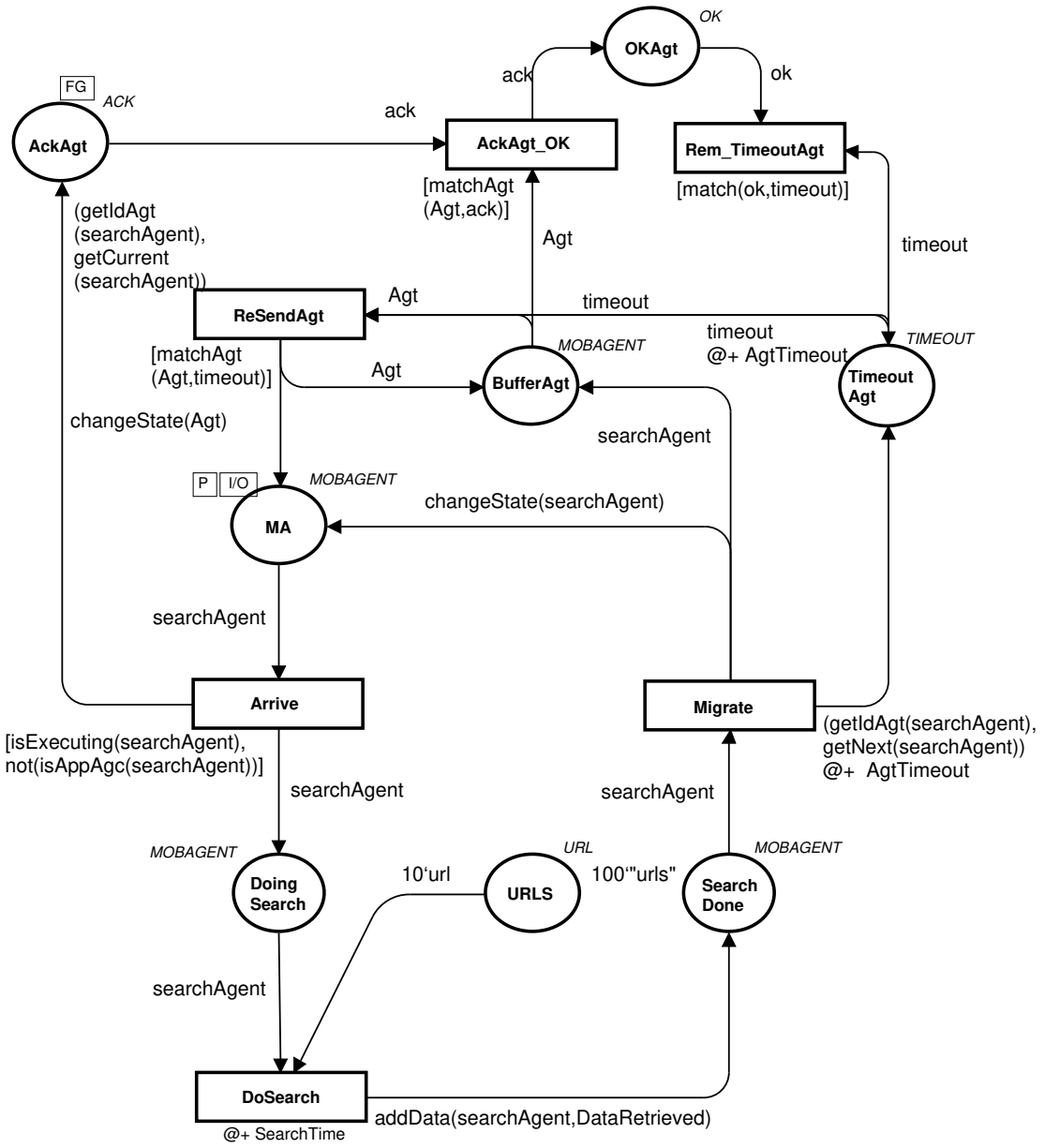


Figura 5.27: Primeira solução: modelo da agência remota

5.3.2 Segunda Solução

A visão de mais alto nível para a segunda solução é igual à apresentada para a primeira solução (Figura 5.17), contendo os 5 elementos principais do sistema: a agência servidor,

a agência da aplicação, a agência remota e as redes entre o servidor e a aplicação e entre a aplicação e a agência remota.

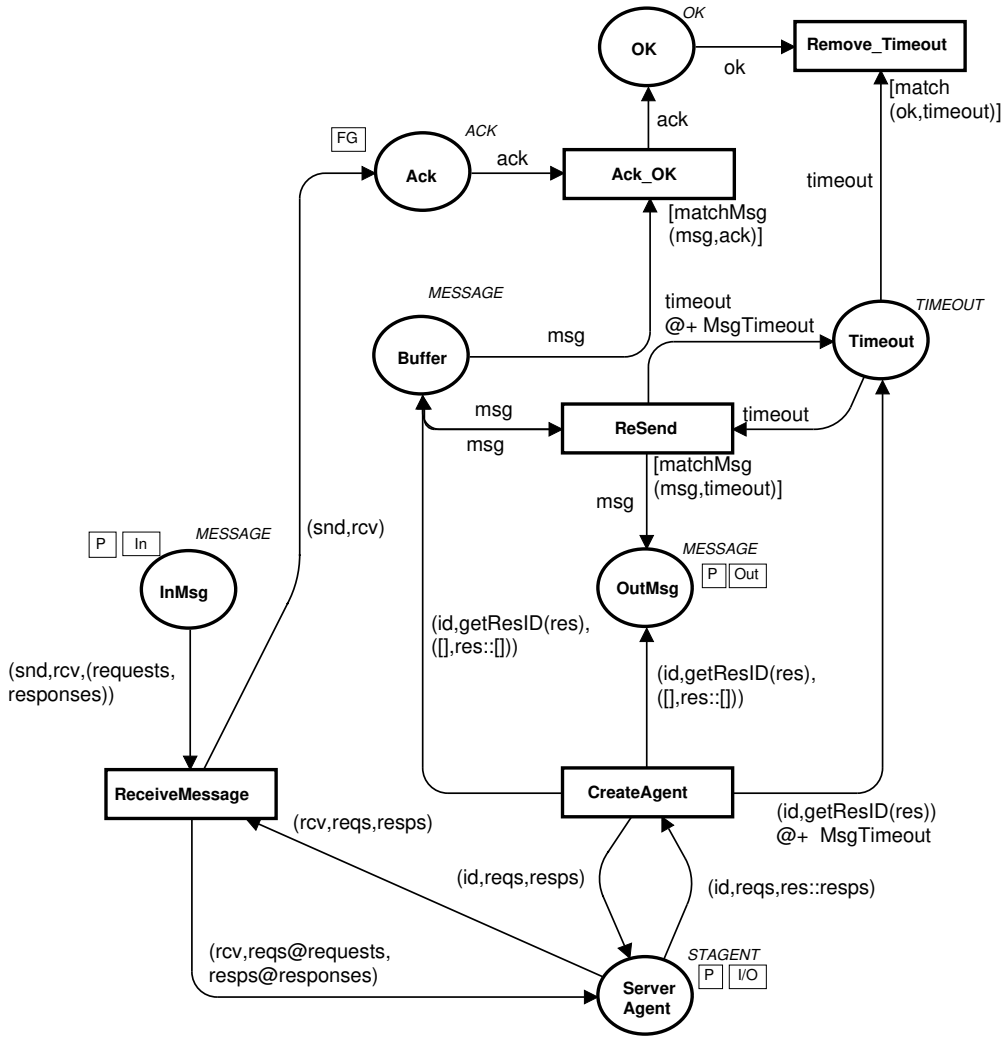


Figura 5.28: Segunda solução: modelo do controle de mensagens da agência servidor

Os modelos que detalham as transições de substituição *ServerAgency*, *ASNet*, *ARNet* e *RemoteAgency* são semelhantes aos modelos apresentados para a primeira solução. No modelo da agência servidor, o detalhamento da transição de substituição *BDAccess* também é igual ao apresentado para a primeira solução. Já o modelo que detalha a transição de substituição *ServerMsgControl* (Figura 5.28) é similar ao da primeira solução. A diferença é que o agente servidor envia uma envia uma mensagem de criação remota do agente, contendo as agências a serem visitadas (transição *CreateAgent*) e fica esperando os resultados

(como definido no padrão *Master-Slave*). A seguir, os modelos que detalham as transições de substituição da agência da aplicação são apresentados.

Agência da Aplicação

A agência da aplicação (Figura 5.29) é detalhada por 2 transições de substituição: *AppMsgControl*, responsável pelo controle de mensagens; e *AppAgentControl*, responsável pelo controle da migração dos agentes.

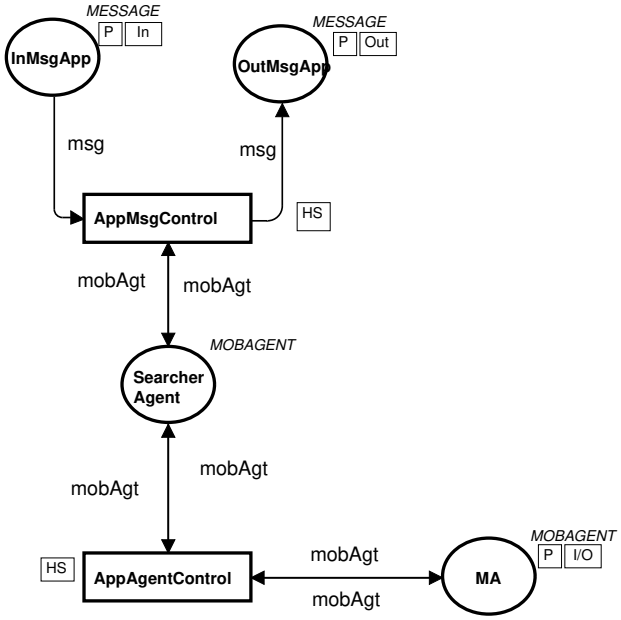


Figura 5.29: Segunda solução: modelo de alto nível da agência da aplicação

No modelo que detalha a transição de substituição *AppMsgControl* (Figura 5.30), a aplicação (lugar *Application*) envia uma mensagem para a agência servidor solicitando URLs para serem pesquisadas. Como resposta, um agente de busca é criado remotamente pela agência servidor na agência aplicação (transição *CreateSearcher*). Ao disparar, esta transição incrementa o tempo global de acordo com o valor da constante *CreationTime*. Quando termina a busca, o agente envia uma mensagem com os resultados para o servidor através da transição *SendResults* e se destrói.

No detalhamento da transição de substituição *AppAgentControl* (Figura 5.31), é represen-

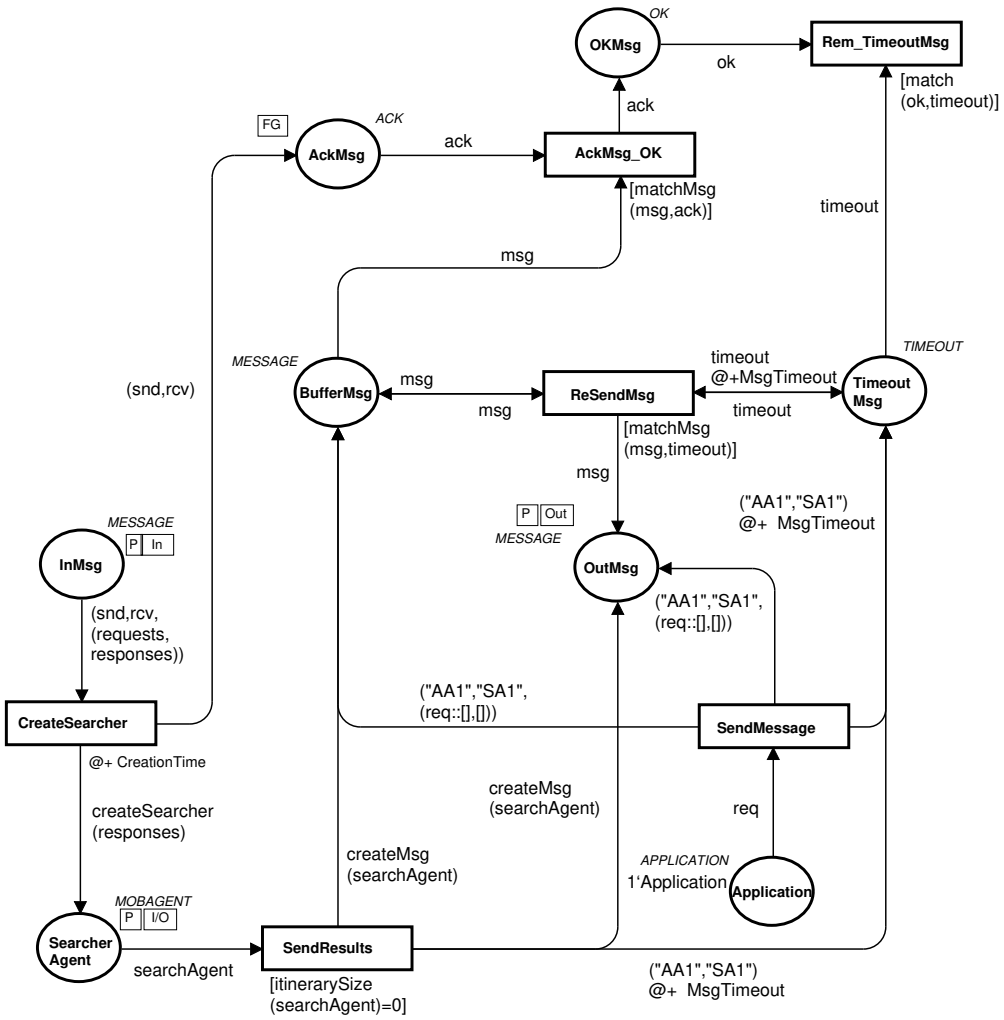


Figura 5.30: Segunda solução: modelo do controle de mensagens da agência da aplicação

tado o processo de migração e recebimento do agente na agência da aplicação. A migração do agente é representada pela transição *Migrate*. Já o recebimento do agente de busca pode ser realizado por duas transições. A transição *ReceiveData* é usada quando o agente volta para a agência da aplicação apenas para deixar os resultados da busca no lugar *Data*, e ainda irá migrar para outras agências (como definido pelo padrão *Star-Shaped*). A transição *ReceiveAgent* é utilizada quando o agente já migrou por todas as agências. Após terminar a busca, o agente filtra os resultados coletados e está pronto para enviar os resultados para a agência servidor.

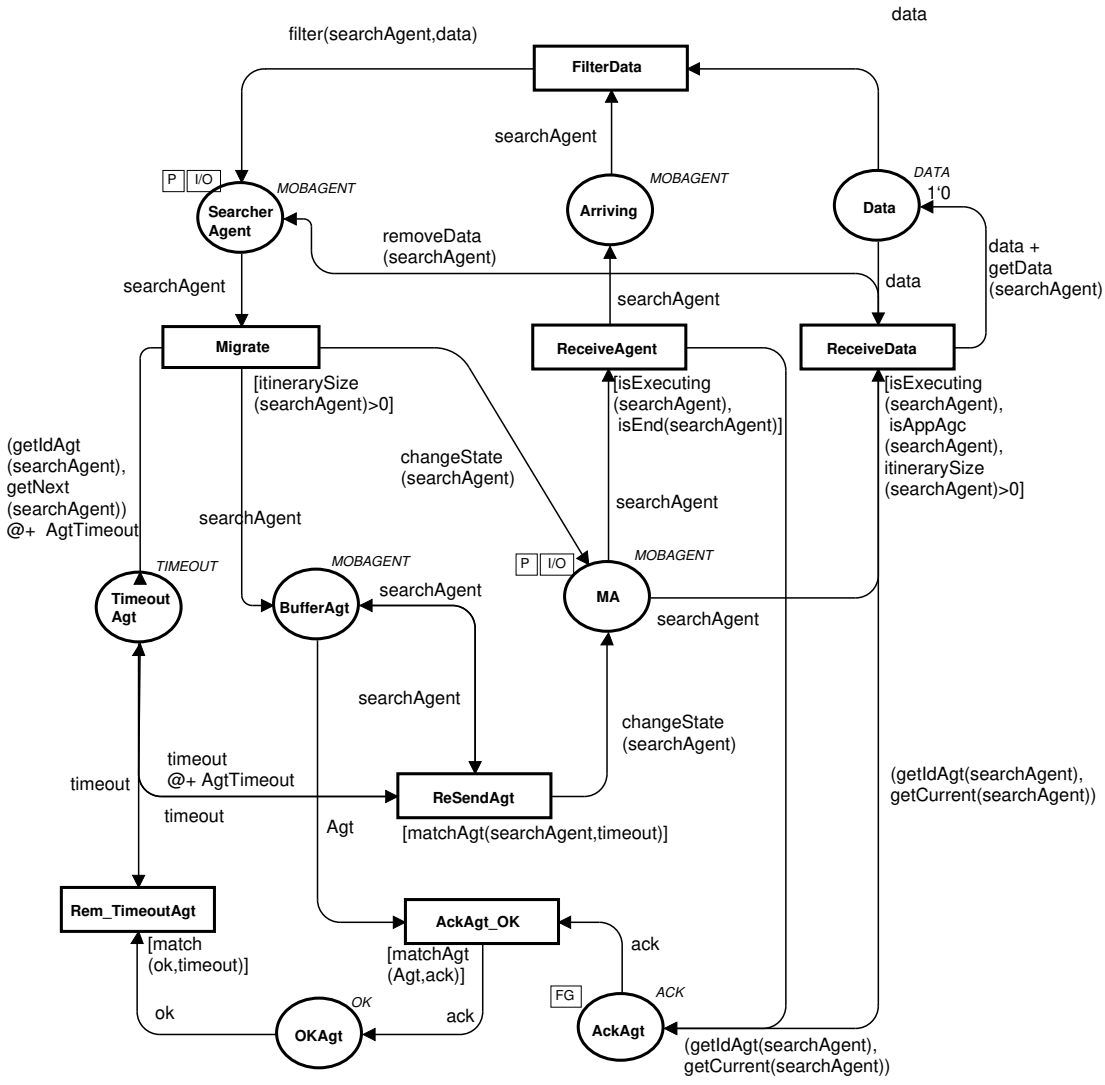


Figura 5.31: Segunda solução: modelo da de migração e recebimento do agente na agência da aplicação

Solução	*Primeira*	*Segunda*
Num. de Agências	**2**	
Nós	107366	136668
Arcos	1779164	2060894
Num. de Agências	**4**	
Nós	113160	132646
Arcos	1767028	2025821
Num. de Agências	**6**	
Nós	102723	121361
Arcos	1539662	1855096

Tabela 5.1: Resumo do grafo de ocorrência para o estudo de caso

5.3.3 Validação dos Modelos

Assim como na validação individual dos padrões, para validar estes modelos das soluções para o estudo de caso, foram utilizadas técnicas formais e informais. Através das simulações, foi possível compreender melhor o comportamento dos modelos e alguns problemas da modelagem puderam ser detectados e corrigidos. Nestas simulações, tanto interativas quanto automáticas, vários cenários para o sistema foram utilizados, incluindo variação do número de agências e da confiabilidade das redes. Desta forma, uma maior confiança na corretude do modelo foi alcançada.

Contudo, apesar de eficientes, as simulações não são suficientes para validação de propriedades do modelo que possam garantir o seu correto funcionamento. Para tanto, utilizamos a técnica de análise formal do grafo de ocorrência.

O grafo de ocorrência foi gerado para alguns cenários e na Tabela 5.3.3, é mostrada quantidade de nós para cada solução considerando 2, 4 e 6 agências de busca. É importante destacar que estes grafos foram gerados de forma parcial, uma vez que o espaço de estados dos modelos é infinito. Isto ocorre porque o envio de um agente ou mensagem pode não ser completado infinitas vezes, levando a um re-envio infinito do agente ou mensagem. Desta forma, cada re-envio incrementa o tempo global do modelo, levando a uma marcação semelhante a outras já alcançadas, porém com tempo diferente.

A partir do relatório padrão gerado pela ferramenta *Design*/CPN, uma análise mais de-

talhada pôde ser realizada, verificando algumas propriedades, limites e marcações de alguns lugares do modelo. Na Figura 5.30, é apresentado um resumo do relatório padrão para a primeira solução, considerando a solicitação de duas agências para serem pesquisadas. Na parte de estatísticas, a quantidade de nós e arcos gerados pela ferramenta, bem como o *status* do grafo é apresentado.

Na parte de propriedades de limitação (*Boundedness Properties*), alguns fatos podem ser verificados. Como se pode verificar, nos lugares *DoingSearch* da agência de destino, *Searcher* e *Arriving* do modelo principal, no máximo, dois agentes estarão serão criados, executarão a tarefa de busca e retornarão para a agência da aplicação. Além disto, o lugar *URLS* da agência de destino tem no máximo 100 fichas (marcação inicial) e no mínimo 80, que é após a coleta das URLs pelos agentes de busca. Assim, é possível garantir que a criação dos agentes e a tarefa de busca está sendo executada da forma correta.

Outras propriedades podem ser verificadas pelos limites de multi-conjunto dos lugares. No lugar *AckAgt* da agência da aplicação 4 tipos de fichas podem ser armazenadas, a confirmação do agente 1 e 2 na agência da aplicação, e a confirmação destes agentes em suas agências de destino. No lugar *DoingSearch*, pode-se verificar que os dois agentes que executam neste lugar são os agentes 1 e 2 em suas respectivas agências de destino. Outra propriedade interessante está no lugar *DataBase* da agência servidor. Nele, inicialmente existem três agências disponíveis para busca. Em seguida, duas destas agências são alocadas para busca, de acordo com a solicitação da agência da aplicação. E por fim, temos as 3 agências de busca, juntamente com os resultados da busca pelos agentes.

Na última parte do relatório padrão, temos as propriedades de vivacidade do modelo. Segundo o relatório, existem várias marcações mortas, ou seja, marcações a partir das quais nenhuma transição está habilitada a disparar. Conforme foi inspecionado no modelo, estas marcações correspondem ao estado final do modelo, quando os resultados da pesquisa são armazenados no banco de dados, porém cada marcação têm um tempo diferente, de acordo com o número de erros na transmissão de agentes e mensagens.

```
 Statistics
------------------------------------------------------------------------
  Occurrence Graph
    Nodes:  107366
    Arcs:   1779164
    Secs:   900
    Status: Partial

 Boundedness Properties
------------------------------------------------------------------------
  Best Integers Bounds    Upper      Lower
  AppAgentControl'AckAgt
                          2          0
  AppAgentControl'BufferAgt
                          2          0
  DestinationAgency'DoingSearch
                          2          0
  DestinationAgency'URLS 1
                          100        80
  Main'Arriving           2          0
  Main'InMsg              1          0
  Main'OutMsg             1          0
  Main'Searcher           2          0
  Main'Server             1          0

  Best Upper Multi-set Bounds

  AppAgentControl'AckAgt
                     1`("1","AppAgc")
                   ++ 1`("1","www.ufal.br")
                   ++ 1`("2","AppAgc")
                   ++ 1`("2","www.ufcg.edu.br")

  AppAgentControl'BufferAgt
                     1`(1,100,0,"AppAgc",["www.ufal.br"],executing)
                   ++ 1`(2,100,0,"AppAgc",["www.ufcg.edu.br"],executing)

  DestinationAgency'DoingSearch
                     1`(1,100,0,"www.ufal.br",[],executing)
                   ++ 1`(2,100,0,"www.ufcg.edu.br",[],executing)

  Main'Arriving      1`(1,100,10,"AppAgc",[],executing)
                   ++ 1`(2,100,10,"AppAgc",[],executing)

  Main'InMsg         1`("AA1","SA1",([("AA1",("read",["2"]))],[]))
                   ++ 1`("AA1","SA1",([("AA1",("write",["18"]))],[]))

  Main'OutMsg        1`("SA1","AA1",([],[("AA1",(["www.ufcg.edu.br","www.ufal.br"],["SA1"]))]))

  Main'Searcher      1`(1,100,0,"AppAgc",["www.ufcg.edu.br","www.ufal.br"],executing)
                   ++ 1`(1,100,0,"AppAgc",["www.ufal.br"],executing)
                   ++ 1`(2,100,0,"AppAgc",["www.ufcg.edu.br"],executing)

  Main'Server        1`("SA1",[],[])++ 1`("SA1",[],[("AA1",(["www.ufcg.edu.br","www.ufal.br"],["SA1"]))])
                   ++ 1`("SA1",[("AA1",("read",["2"]))],[])++ 1`("SA1",[("AA1",("write",["18"]))],[])

  ServerAgency'DataBase
                     1`[(n,"www.ufcg.edu.br"),(n,"www.ufal.br"),(n,"www.ufpe.br")]
                   ++ 1`[(y,"www.ufcg.edu.br"),(y,"www.ufal.br"),(n,"www.ufpe.br")]
                   ++ 1`[(y,"18"),(y,"www.ufcg.edu.br"),(y,"www.ufal.br"),(n,"www.ufpe.br")]

 Liveness Properties
------------------------------------------------------------------------
  Dead Markings:  4226 [99914,99774,99754,99747,99736,...]
```

Figura 5.32: Relatório padrão para a primeira solução considerando duas agências de busca

5.4 Comparativo das Soluções

Para realizar o estudo comparativo das soluções propostas, foi definido um cenário em que o cliente utiliza um computador pessoal conectado à um provedor de Internet. Para este cenário, os parâmetros e valores apresentados na Tabela 5.4 foram considerados.

O tamanho do agente foi definido a partir de uma média do tamanho de agentes já implementados. A confiabilidade na transmissão de agentes é menor do que na transmissão de mensagens, uma vez que estes são bem maiores que mensagens, estando mais suscetíveis a falhas na rede. O tempo de espera pela confirmação da chegada do agente ou da mensagem (*Timeout*) é definido de acordo com o tempo médio de transmissão do agente ou mensagem. Assim, são dados alguns segundos além do tempo da transmissão para que a confirmação possa ser recebida. O tempo de criação de um agente, foi definido de acordo com o tempo médio de criação deste em um computador pessoal, desde a solicitação da criação até a instanciação do novo agente.

Parâmetro	**Valor**
Tamanho do agente	40Kb
Capacidade da rede de agentes	56Kbps
Confiabilidade da rede de agentes	85 %
Confiabilidade da rede de mensagens	95 %
Transferência da mensagem	2s
Timeout dos agentes	8s
Timeout das mensagens	4s
Criação de um agente	1s
Número de agências de busca	2, 4 e 6

Tabela 5.2: Valores dos parâmetros considerados no comparativo entre as soluções.

Então, utilizando os modelos formais e os valores mostrados, e com a ajuda da ferramenta de análise de desempenho do *Design*/CPN [LW99], o estudo comparativo foi realizado. Esta ferramenta possibilita a coleta dos dados necessários para geração dos gráficos que auxiliam a comparação, através de simulações, juntamente com três funções de coleta: *Predicate*, *Observation* e *Create*. A primeira função diz quando os dados para comparação devem ser

coletados, a segunda indica que dados devem coletados, enquanto a terceira indica onde os dados devem ser guardados. Neste estudo, o tempo gasto para realização da busca por URLs de imagens foi guardado em um arquivo de log, quando os resultados finais eram armazenados no banco de dados (transição *Write* do modelo *ServerAgency*). Os gráficos para comparação foram gerados a partir do arquivo de log. É importante destacar que este tempo coletado foi modelado com a técnicas de redes temporizadas. Desta forma, o incremento do tempo é definido pelas transições temporizadas, que podem ser percebidas nos modelos formais, como, por exemplo, na transmissão dos agentes.

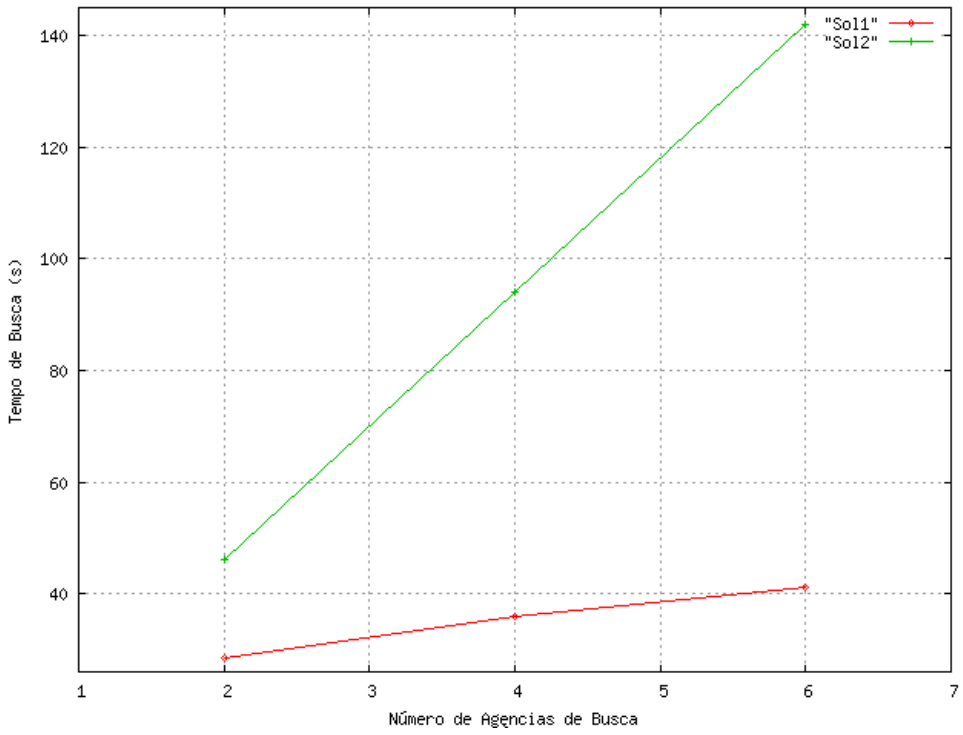


Figura 5.33: Gráfico comparativo considerando uma rede de 56Kbps

Na Figura 5.33 é apresentado um gráfico com os resultados da coleta realizada pela ferramenta *Design*/CPN.

Neste gráfico, pode-se perceber que a primeira primeira solução têm grande vantagem sobre a segunda em relação ao tempo total para realização da busca. Mesmo com apenas duas agências a serem pesquisadas, esta diferença já é grande. À medida que o número de agências de busca é aumentando, esta diferença só aumenta. Este fato pode ser verificado quando são utilizadas 4 e 6 agências de busca. Isto ocorre, porque a busca na primeira solução é realizada paralelamente por vários agentes, enquanto na segunda solução a busca

é seqüencial. Assim, a segunda solução têm o seu tempo mais afetado pelo aumento do número de agências de busca.

Apesar destes resultados, não se pode afirmar que a primeira solução é melhor que a segunda, uma vez que existem outras variáveis envolvidas. Se o processamento na agência da aplicação for lento, por exemplo, o tempo de criação de um agente pode ser mais demorado. Como na primeira solução existe a criação de vários agentes, isto pode afetar diretamente o seu desempenho. Desta forma, intuitivamente, pode-se pensar que a primeira e a segunda solução podem ter desempenhos mais próximos, considerando um computador mais lento.

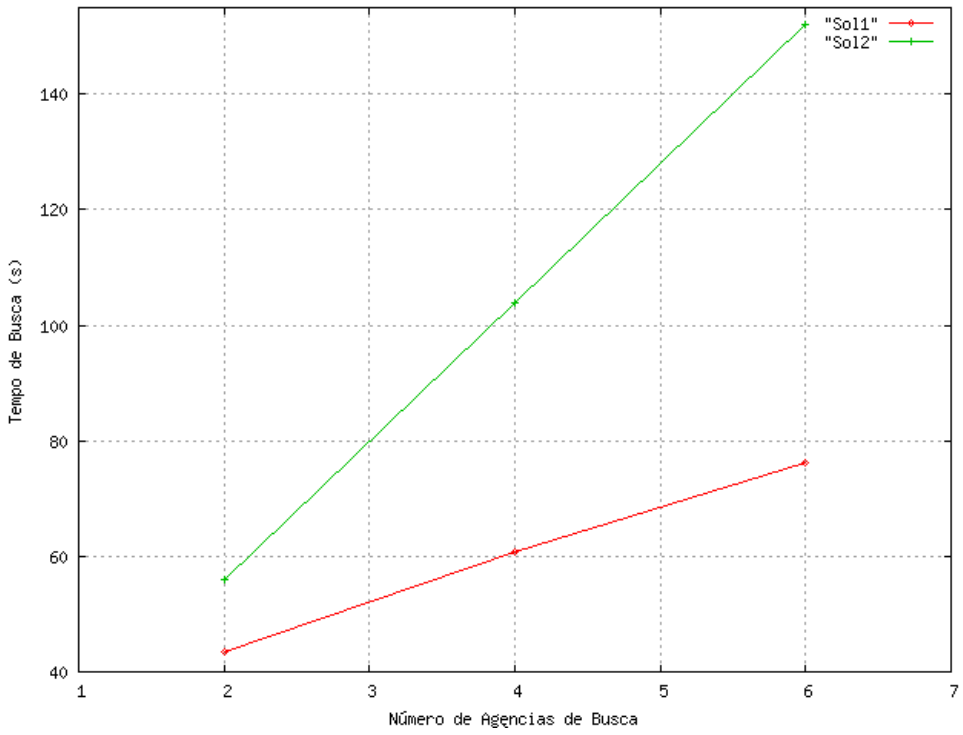


Figura 5.34: Gráfico comparativo considerando uma rede de 56Kbps e um computador lento

No entanto, isto não é confirmado pelo gráfico da figura 5.34, construído também a partir dos modelos formais. Esta gráfico mostra um comparativo para um cenário igual ao mostrado anteriormente, porém considerando um computador mais lento, em que o tempo de criação de um agente é de 5s. Como pode ser verificado, apesar de o desempenho da primeira solução piorar bastante, este ainda continua melhor do que o da segunda solução.

Se considerarmos um cenário em que a agência cliente está funcionando em um computador conectado em uma rede de alta velocidade, como em uma universidade, por exemplo, os resultados do estudo podem mudar bastante. No gráfico da Figura 5.35, é mostrado um es-

tudo considerando-se que a agência da aplicação possui uma rede com velocidade de 8Mbps. Como podemos verificar, o desempenho das duas soluções são bastante próximos, até com pequena vantagem para a segunda solução.

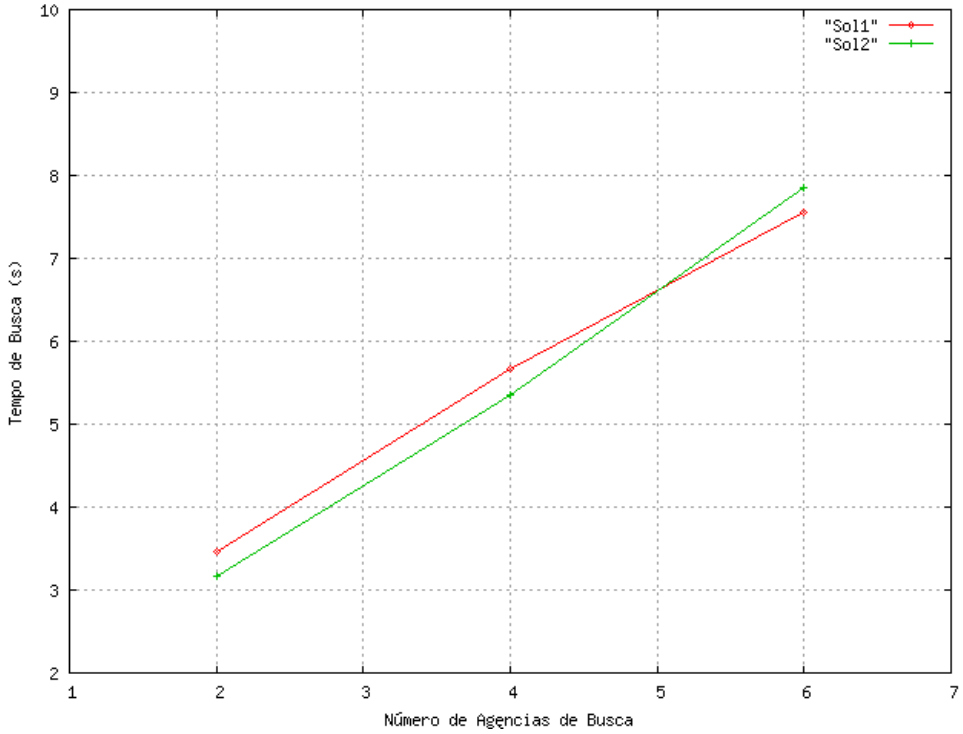


Figura 5.35: Gráfico comparativo considerando uma rede de 8Mbps

Desta forma, pode-se perceber que a escolha da solução a ser implementada depende do cenário onde o sistema funcionará. Assim, tendo as soluções formalizadas, fica bastante simples realizar análises, variando os cenários, para verificação do comportamento das soluções. Além do tempo total da busca, outros fatores podem ser considerados na escolha da melhor solução. Se for considerada, por exemplo, a utilização da agência da aplicação, a segunda solução pode levar vantagem, uma vez que terá apenas um agente executando, que só estará nesta agência para armazenamento dos resultados. Já na primeira solução, sempre existirá um agente (o agente da aplicação) executando, e em alguns momentos teremos vários outros agentes também executando nesta agência (agentes de busca). Isto pode tornar a máquina onde está rodando a aplicação mais lenta, o que pode não ser interessante em alguns caso, como, por exemplo, a aplicação estar executando na máquina de um usuário comum, sem grande poder de processamento.

5.5 Conclusões

Neste capítulo, um sistema de recuperação de URL de imagens a partir de arquivos HTML foi apresentado como estudo de caso. Para este sistema, foram definidas duas soluções de projeto usando padrões de projeto para agentes móveis. Como mostrado, o catálogo e a documentação dos padrões de projeto para agentes móveis foi bastante importante na definição das soluções. Cada uma das soluções propostas foi formalizada com redes de Petri, formalização esta que foi facilitada pela existência dos modelos formais individuais dos padrões.

A partir destes modelos formais foram realizadas simulações e análises formais que garantiram um melhor entendimento sobre o funcionamento do sistema, permitindo a correções de erros e dando uma maior confiança acerca da validade do modelo. Tendo os modelos analisados, um estudo comparativo das duas soluções propostas pôde ser realizado. Isto permitiu a verificação de como as soluções se comportariam diante de alguns cenários e, assim, as mesmas puderam ser comparadas de forma a saber qual teria o melhor desempenho.

Assim, a formalização das soluções se mostrou uma ferramenta bastante interessante, uma vez que pode servir como forma de documentação, pode ser simulada, analisada, e seu comportamento pode ser verificado a partir de valores variáveis de um cenário real, mesmo antes de ser iniciada a implementação do sistema.

Capítulo 6

Considerações Finais

O tema central deste trabalho foi a catalogação e formalização de padrões de projeto para agentes móveis. A seguir, destacamos os passos para atingirmos os objetivos definidos, bem como os resultados alcançados com a finalização da dissertação. Por fim, alguns trabalhos futuros possíveis são destacados.

Como mostrado, inicialmente foi necessário realizar um levantamento dos padrões de projeto para agentes móveis que têm sido propostos. A partir destes padrões, categorias, como comunicação e migração, foram definidas para classificação dos padrões. Também foi definida uma forma de documentação para catalogação destes padrões, que incluiu elementos como, por exemplo, descrição, motivação e estrutura. Tendo a forma de classificação e documentação dos padrões, um catálogo foi construído. Este catálogo, demandou um certo trabalho, principalmente no que diz respeito ao entendimento dos padrões. Para isso, foi necessária a modelagem UML destes em duas fases: a modelagem independente de plataforma; e a modelagem dependente nas três plataformas escolhidas (Aglets, Grasshopper e JADE).

Um segundo passo foi a formalização dos padrões de projeto para agentes móveis. Para esta formalização, as redes de Petri coloridas hierárquicas temporizadas foram escolhidas por possuir características que consideramos importantes para que os modelos fossem melhor aproveitados. Em seguida, os padrões de projeto para agentes móveis foram formalizados e, tendo as formalizações, simulações foram realizadas, o que permitiu um melhor entendimento acerca do funcionamento destes padrões. Também foram realizadas análises formais para validação dos padrões e da formalização.

Por fim, foi definido um estudo de caso, para o qual foram modelados dois projetos que se basearam na aplicação de padrões de projeto para agentes móveis. Após a modelagem UML destes projetos, os mesmos foram formalizados com redes de Petri. Concluída a formalização, cada solução para o estudo de caso passou por simulações e análises, e um estudo comparativo foi realizado, usando os modelos formais.

6.1 Contribuições

Após a conclusão deste trabalho, algumas contribuições trazidas pelo mesmo podem ser destacadas. Primeiramente, a forma de documentação, bem como as categorias definidas, devem facilitar a descrição dos padrões já propostos e de padrões que venham a ser propostos, o que pode contribuir para um melhor entendimento destes. A catalogação poderá centralizar os padrões de projeto para agentes móveis, facilitando a identificação e acesso aos padrões já existentes. Desta forma, todo este trabalho, pode incentivar a aplicação destes padrões, melhorando e facilitando o processo de desenvolvimento de aplicações baseadas em agentes móveis.

Outra contribuição é em relação à formalização dos padrões de projeto para agentes móveis. Esta formalização, além de servir como documentação, uma vez que não apresenta ambigüidade, permite a realização de simulações e análises, permitindo um melhor entendimento sobre o padrão, o que também pode facilitar e incentivar o uso dos mesmos.

Como visto no estudo de caso, o uso de padrões pode simplificar bastante a definição do projeto de uma aplicação. Além disto, conhecendo os padrões aplicados, o entendimento da solução de projeto apresentada pode ser bastante facilitada. Com a formalização do projeto, que é simplificada pela existência dos modelos formais dos padrões individualmente, é possível realizar análises sobre o mesmo, buscando a identificação de erros e verificando como o sistema projetado irá se comportar diante de um dado cenário, mesmo antes de ser iniciada a implementação. Isto tudo pode contribuir para um desenvolvimento mais rápido e eficiente das aplicações baseadas em agentes móveis.

6.2 Trabalhos Futuros

Alguns trabalhos futuros podem ser destacados. É necessário, um estudo sobre os demais padrões classificados, mas que não foram detalhados, de forma a completar o catálogo dos padrões de projeto para agentes móveis. Além disto, deve ser realizado um levantamento buscando a identificação de padrões que ainda não tenham sido catalogados. Outro trabalho interessante, é a inclusão, no catálogo, de variações dos padrões, como, por exemplo, a variação do padrão *Master-Slave* apresentada do estudo de caso, ou a variação do padrão *Itinerary*, apresentada em [Gue02a]. Uma idéia interessante é disponibilizar os modelos formais *on-line*. Assim, poderia-se facilitar o acesso a estes modelos, permitindo também a simulação dos mesmos.

Outras questões também podem ser consideradas. Pode-se analisar o desempenho de uma aplicação implementada e realizar um estudo comparativo com resultados de análises da formalização, verificando-se, desta forma, o quão próximos seriam os resultados destas análises. É interessante a definição de teste para os padrões de projeto para agentes móveis. Estes testes, poderiam ser aplicados em sistemas que estejam utilizando determinado padrão, de forma a verificar se o mesmo está aplicado corretamente e se seu funcionamento está de acordo com o proposto. A especificação de propriedades dos padrões também é um trabalho importante. Assim, tendo um projeto formalizado, poderiam ser verificadas estas propriedades utilizando técnicas de verificação de modelos, o que poderia garantir se os padrões estão, ou não, sendo aplicados de forma correta.

Bibliografia

[AD98] Silva A. and J Delgado. The agent pattern for mobile agent systems. In *European Conference on Pattern Languages of Programming and Computig, EuroPLoP'98*, 1998.

[AIS77] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language - Towns Building-Construction*. Oxford University Press, New York, 1977.

[AL98] Yariv Aridor and Danny B. Lange. Agent design patterns: Elements of agent application design. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 108–115. ACM Press, May 1998.

[AMM01] Jean-Paul Arcangeli, Christine Maurel, and Frédéric Migeon. An api for high-level software engineering of distributed and mobile applications. In *Eighth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS01)*, 2001.

[BCTR03] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, and Giovanni Rimassa. *JADE PROGRAMMER'S GUIDE*. http://jade.cselt.it/docs, February 2003.

[BG03] L. B. Batista and H. M. Gomes. Um método para classificação e recuperação de imagens da world wide web utilizando mapas auto-organizáveis. In *Simpósio Brasileiro de Automação Inteligente - SBAI 03*, 2003.

[BM00] M. C. Bernardes and E. S. Moreira. Implementation of an intrusion detection system based on mobile agents. In *International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE 2000)*. IEEE, 2000.

[Car99] L. Cardelli. Abstractions for mobile computation. In *Proceedings of Secure Internet Programming*, volume 1603 of *Lecture Notes in Computer Science*, 1999.

[Che98] D. M. Chess. Security issues in mobile code systems. In *Mobile Agents and Security*, pages 1–14, 1998.

[CLZ01] G. Cabri, L. Leonardi, and F. Zambonelli. Engineering mobile-agent applications via context-dependent coordination. In *Proceedings of International Conference on Software Engineering - ICSE'2001*, 2001.

[D'A] D'Agent. http://agent.cs.dartmouth.edu/.

[DOKO99] D. Deugo, F. Oppacher, J. Kuester, and I. Von Otte. Patterns as a means for intelligent software engineering. In *IC-AI-99*, School of Computer Science, Carleton University, Ottawa, Ontario, Canada,, 1999.

[DW99] D. Deugo and M. Weiss. A case for mobile agent patterns. In *Mobile Agents in the Context of Competition and Co-operation (MAC3)*, 1999.

[FIP] FIPA. Foundation for intelligent physical agents. http://www.fipa.org/repository/componentspecs.html.

[Fok98] GMD Fokus. Mobile agent system interoperability facilities specification. Technical report, International Business Machine Corporation, 1998.

[FPV98] A. Fuggeta, G.P. Picco, and G. Vigna. Understanding code mobility. In *IEEE Transactions on Software Engineering*, volume 24 / 5, May 1998.

[GE99] A. Gustavsson and M. Ersson. Formalizing the intent of design patterns. Technical report, Uppsala Universitet, July 1999.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[GM01] Vincenzo Grassi and Raffaela Mirandola. UML modelling and performance analysis of mobile software architectures. In Martin Gogolla and Cris Kobryn, editors, *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools. 4th International Conference, Toronto, Canada, October 2001, Proceedings*, volume 2185 of *LNCS*, pages 209–224. Springer, 2001.

[Gmb01] IKV++ GmbH. *Grasshopper Basic and Concepts - Release 2.2*. http://www.grasshopper.de, march 2001.

[GMM03] F. P. Guedes, P. D. L. Machado, and V. N. Medeiros. Developing mobile agent-based applications. In *XXIX Conferencia Latino Americana de Informática - CLEI 2003*, La Paz, 2003.

[GOP02] R. H. Glitho, E. Olougouna, and S. Pierre. Mobile agents and their use for information retrieval: A brief overview and an elaborate case study. *IEEE Network*, 2002.

[Gue02a] F. P. Guedes. Um modelo para o desenvolvimento de aplicações baseadas em agentes móveis. Master's thesis, Universidade Federal de Campina Grande, 2002.

[Gue02b] Dalton Dario Serey Guerrero. Redes de petri orientadas a objeto. *Tese de Doutorado - COPELE, UFCG*, 2002.

[HCK95] C. G. Harrison, D. M. Chess, and A. Kershenbaum. Mobile agents: are they a good idea? Technical Report RC 19887, IBM Thomas J. Watson Research Center, March 1995.

[IKV] IKV++, Germany. *Grasshopper Basics and Concepts*. http://www.grasshopper.de.

[JCK96] K. Jensen, S. Christensen, and L. M. Kristensen. *Design/CPN Occurency Graph Manual*. University of Aarhus, Denmark, 3.0 edition, 1996.

[Jen92] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume 1 of *Monographs in Theoretical Computer Science*. Springer-Verlag, 1992.

[JK99] W. Jansen and T. Karygiannis. Mobile agent security. In *NIST Special Publication 800-19*, 1999.

[KKPS98] Elizabeth A. Kendall, P. V. Murali Krishna, Chirag V. Pathak, and C. B. Suresh. Patterns of intelligent and mobile agents. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 92–99, New York, 9–13, 1998. ACM Press.

[KRSW01] Cornel Klein, Andreas Rausch, Marc Sihling, and Zhaojun Wen. Extension of the Unified Modeling Language for mobile agents. In Keng Siau and Terry Halpin, editors, *Unified Modeling Language: Systems Analysis, Design and Development Issues*, chapter 8, pages 116–128. Idea Publishing Group, 2001.

[Lar99] Craig Larman. *Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design*. Prentice Hall, Inc., 1999.

[LdFG04] E. F. A. Lima, J. C. A. de Figueiredo, and D. D. S Guerrero. Using coloured petri nets to compare mobile agent design patterns. *Electronic Notes in Theoretical Computer Science*, 95:287 – 305, 2004. Selected Papers from VI Workshop on Formal Methods - WMF'2003.

[LMAFRS03] E. F. A Lima, P. D. L. M Machado, J. C. Abrantes F., and F. Ronison S. Padrões de projeto para desenvolvimento de aplicações baseadas em agentes móveis. Tutorial apresentado no Simpósio Brasileiro de Engenharia de Software, Outubro 2003.

[LMdFS04] E. F. A Lima, P. D. L. M Machado, J. C. A. de Figueiredo, and F. R. Sampaio. An approach to modelling and applying mobile agent design patterns. *ACM - Software Engeneerin Notes*, 29:1 – 8, May 2004.

[LO98] D. R. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, Reading, MA, 1998.

[LO99] D. B. Lange and M. Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89, March 1999.

[LW99] Bo Lindstrom and Lisa Wells. *Design/CPN Performance Tool Manual*. University of Aarhus, Denmark, 1.0 edition, September 1999.

[MCM01] José Merseguer, Javier Campos, and Eduardo Mena. Performance analysis of internet based software retrieval systems using petri nets. In *Proceedings of the 4th ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 47–56. ACM Press, 2001.

[Med03] P. S. Medcraft. Integração de bancos de dados federados na web usando agentes móveis. Master's thesis, Universidade Federal de Campina Grande, 2003.

[Mik98] Tommi Mikkonen. Formalizing design patterns. In *Proceedings of the 20th international conference on Software engineering*, pages 115–124. IEEE Computer Society, 1998.

[MKC01] J. Mylopoulos, M. Kolp, and J. Castro. Uml for agent-oriented software development: The tropos proposal. In M. Gogolla and C. Kobryn, editors, *Proceedings of UML 2001*, volume 2185 of *Lecture Notes in Computer Science*, pages 422–441, 2001.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE, vol 77, No. 04, April 1989*, pages 541–580, 1989.

[Ode01] J. Odell. Agent uml. http://www.auml.org, 2001.

[OMG] OMG. http://www.omg.org.

[Pei02] H. Peine. Application and programming experience with the ara mobile agent system. Software-Practice and Experience, 2002.

[RB99] O.F. Rana and C. Biancheri. A petri net model of the meeting design pattern for mobile-stationary agent interaction. In *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences, 1999. HICSS-32.*, January 05 - 08 1999.

[RW00] O.F. Rana and D.W. Walker. A performance model for task and interaction patterns in mobile agent systems. In *Performance, Computing, and Communications Conference, 2000. IPCCC '00. Conference Proceeding of the IEEE International*, pages Pages: 478– 484, 2000.

[SG01] Jon Siegel and OMG Staff Strategy Group. Developing in omg's model-driven architecture. Technical Report Revision 2.6, Object Management Group White Paper, November 2001.

[SMT98] G. Di Marzo Serugendo, M. Muhugusa, and C. Tschudin. A survey of theories for mobile agents. *World Wide Web Journal, special issue on Distributed World Wide Web Processing: Applications and Techniques of Web Agents*, 1998.

[SSJ] Ajay Kr. Singh, Ravi Sankar, and Vikram Jamwal. Design patterns for mobile agent applications.

[Sun] Sun. http://java.sun.com.

[TCL] Agent TCL. http://agent.cs.dartmouth.edu/.

[TF02] B. Thalheim T. Feyer. A model for defining and composing interaction patterns. In *12th European-Japanese Conference on Information Modelling and Knowledge Bases - EJC'2002, Krippen, Germany*, May 2002.

[TOH99] Yasuyuki Tahara, Akihiko Ohsuga, and Shinichi Honiden. Agent system development method based on agent patterns. In *Proceedings of the 21st international conference on Software engineering*, pages 356–367. IEEE Computer Society Press, 1999.

[TOH00] Y. Tahara, A. Ohsuga, and S. Honiden. Safety and security in mobile agents. In *Draft Proceedings of AOSE2000*, pages 107– 126, 2000.

[TTOH01] Yasuyuki Tahara, Nobukazu Toshiba, Akihiko Ohsuga, and Shinichi Honiden. Secure and efficient mobile agent application reuse using patterns. In *Proceedings of the 2001 symposium on Software reusability*, pages 78–85. ACM Press, 2001.

[VBML99] M. T. de O. Valente, R. da S. Bigonha, M. de A. Maia, and A. A. F. Loureiro. Aplicação de asm na especificação de sistemas móveis. In *Workshop de Métodos Formais*, 1999.

[vdAtH] W.M.P. van der Aalst and A.H.M. ter Hofstede. Workflow patterns: On the expressive power of (petri-net-based) workflow languages.

[WC01] Mike Wooldridge and P. Ciancarini. Agent-oriented software engineering. In S. K. Chang, editor, *Handbook of Software Engineering and Knowledge Engineering*. World Scientific Publishing Co., 2001.

[Whi96] J. E. White. *TeleScript Technology: Mobile Agents, Software Agents*. ed. AAAI Press/MIT Press, 1996.

[WRMT98] W.J.Brown, R.C.Malveau, H.W.III McCormick, and T.J.Mowbray. *Antipatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, 1998.

[XD00] D. Xu and Y. Deng. Modeling mobile agent systems with high-level petri nets, 2000.

[YBF98] M.-J. Yoo, J-.P. Briot, and J. Ferber. Using components for modeling intelligent and collaborative mobile agents. In *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE 1998*. IEEE, 1998.

Apêndice A

Catálogo de Padrões de Projeto para Agentes Móveis

A.1 Migração

- *Itinerary*[AL98];
- *Forwarding*[AL98];
- *Ticket*[AL98];
- *Basic Moblity*[TOH99];
- *Clone*[KKPS98];
- *Remote Configurator*[KKPS98];
- *Migration Thread Factory*[KKPS98].

Itinerary

- **Descrição**: Este padrão provê uma forma de migração de um agente, que será responsável por executar uma dada tarefa em computadores remotos. O agente recebe um itinerário na agência de origem, indicando a seqüência de agências que deve visitar. Uma vez em uma agência, o agente executa sua tarefa localmente, e depois continua

em seu itinerário. Após visitar a última agência, o agente retorna para sua agência de origem. Proposto em [TOH99].

- **Motivação**: O agente realiza toda a migração sem precisar voltar na agência de origem. Além disto, a tarefa é realizada localmente em cada agência.
- **Aplicabilidade**: Execução de tarefas remotas e seqüenciais.
- **Conseqüências**: O agente precisa conhecer todas as agências que irá visitar antes de sair da agência de origem. Além disto, a migração do agente pode ficar lenta se ele recuperar muitos dados em cada agência que visitar.
- **Usos conhecidos**: Sistema de recuperação de informação distribuída [LdFG04]. Sistema de Apoio às Atividades de Comitês de Programa em Conferências [Gue02a].
- **Padrões relacionados**: *Branching* e *Star-Shaped*.
- **Estrutura e Implementação**

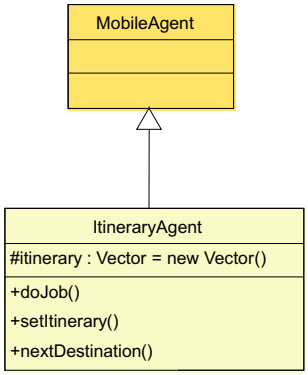


Figura A.1: Diagrama de classes independente para o padrão *Itinerary*.

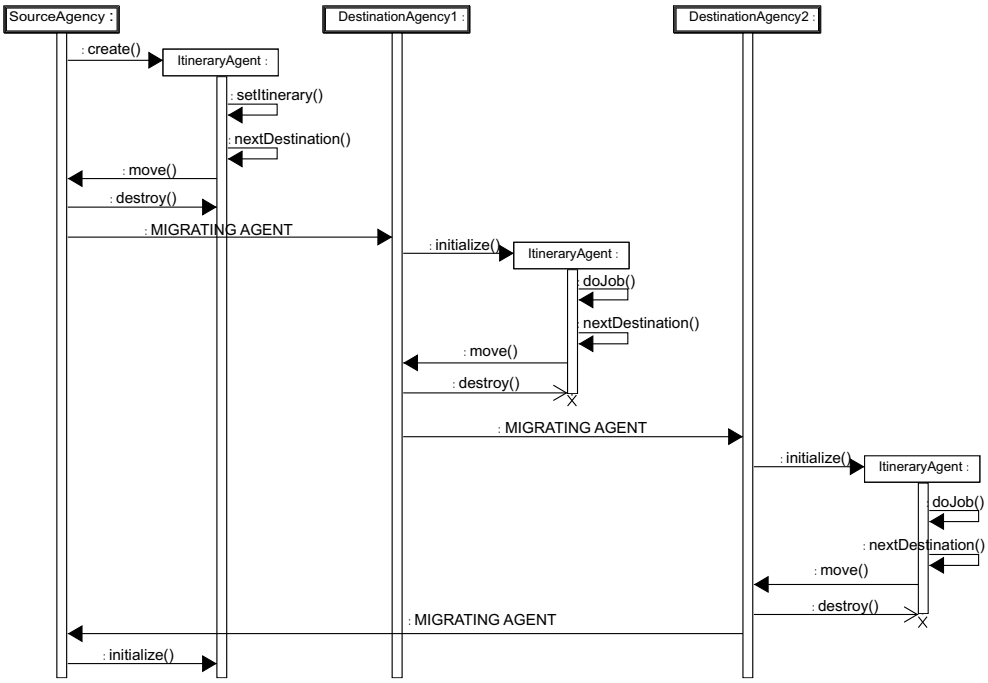


Figura A.2: Diagrama de seqüências independente para o padrão *Itinerary*.

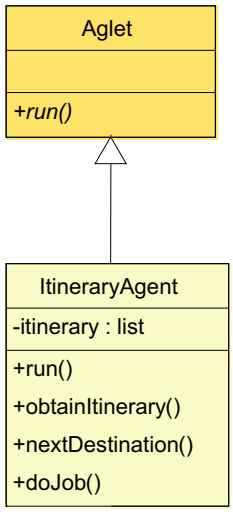


Figura A.3: Diagrama de classes para o padrão *Itinerary* na plataforma Aglets.

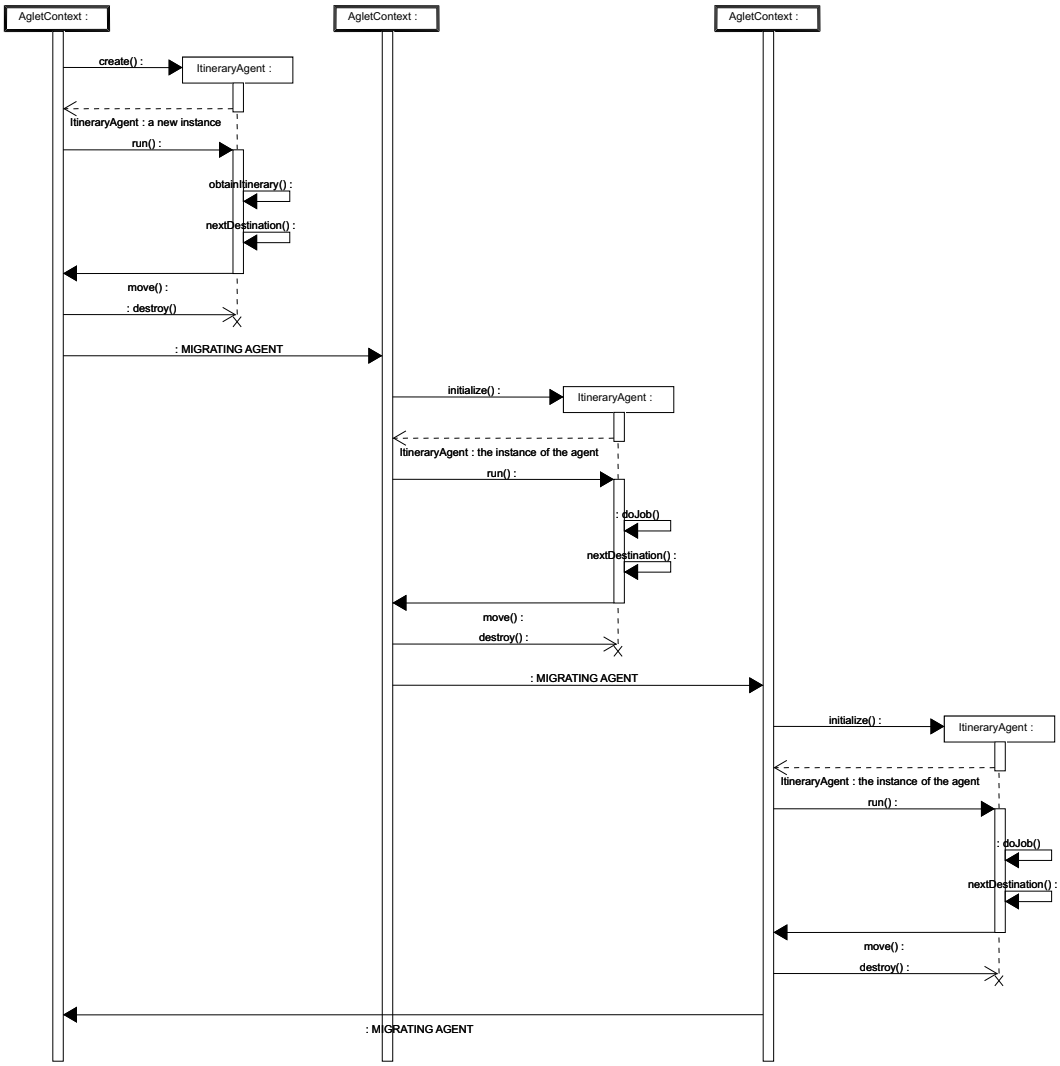


Figura A.4: Diagrama de seqüências para o padrão *Itinerary* na plataforma Aglets.

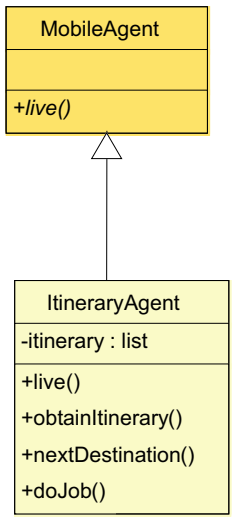


Figura A.5: Diagrama de classes para o padrão *Itinerary* na plataforma Grasshopper.

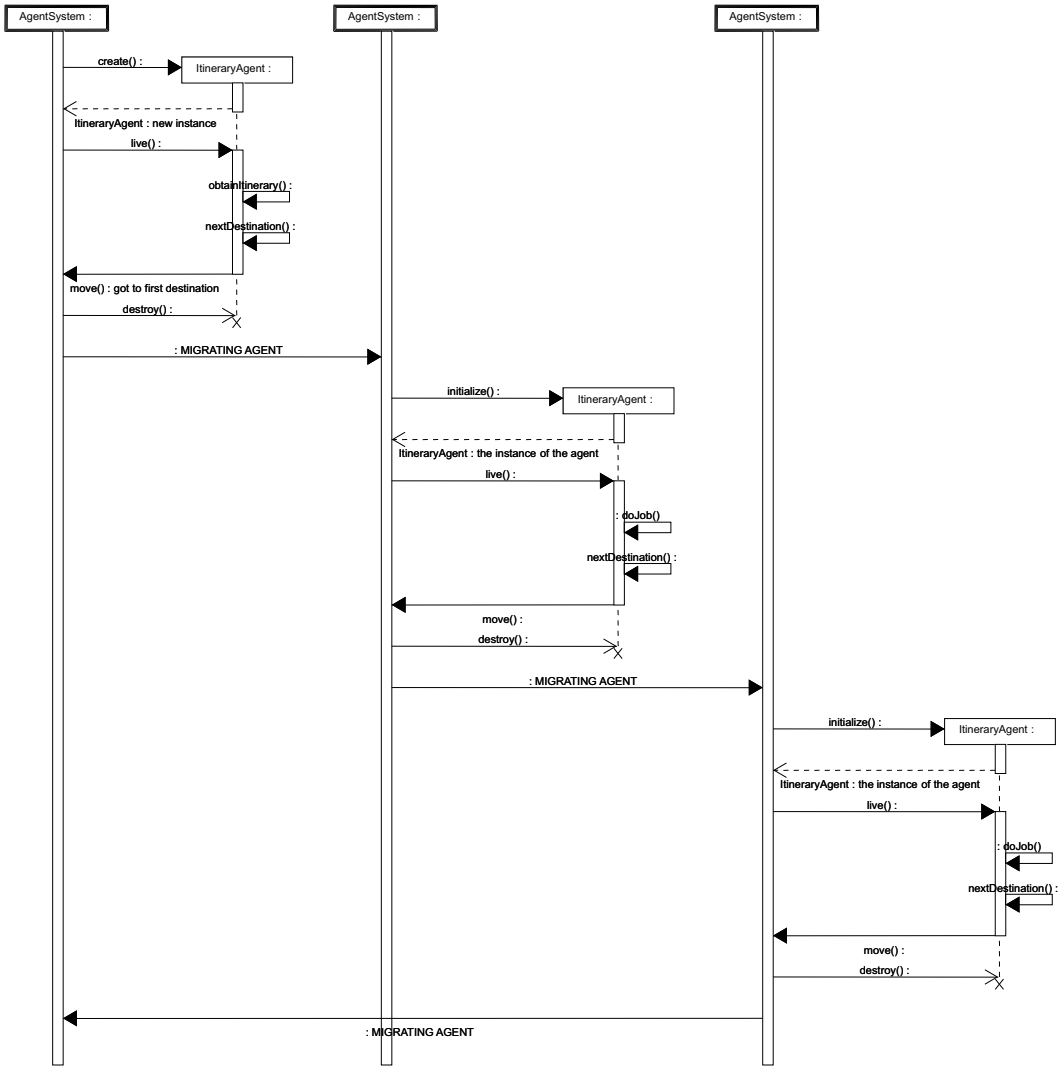


Figura A.6: Diagrama de seqüências para o padrão *Itinerary* na plataforma Grasshopper.

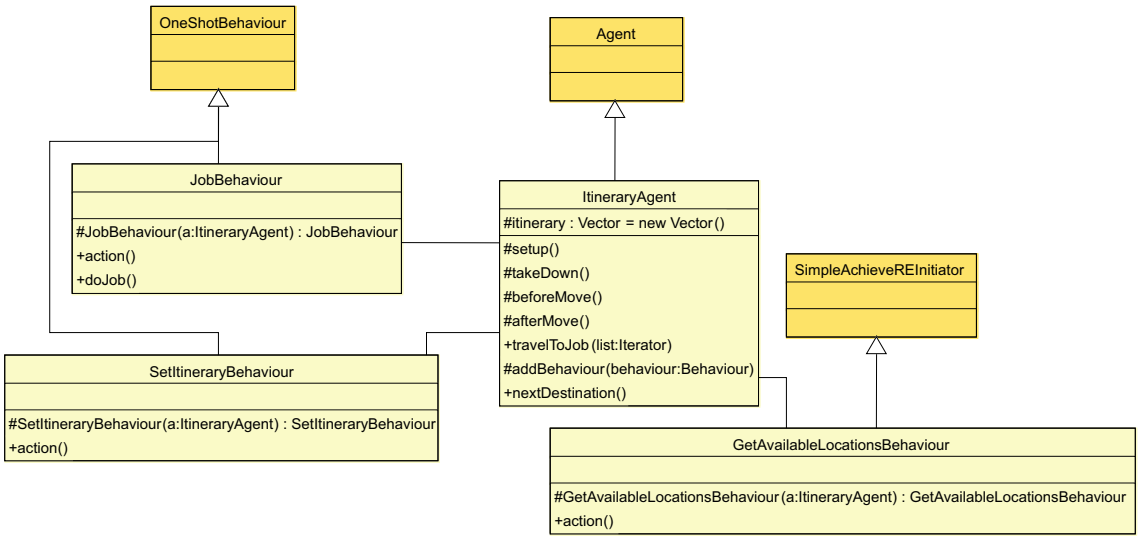


Figura A.7: Diagrama de classes para o padrão *Itinerary* na plataforma JADE.

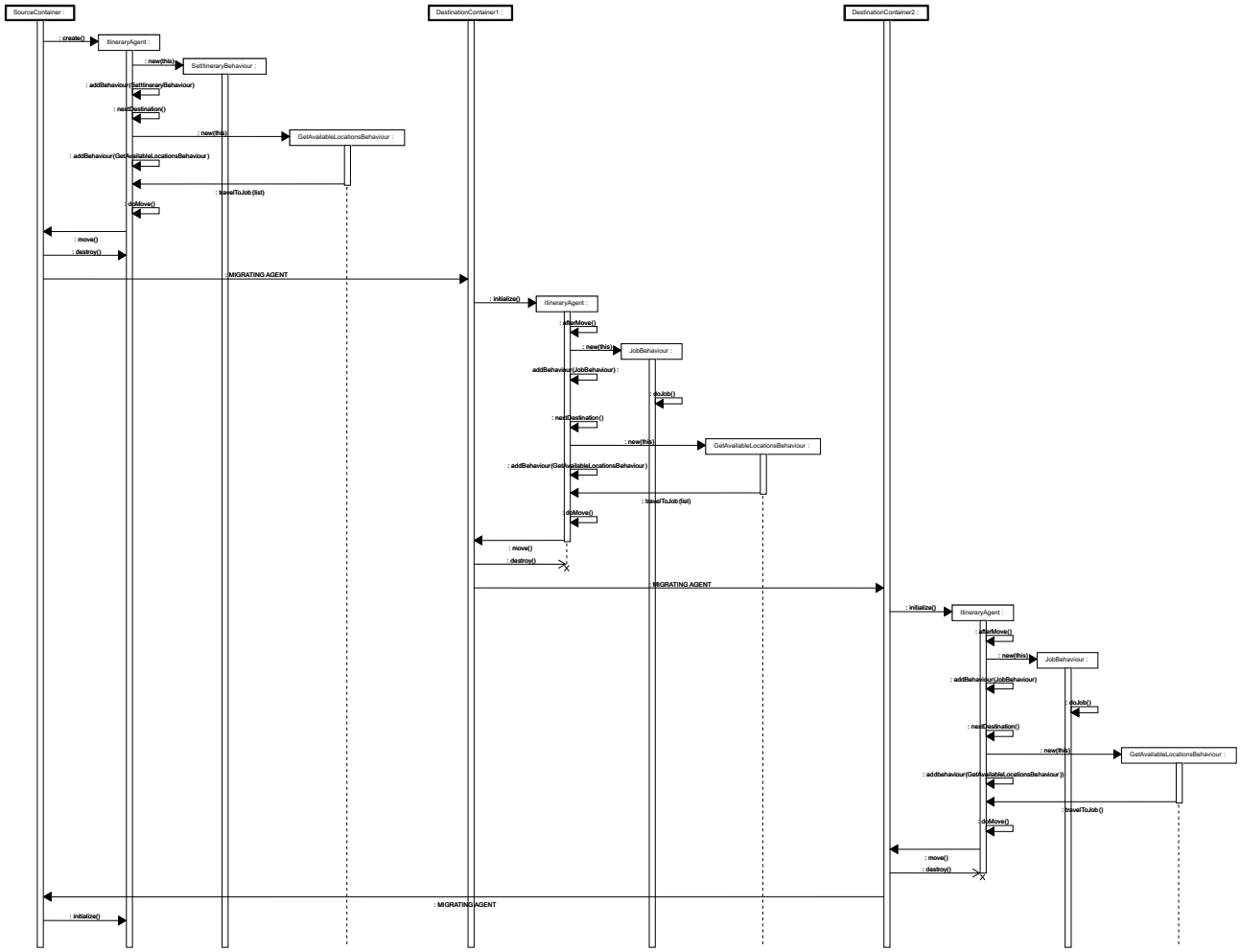


Figura A.8: Diagrama de seqüências para o padrão *Itinerary* na plataforma JADE.

Star-shaped

- **Descrição**: No padrão Star-Shaped, o agente recebe uma lista de agências para onde deve migrar. Então, ele migra para a primeira agência de destino, onde executa a tarefa e retorna para a agência de origem. O agente repete esse ciclo até visitar a última agência em sua lista. Proposto em [TOH99].

- **Motivação**: Neste padrão, como o agente sempre volta para a agência de após visitar as agências remotas, ele poderá armazenar os dados coletados na agência de origem, não precisando migrar para as demais com estes dados. Além disto, se alguma agência de sua lista estiver com problemas, o agente poderá ser avisado ao retornar para agência de origem.

- **Aplicabilidade**: Este padrão é interessante para execução de tarefas seqüenciais remotas, em que o agente irá carregar muitos dados.

- **Conseqüências**: É necessário definir um local para armazenamento dos dados coletados pelo agente, na agência de origem. Deve-se cuidar da segurança dos dados armazenados, evitando o acesso e/ou alteração por entidade não-autorizadas.

- **Usos conhecidos**: Sistema de recuperação de informação distribuída [LdFG04].

- **Padrões relacionados**: *Branching* e *Itinerary*.

- **Estrutura e Implementação**

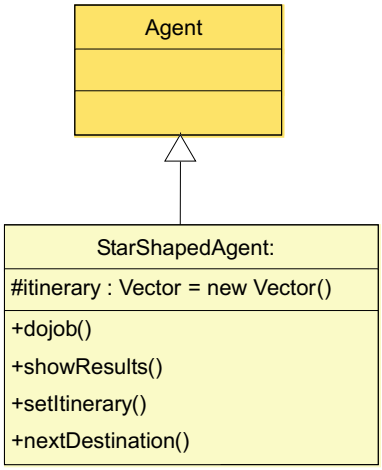


Figura A.9: Diagrama de classes independente para o padrão *Star-Shaped*.

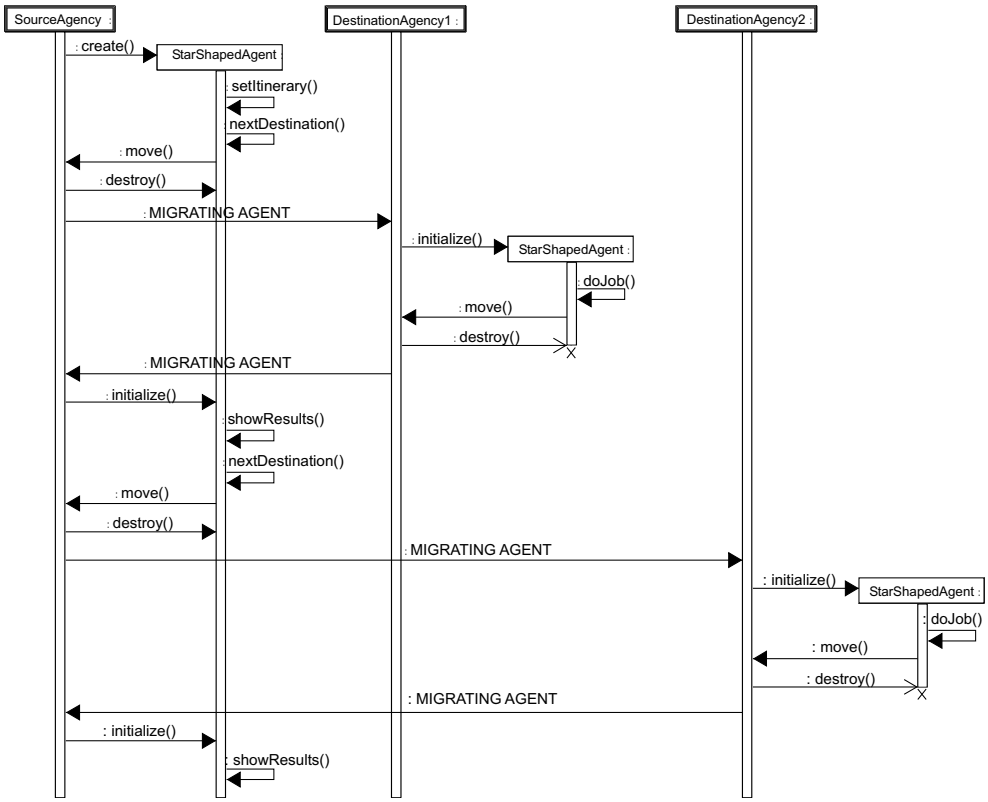


Figura A.10: Diagrama de seqüências independente para o padrão *Star-Shaped*.

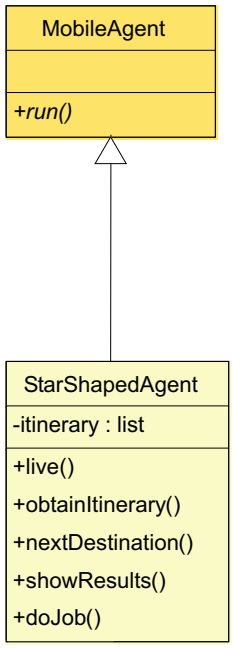


Figura A.11: Diagrama de classes para o padrão *Star-Shaped* na plataforma Aglets.

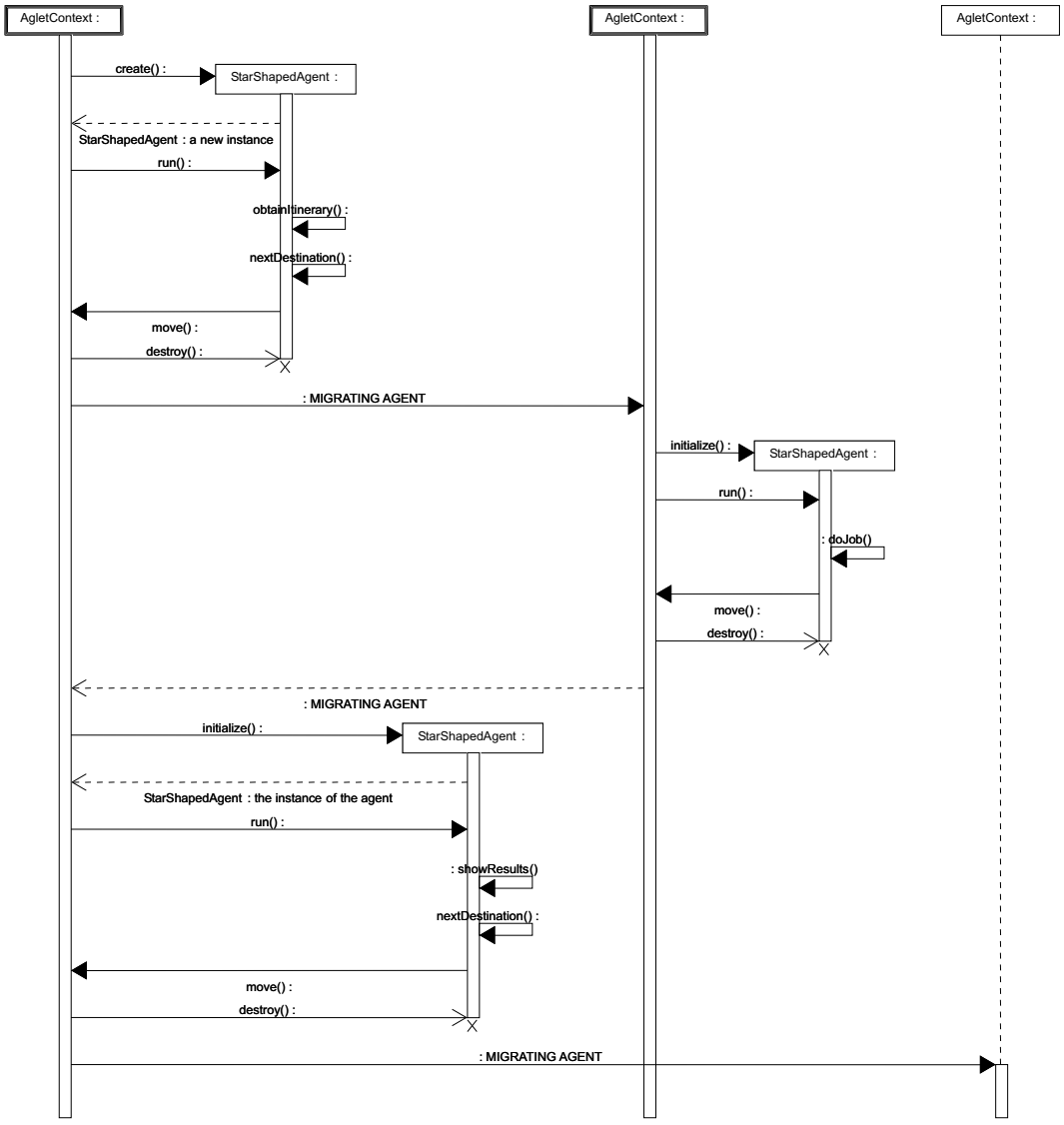


Figura A.12: Diagrama de seqüências para o padrão *Star-Shaped* na plataforma Aglets.

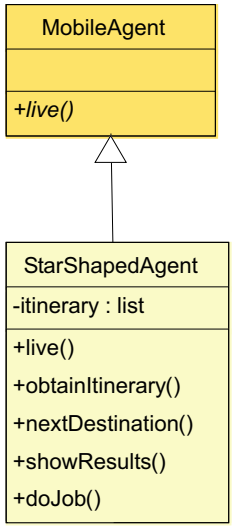


Figura A.13: Diagrama de classes para o padrão *Star-Shaped* na plataforma Grasshopper.

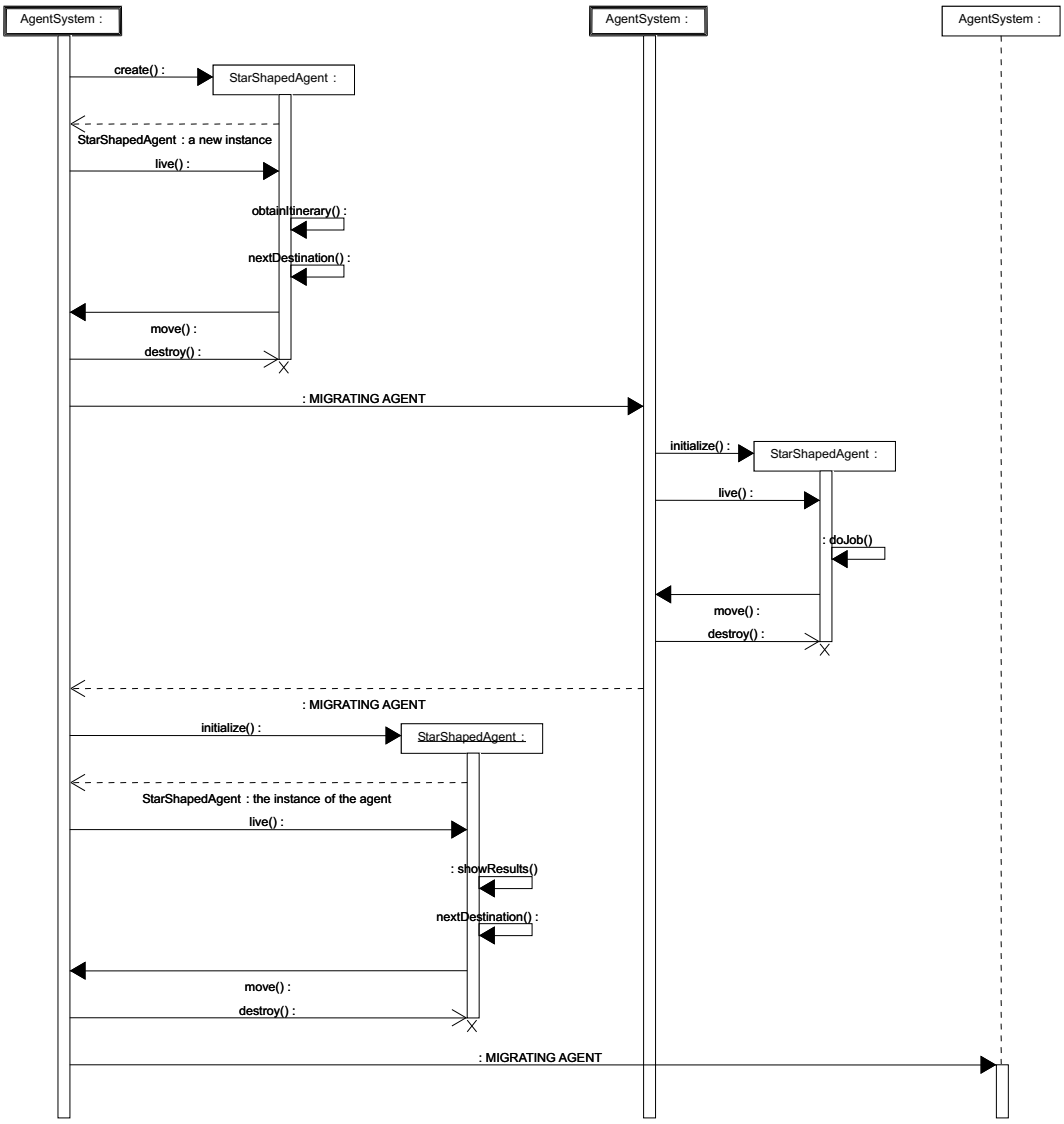


Figura A.14: Diagrama de seqüências para o padrão *Star-Shaped* na plataforma Grasshopper.

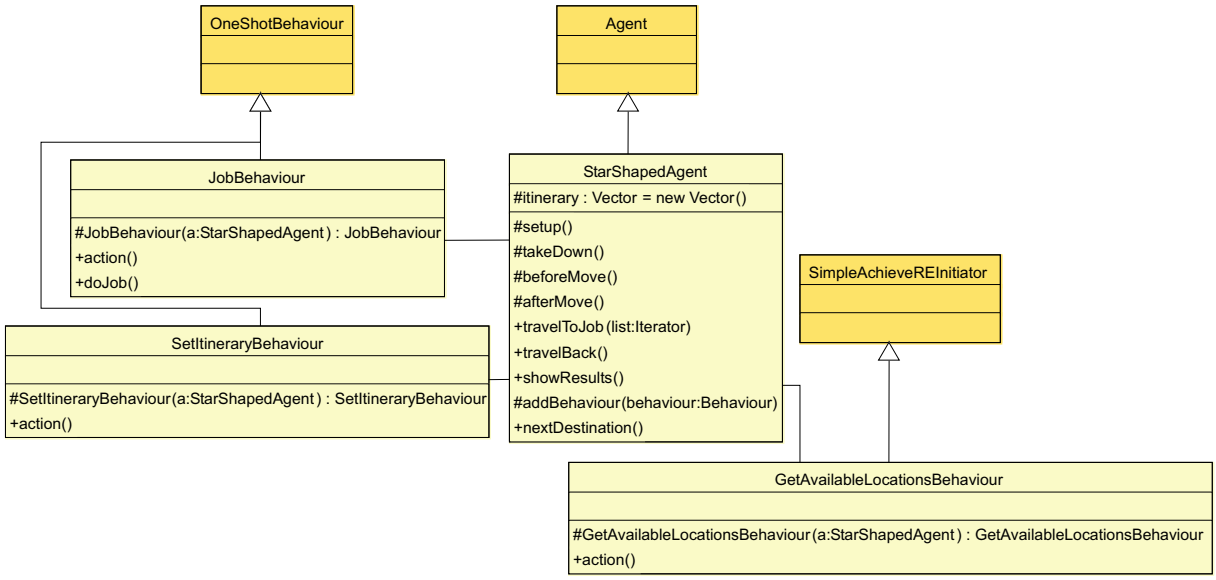


Figura A.15: Diagrama de classes para o padrão *Star-Shaped* na plataforma JADE.

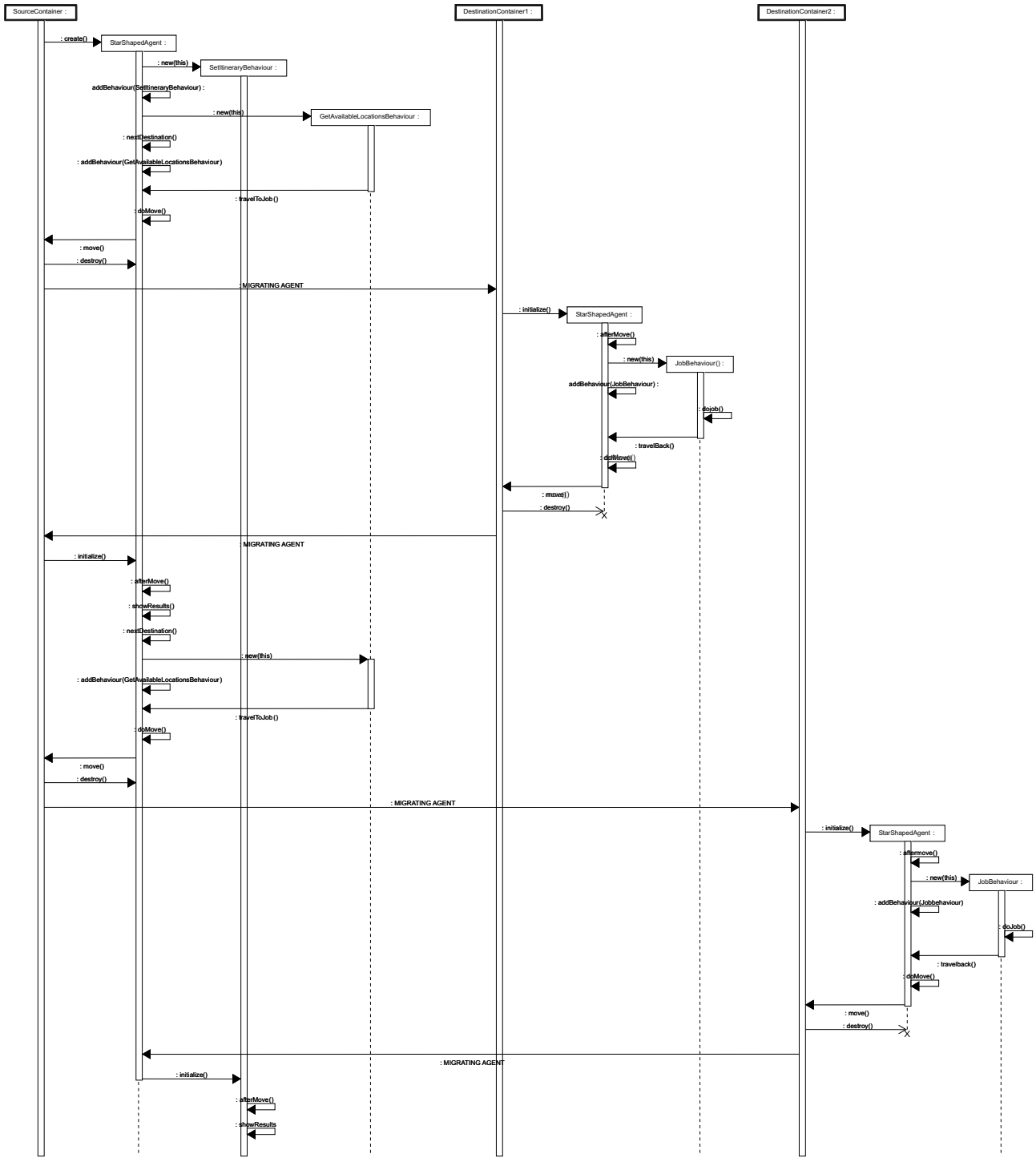


Figura A.16: Diagrama de seqüências para o padrão *Star-Shaped* na plataforma JADE.

Branching

- **Descrição**: Neste padrão, o agente recebe uma lista de agências para visitar e se clona de acordo com o número de agências em seu itinerário. Então, cada clone irá visitar uma agência da lista, onde executará uma tarefa e notificará a agência de origem quando a tarefa estiver completa. O tratamento dos resultados finais é uma questão não coberta por esse padrão. Por exemplo, os clones podem mostrar os resultados da tarefa em uma interface com o usuário ou enviá-los para um outro agente. Proposto em [TOH99].

- **Motivação**: A importância desse padrão é que ele divide tarefas que podem ser executadas em paralelo. Assim, pode-se completar a realização de toda a tarefa de forma mais rápida.

- **Aplicabilidade**: Execução de tarefas remotas, que possam ser divididas para execução em paralelo por vários agentes.

- **Conseqüências**: É necessário ter um controle sobre os agentes que estão executando a tarefa, de forma a evitar que algum agente se perca ou não finalize a tarefa.

- **Usos conhecidos**: Sistema de recuperação de informação distribuída [LdFG04].

- **Padrões relacionados**: *Itinerary* e *Star-Shaped*.

- **Estrutura e Implementação**

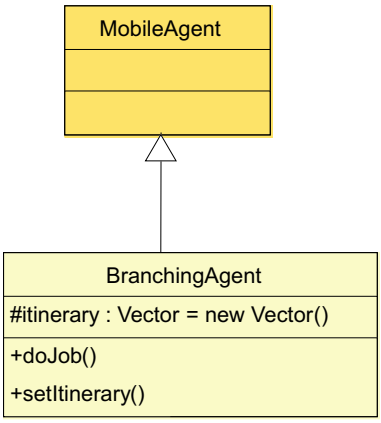


Figura A.17: Diagrama de classes independente para o padrão *Branching*.

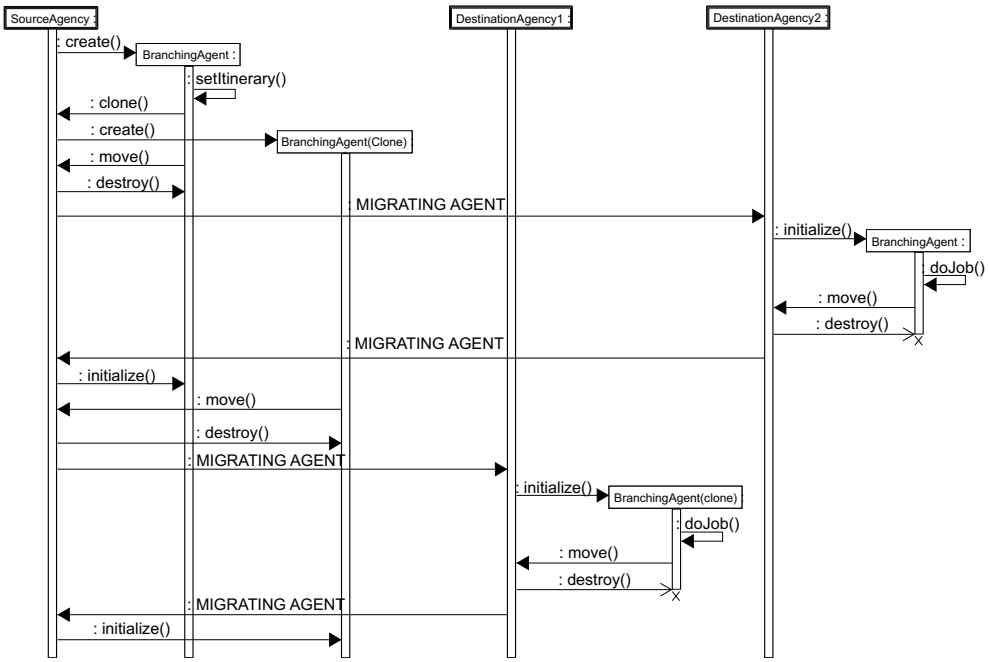


Figura A.18: Diagrama de seqüências independente para o padrão *Branching*.

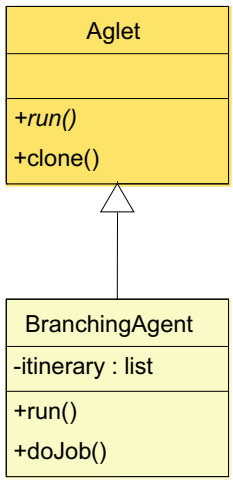


Figura A.19: Diagrama de classes para o padrão *Branching* na plataforma Aglets.

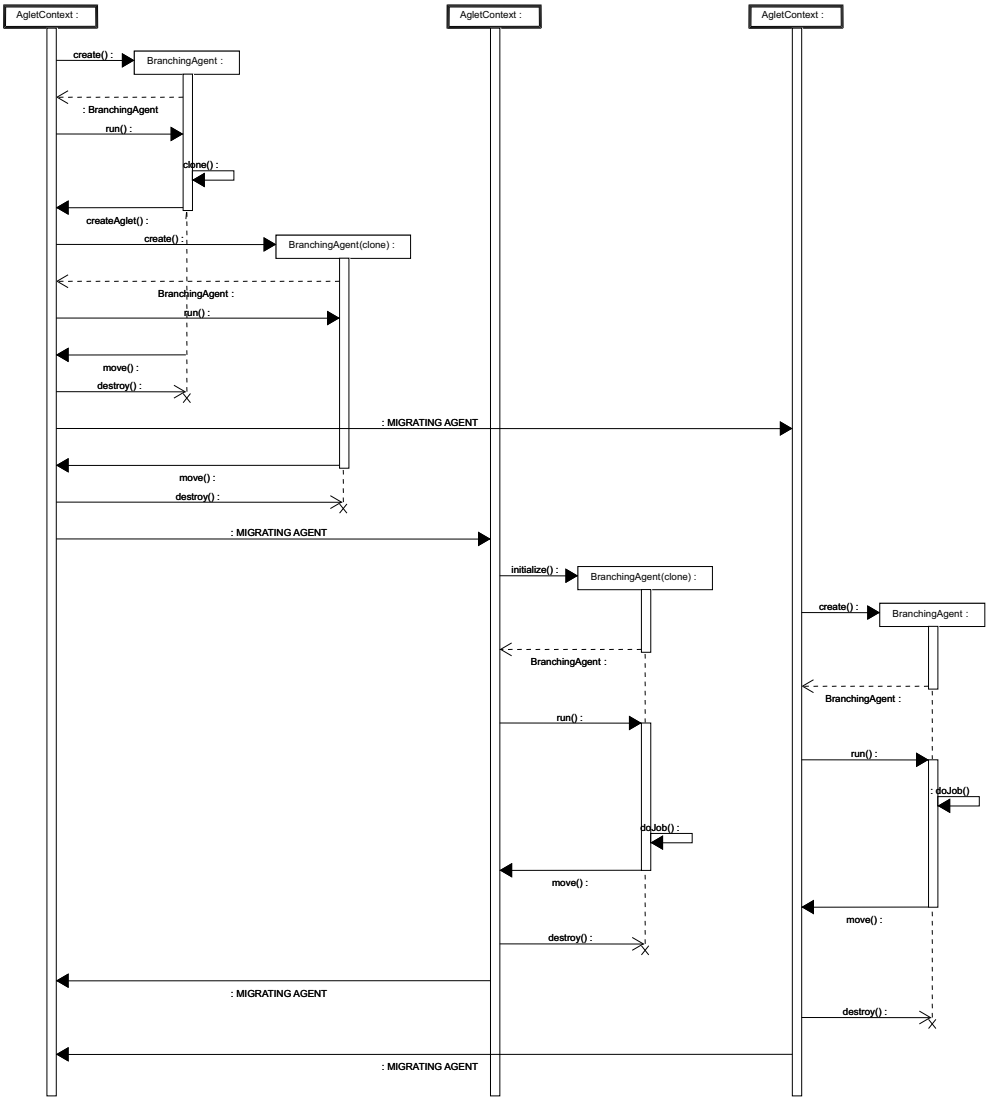


Figura A.20: Diagrama de seqüências para o padrão *Branching* na plataforma Aglets.

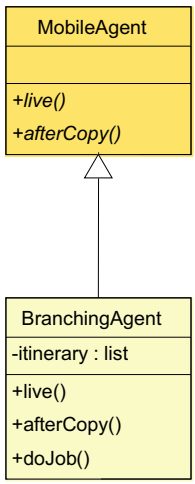


Figura A.21: Diagrama de classes para o padrão *Branching* na plataforma Grasshopper.

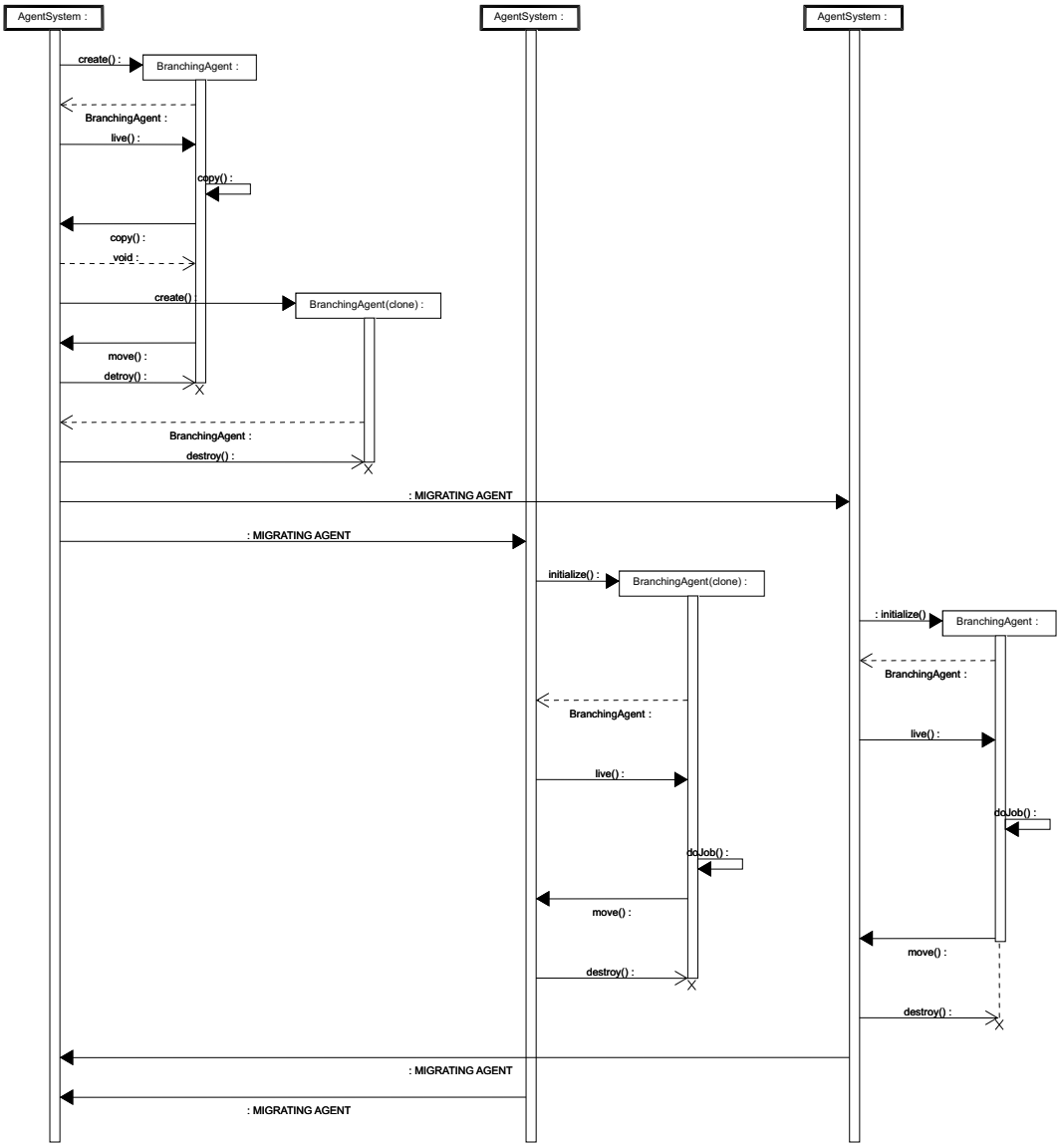


Figura A.22: Diagrama de seqüências para o padrão *Branching* na plataforma Grasshopper.

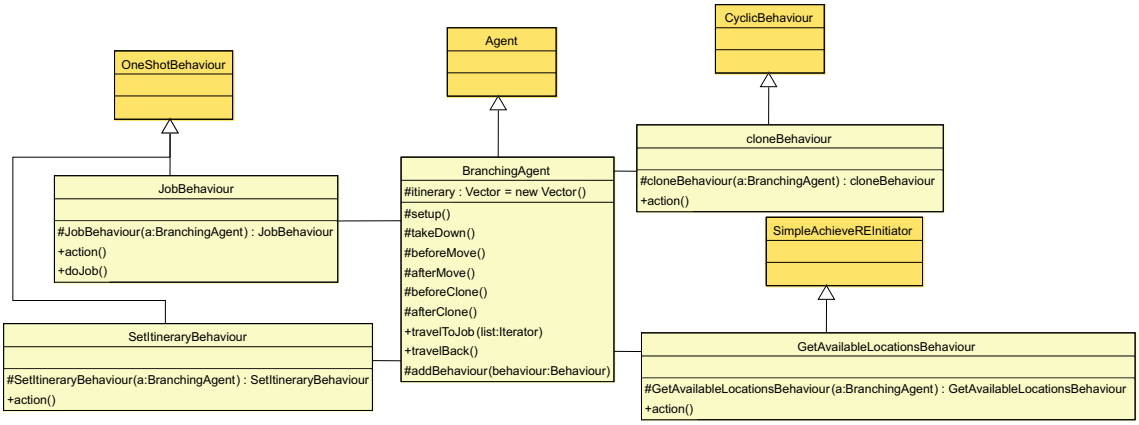


Figura A.23: Diagrama de classes para o padrão *Branching* na plataforma JADE.

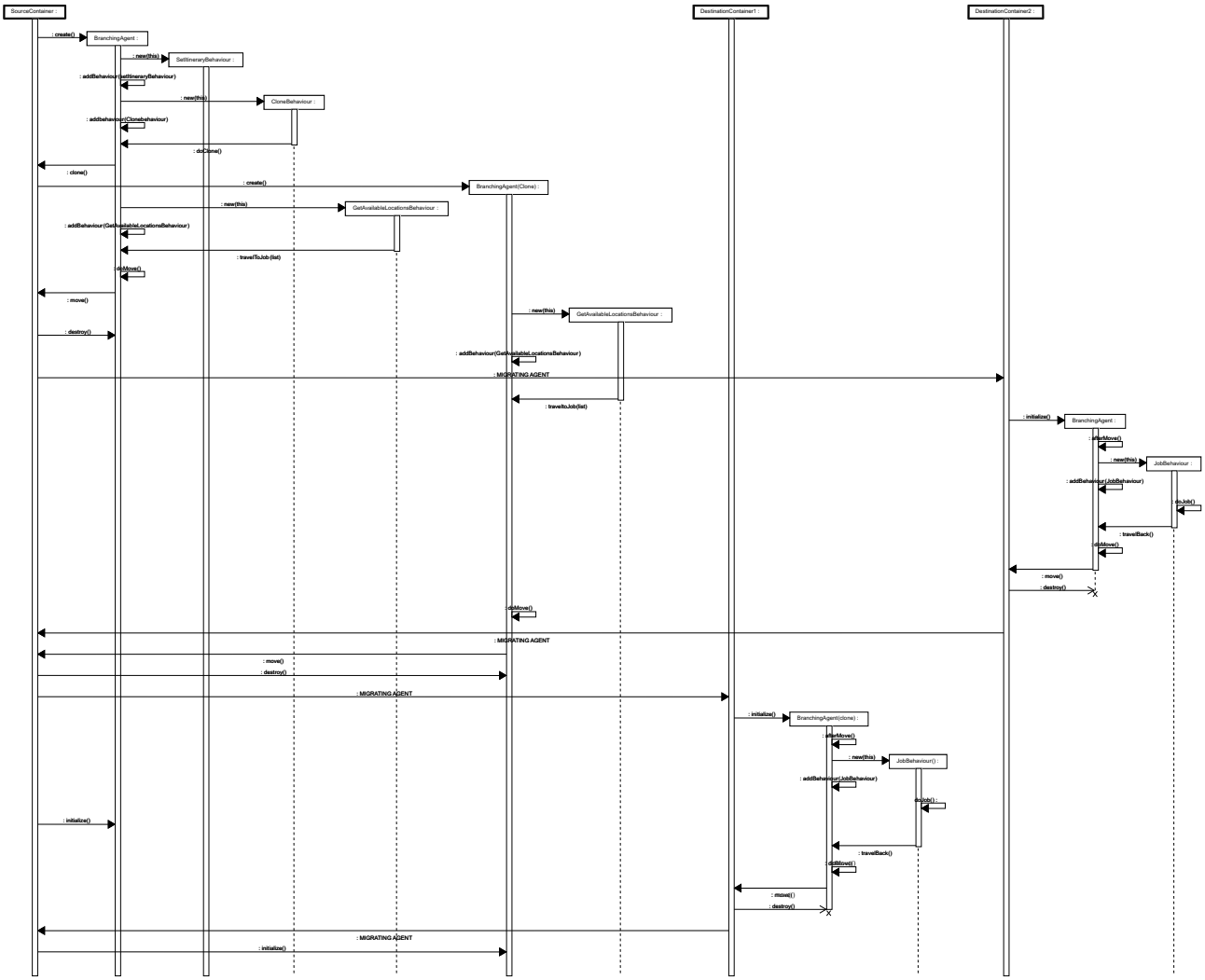


Figura A.24: Diagrama de seqüências para o padrão *Branching* na plataforma JADE.

A.2 Tarefa

- *Master-Slave*[AL98];
- *Plan*[AL98]);

A.3 Comunicação

- *Locker*[AL98];
- *Messenger*[AL98];
- *Organized Group*[AL98];
- *Cooperation Protocol*[TOH99];
- *Direct Interaction*[TOH99];
- *Mediation*[TOH99];
- *Dispatching*[TOH99];
- *Direct Coupling*[DOKO99];
- *Proxy Agent*[DOKO99];
- *Communication Session*[DOKO99];
- *Badges*[DOKO99];
- *Event Dispatcher*[DOKO99];
- *Messenger*[SSJ];
- $P_1 ... P_7$[TTOH01].

Meeting

- **Descrição**: Agentes dispersos na rede migram para um local comum com o objetivo de interagirem localmente. Proposto em [AL98].
- **Motivação**: Com esta padrão, poderá se obter uma otimização de resultados, fazendo com que resultados repetidos, por exemplos, sejam descartados. Além disto, como a interação é local, ocorrerá um redução da latência, e os agentes terão uma comunicação rápida e direta.
- **Aplicabilidade**: Sistemas que envolvam muitos agentes dispersos, que estejam carregando muitos dados, como sistemas de comércio eletrônico e recuperação de informação.
- **Conseqüências**: É necessário manter uma agência disponível, onde ocorrerá o encontro. Além disto, nesta agência deverá existir um gerenciador do encontro, que será responsável pelo cadastro e remoção de agentes no encontro.
- **Usos conhecidos**: Pesquisa de Dados Distribuídos da Web [LMAFRS03].
- **Padrões relacionados**: *Group Communication*
- **Estrutura e Implementação**

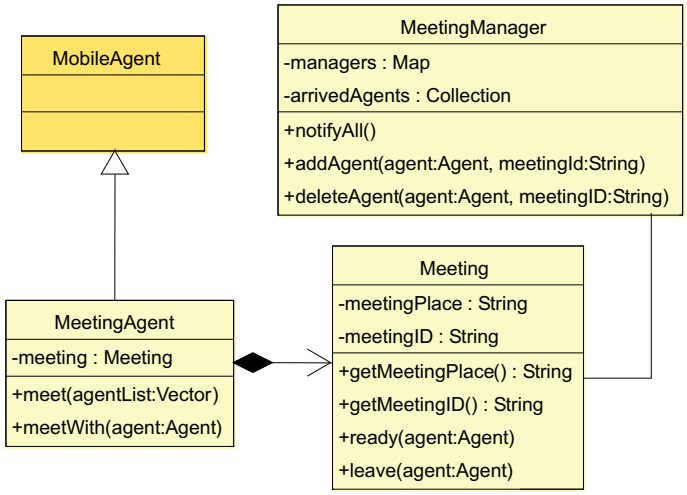


Figura A.25: Diagrama de classes independente para o padrão *Meeting*.

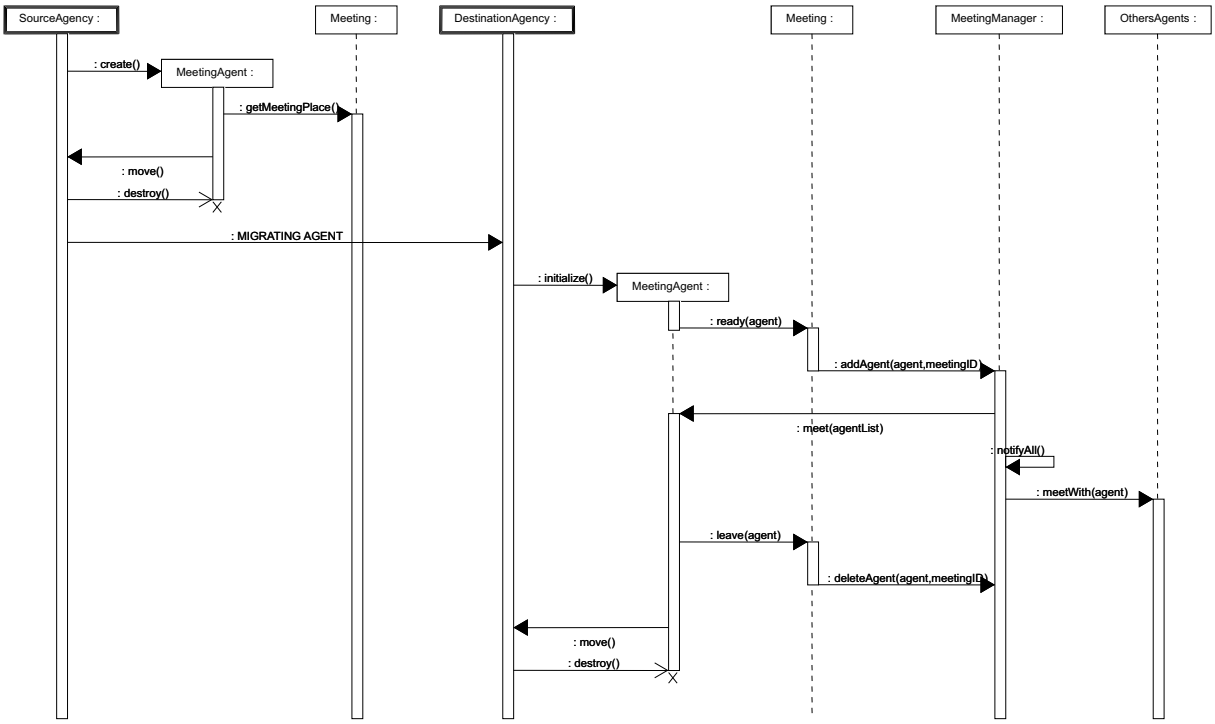


Figura A.26: Diagrama de seqüências independente para o padrão *Meeting*.

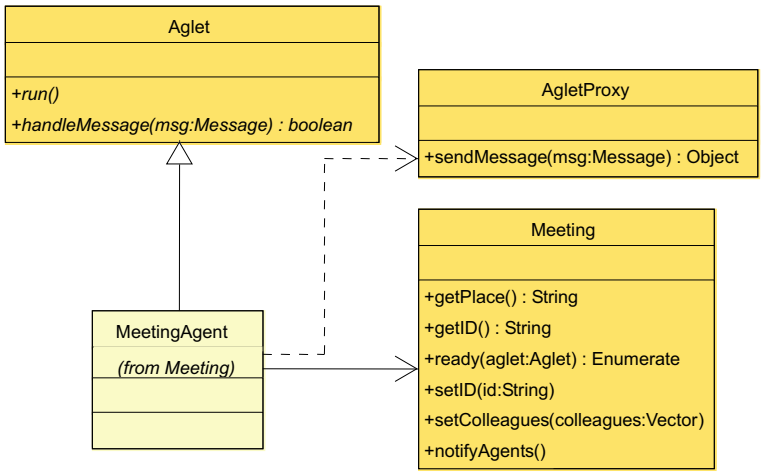


Figura A.27: Diagrama de classes para o padrão *Meeting* na plataforma Aglets.

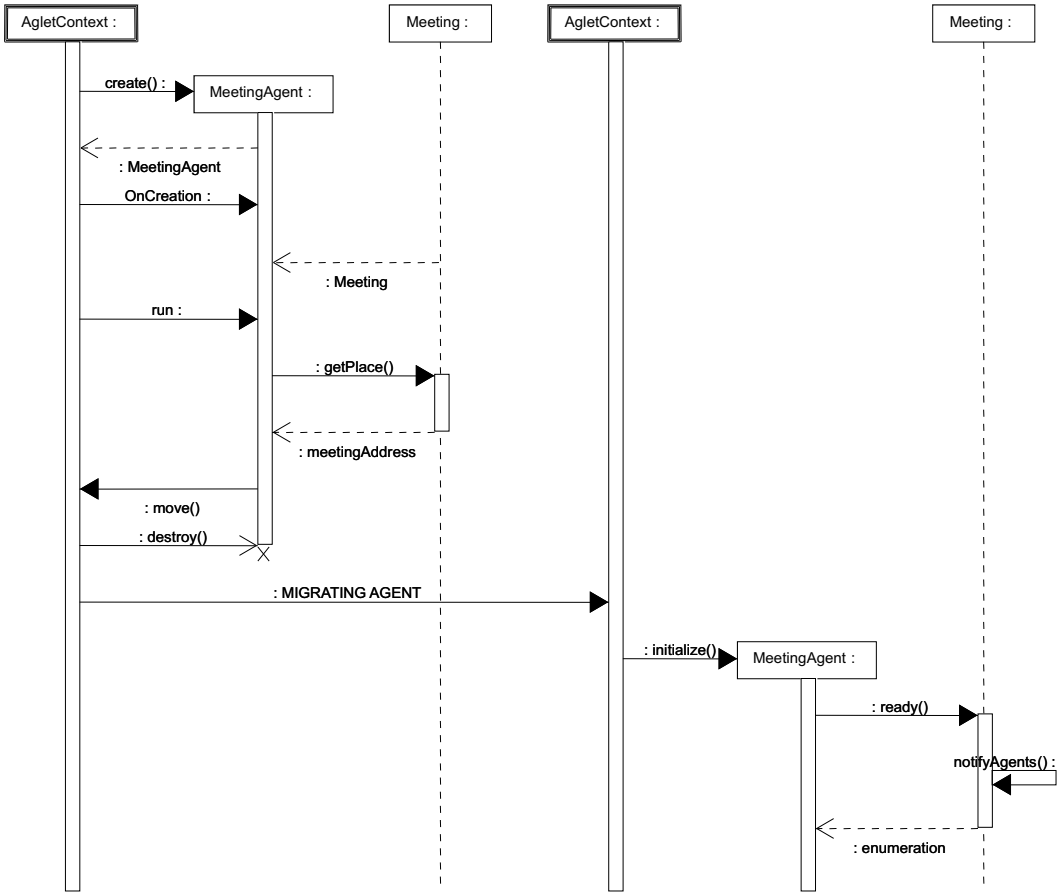


Figura A.28: Diagrama de seqüências para o padrão *Meeting* na plataforma Aglets.

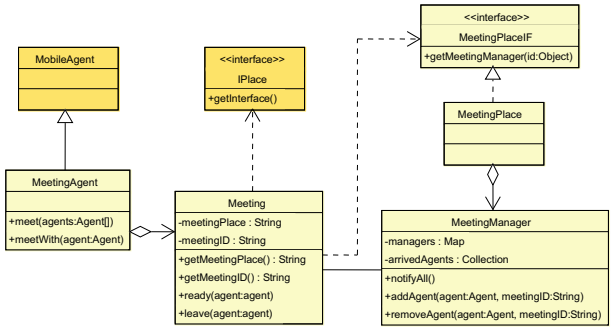


Figura A.29: Diagrama de classes para o padrão *Meeting* na plataforma Grasshopper.

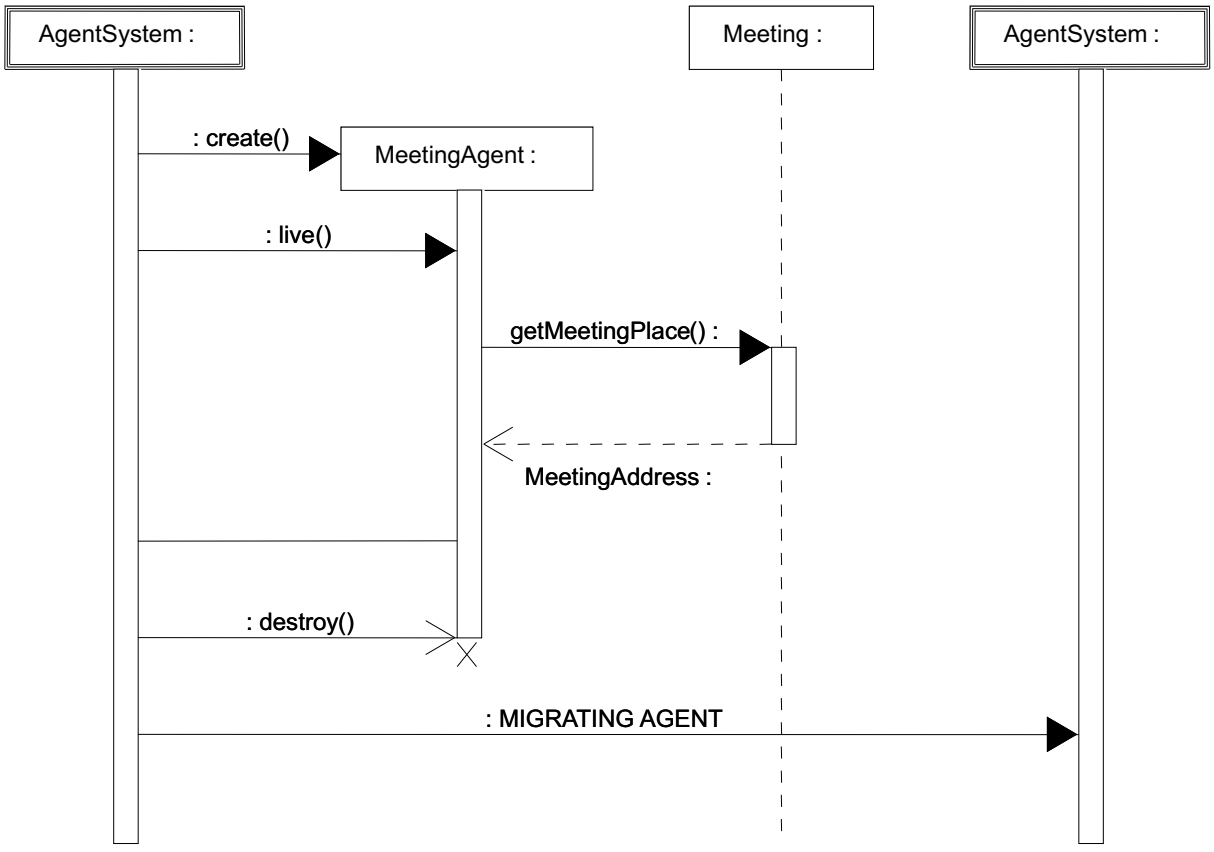


Figura A.30: Diagrama de seqüências para o padrão *Meeting* na plataforma Grasshopper (primeira parte).

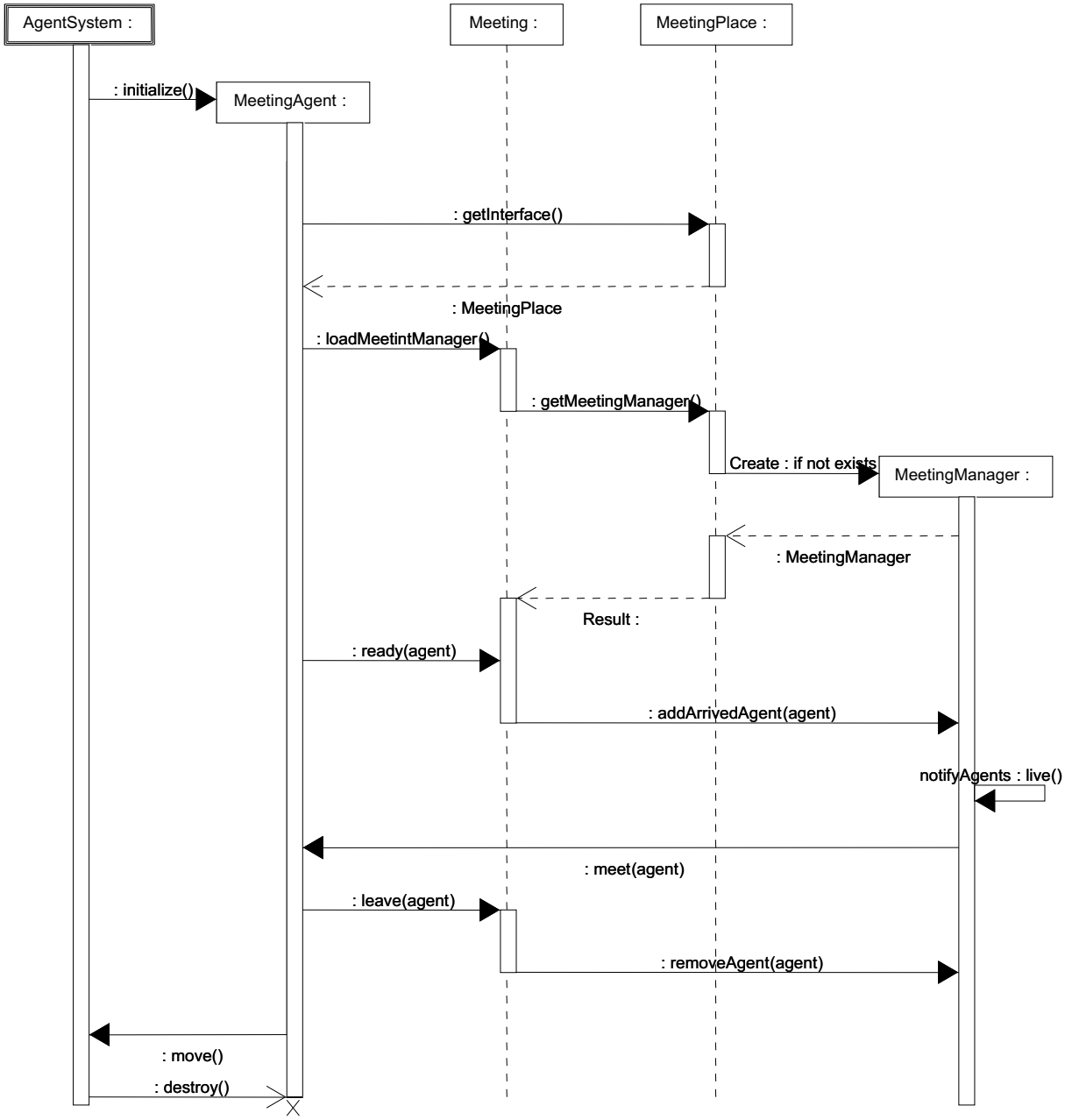


Figura A.31: Diagrama de seqüências para o padrão *Meeting* na plataforma Grasshopper (segunda parte).

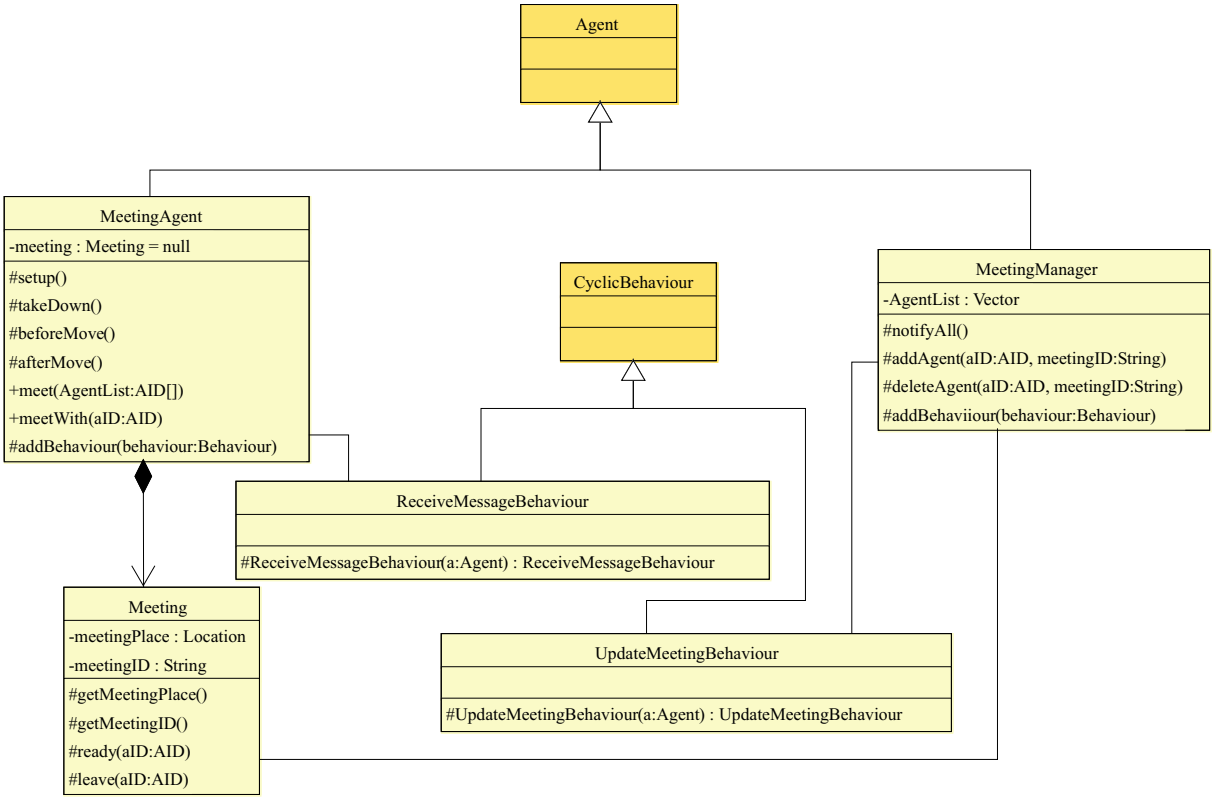


Figura A.32: Diagrama de classes para o padrão *Meeting* na plataforma JADE.

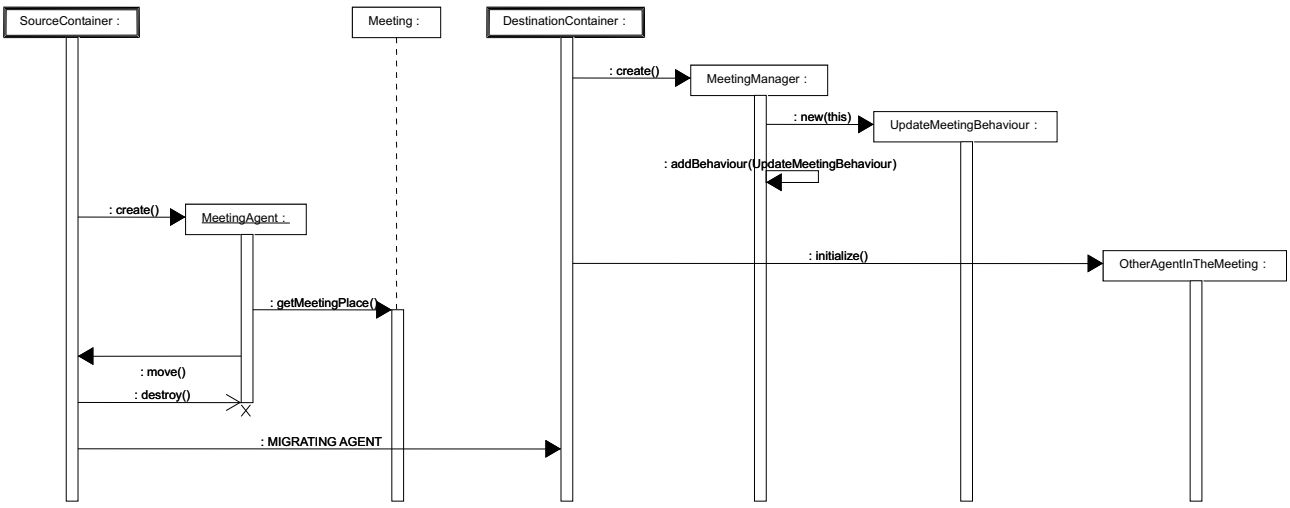


Figura A.33: Diagrama de seqüências para o padrão *Meeting* na plataforma JADE (primeira parte).

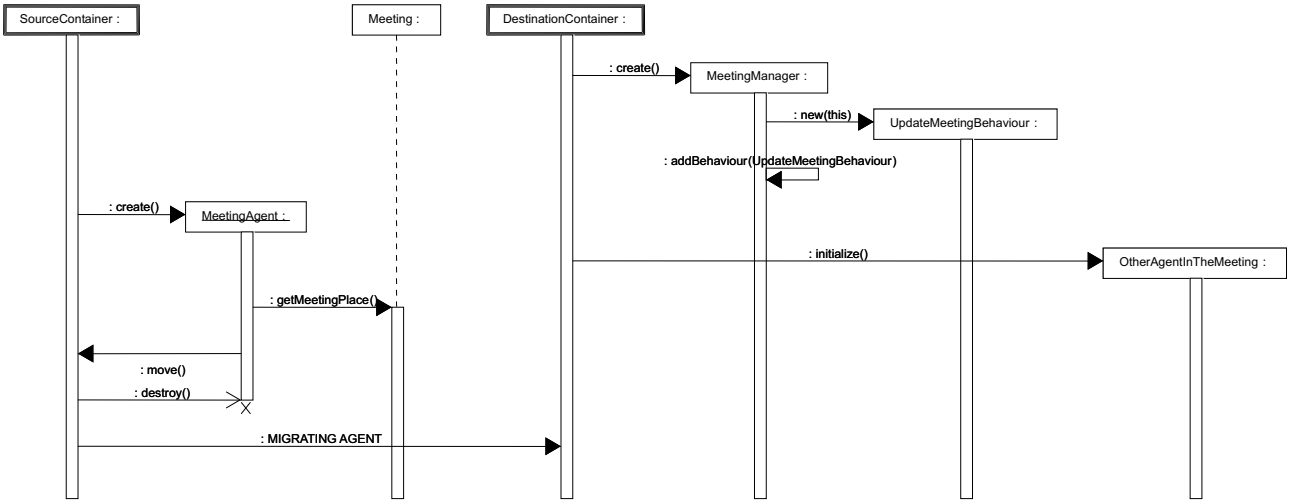


Figura A.34: Diagrama de seqüências para o padrão *Meeting* na plataforma JADE (segunda parte).

Group Communication

- **Descrição**: Quando precisa se comunicar com um grupo de agentes, o agente cria um agente mensageiro lhe informando a mensagem e o identificador do grupo. O mensageiro, então, recupera a lista de agentes daquele grupo em um diretório de grupos e migra para a agência de cada agente na lista, onde envia a mensagem para ele. Proposto em [SSJ].

- **Motivação**: Esta padrão facilita o envio de mensagens para vários agentes. Assim, o agente só precisará se preocupar em saber o grupo para o qual deseja enviar a mensagem, não precisando sequer conhecer os membros deste grupo.

- **Aplicabilidade**: Sistemas que dependam da comunicação entre vários agentes de um grupo cujos membros estão sempre mudando. Assim, poderíamos ter, por exemplo, um sistema de comércio eletrônico em que os agentes participantes da busca estão sempre mudando. Com este padrão, um agente só precisaria saber o nome do grupo de agentes executando, não precisando saber quais agente estão realizando a busca no momento.

- **Conseqüências**: É preciso manter um diretório atualizado com os grupos, membros do grupo e localização de cada membro.

- **Padrões relacionados**: *Meeting*

- **Estrutura e Implementação**

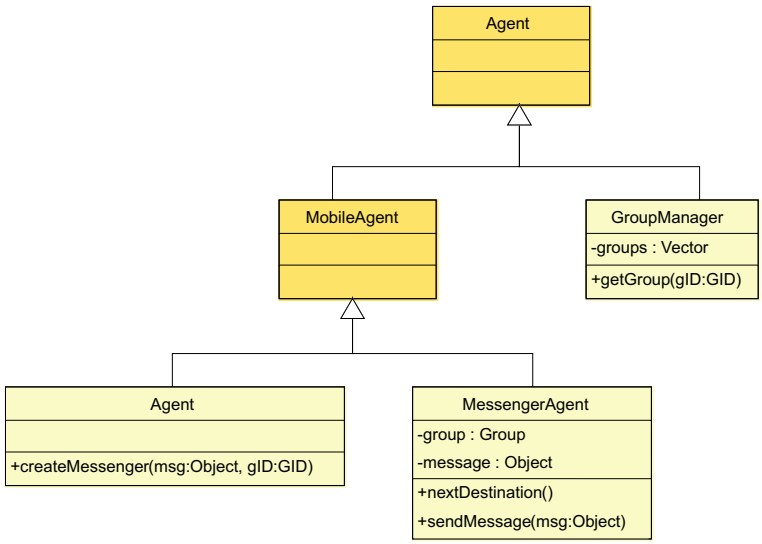


Figura A.35: Diagrama de classes independente para o padrão *Group Communication*.

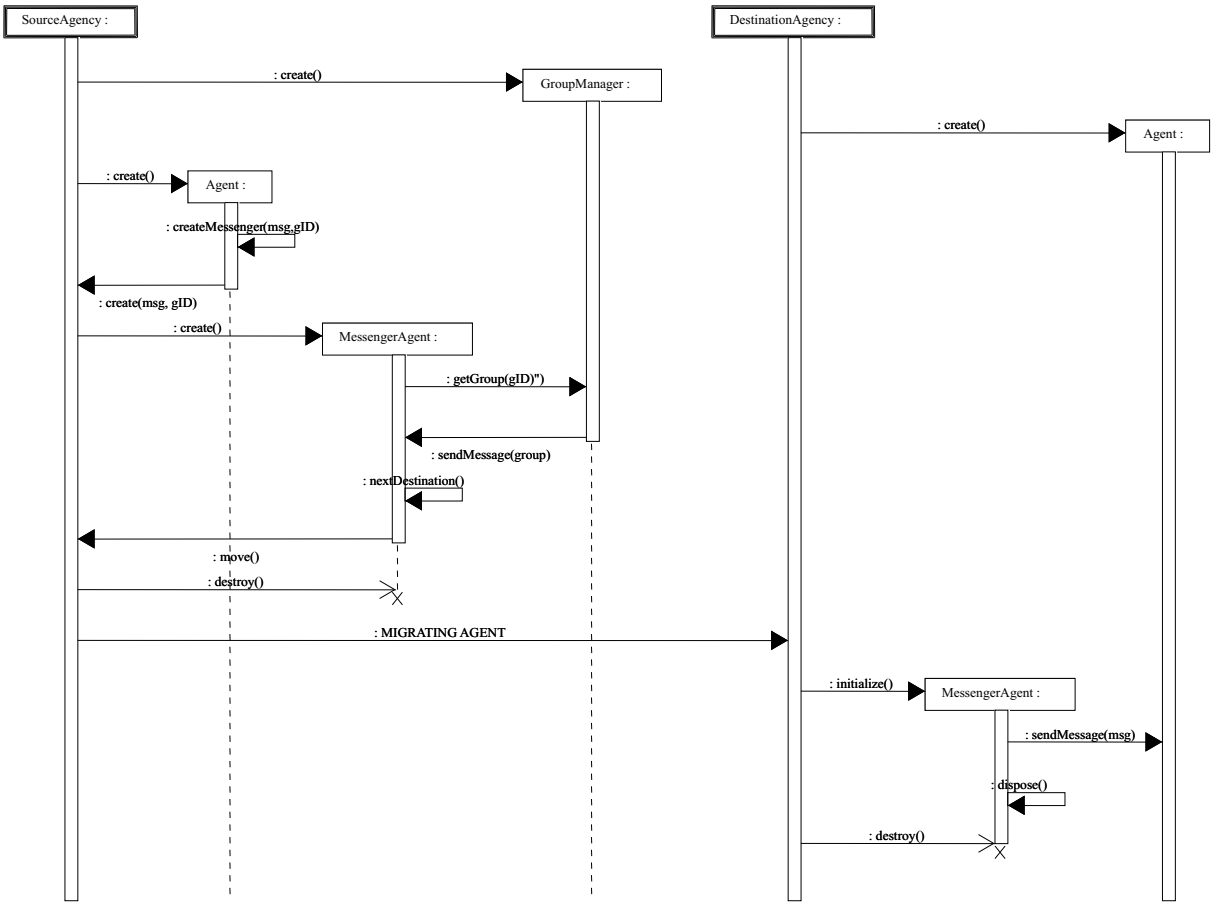


Figura A.36: Diagrama de seqüências independente para o padrão *Group Communication*.

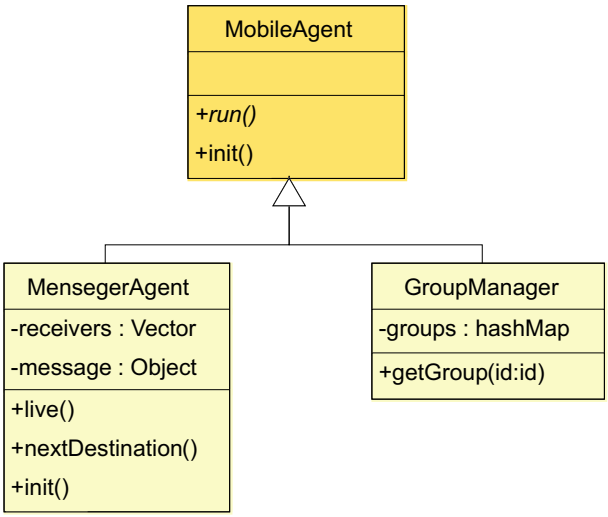


Figura A.37: Diagrama de classes para o padrão *Group Communication* na plataforma Aglets.

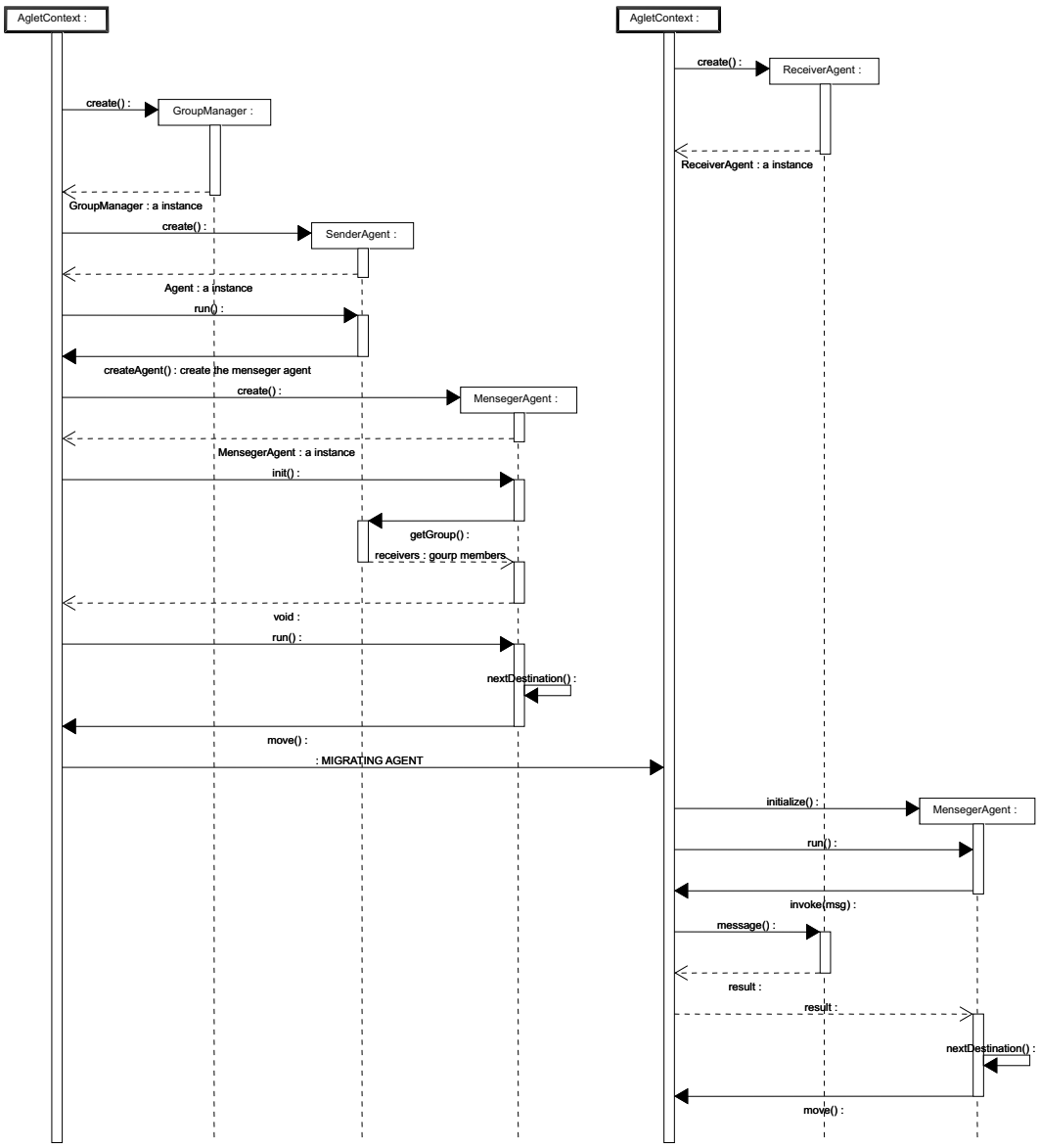


Figura A.38: Diagrama de seqüências para o padrão *Group Communication* na plataforma Aglets.

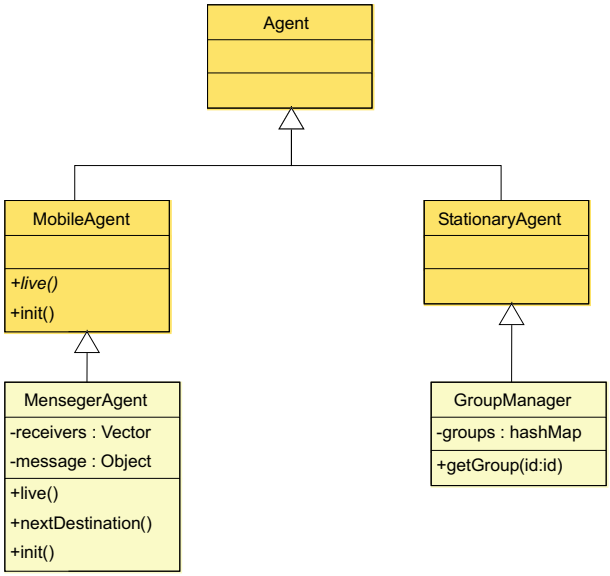


Figura A.39: Diagrama de classes para o padrão *Group Communication* na plataforma Grasshopper.

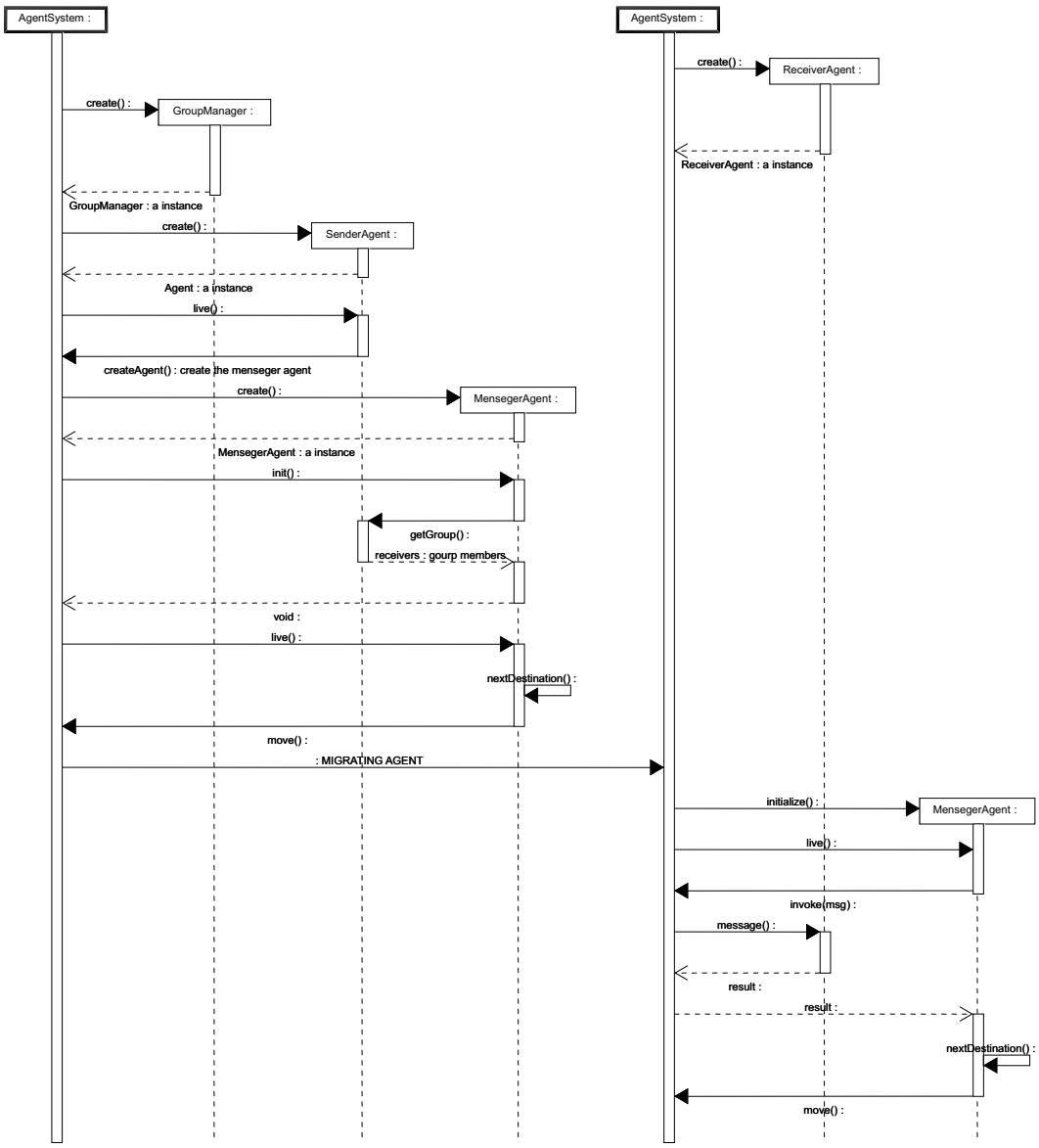


Figura A.40: Diagrama de seqüências para o padrão *Group Communication* na plataforma Grasshopper.

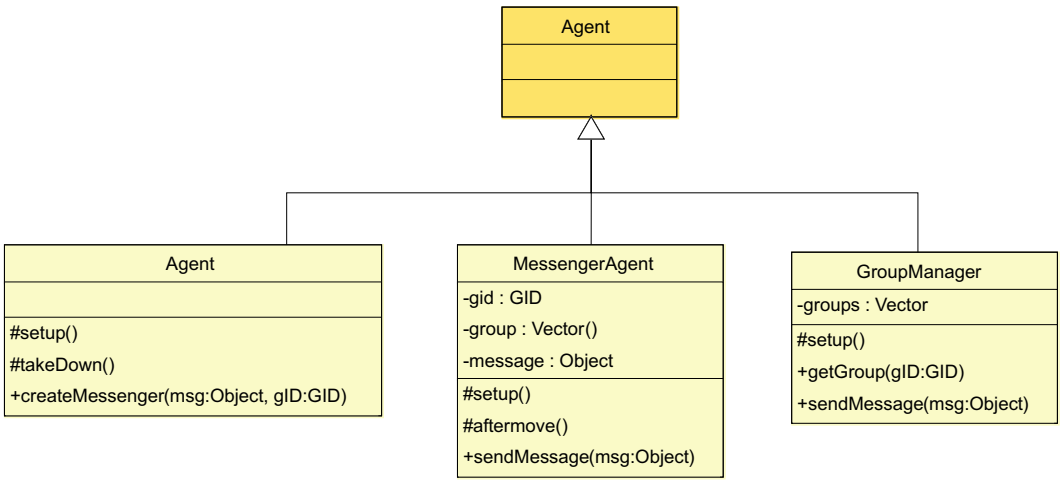


Figura A.41: Diagrama de classes para o padrão *Group Communication* na plataforma JADE.

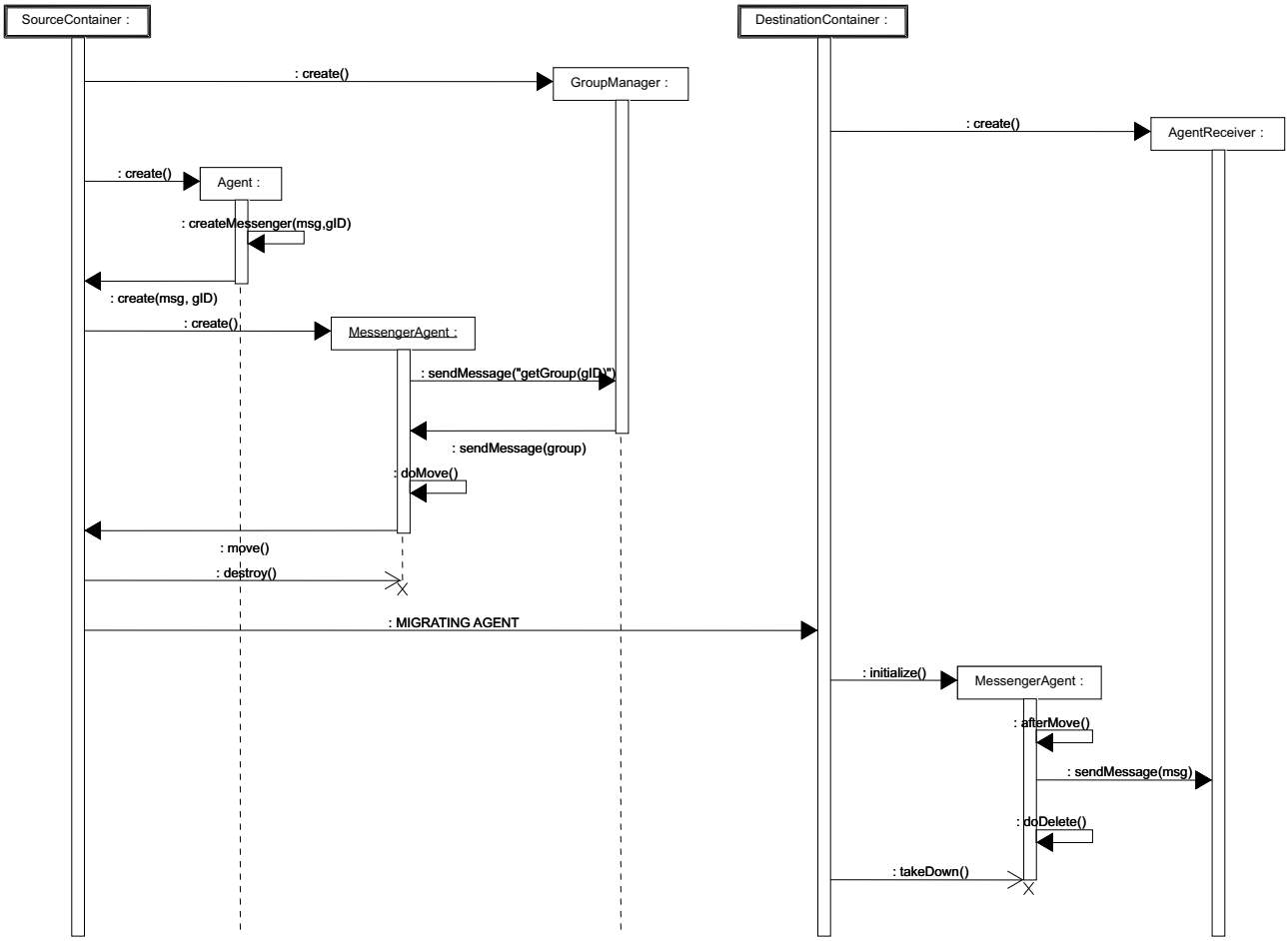


Figura A.42: Diagrama de seqüências para o padrão *Group Communication* na plataforma JADE.

Facilitator

- **Descrição**: O padrão Facilitator define um serviço que provê um serviço de nomes e localização de agentes com habilidades específicas. Proposto em [AL98].
- **Motivação**: Este padrão facilita a identificação e acesso aos agentes e a serviços específicos.
- **Aplicabilidade**: Sistemas muito grandes, em que a localização de diferentes agentes, ou de agentes que disponibilizem serviços específicos, como por exemplo, um agente do tipo mestre.
- **Conseqüências**: É necessário manter um serviço atualizado sobre os agentes disponíveis, o tipo de serviço e a localização de cada um.
- **Estrutura e Implementação**

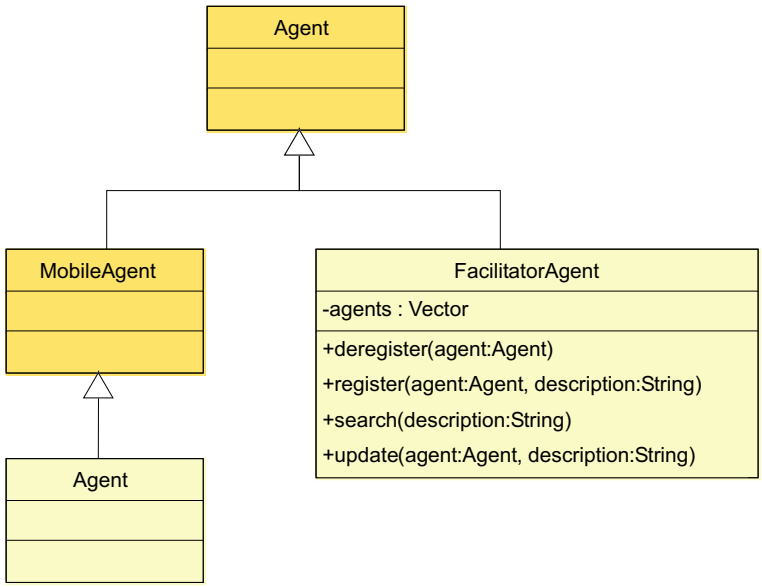


Figura A.43: Diagrama de classes independente para o padrão *Facilitator*.

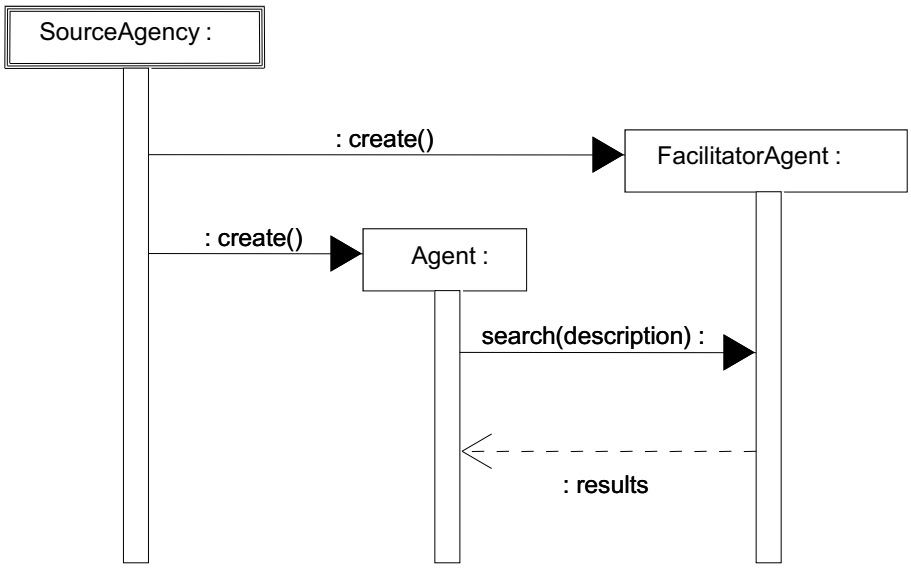


Figura A.44: Diagrama de seqüências independente para o padrão *Facilitator*.

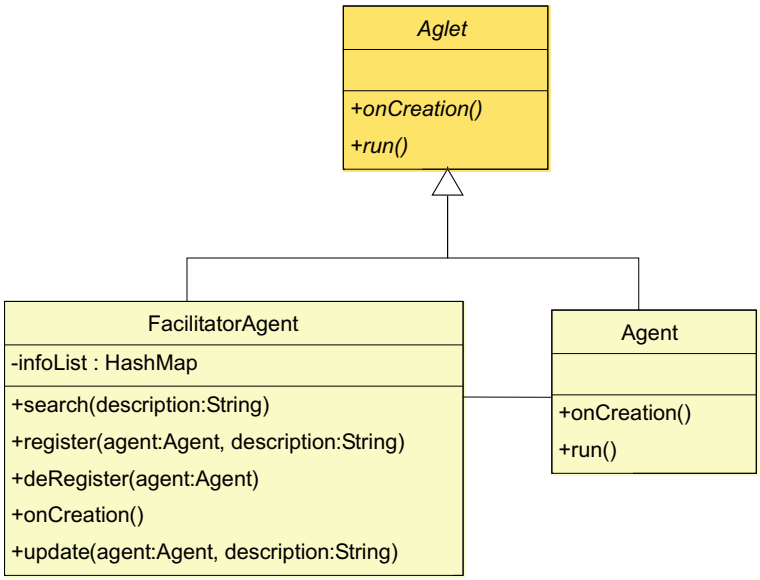


Figura A.45: Diagrama de classes para o padrão *Facilitator* na plataforma Aglets.

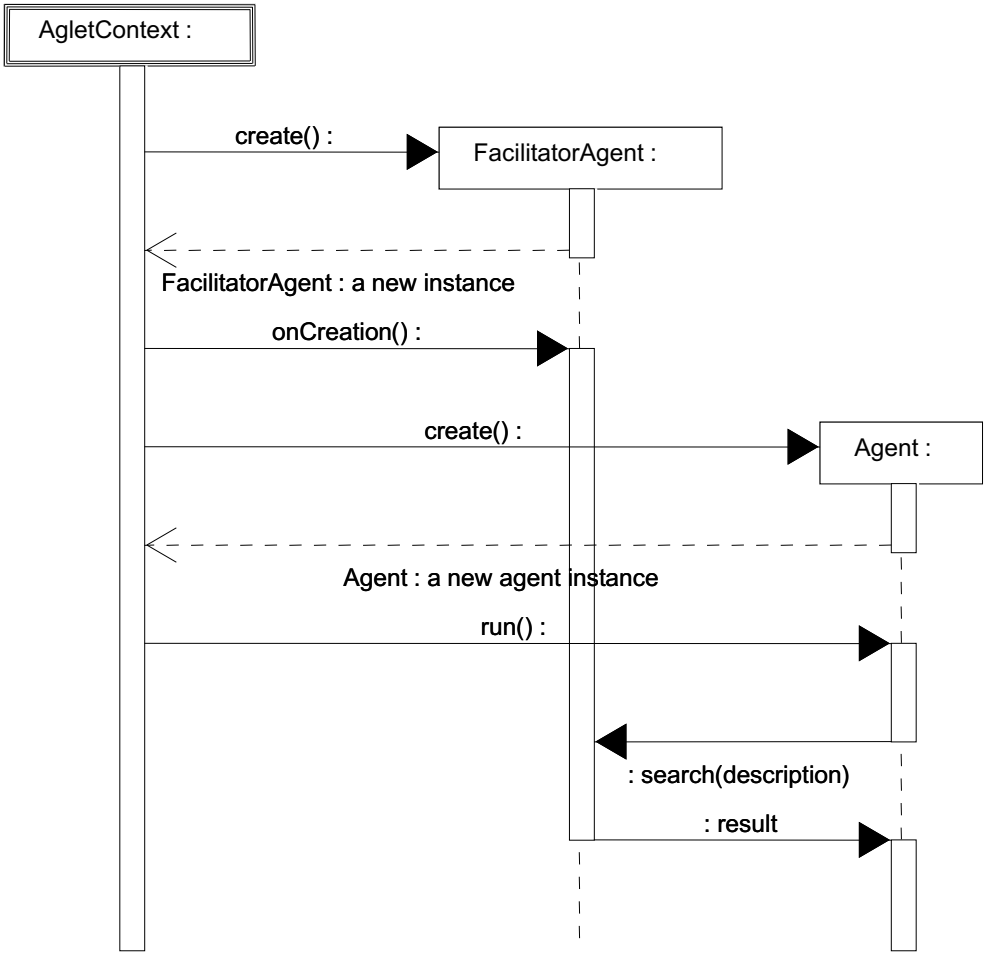


Figura A.46: Diagrama de seqüências para o padrão *Facilitator* na plataforma Aglets.

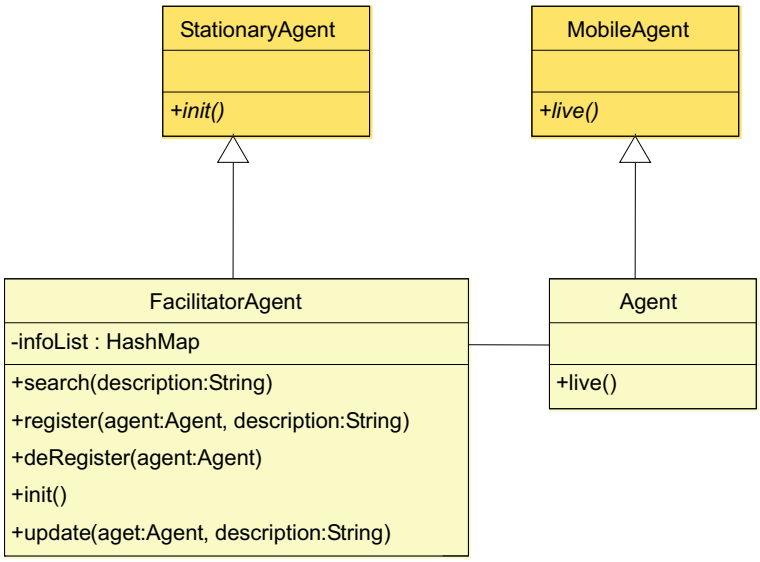


Figura A.47: Diagrama de classes para o padrão *Facilitator* na plataforma Grasshopper.

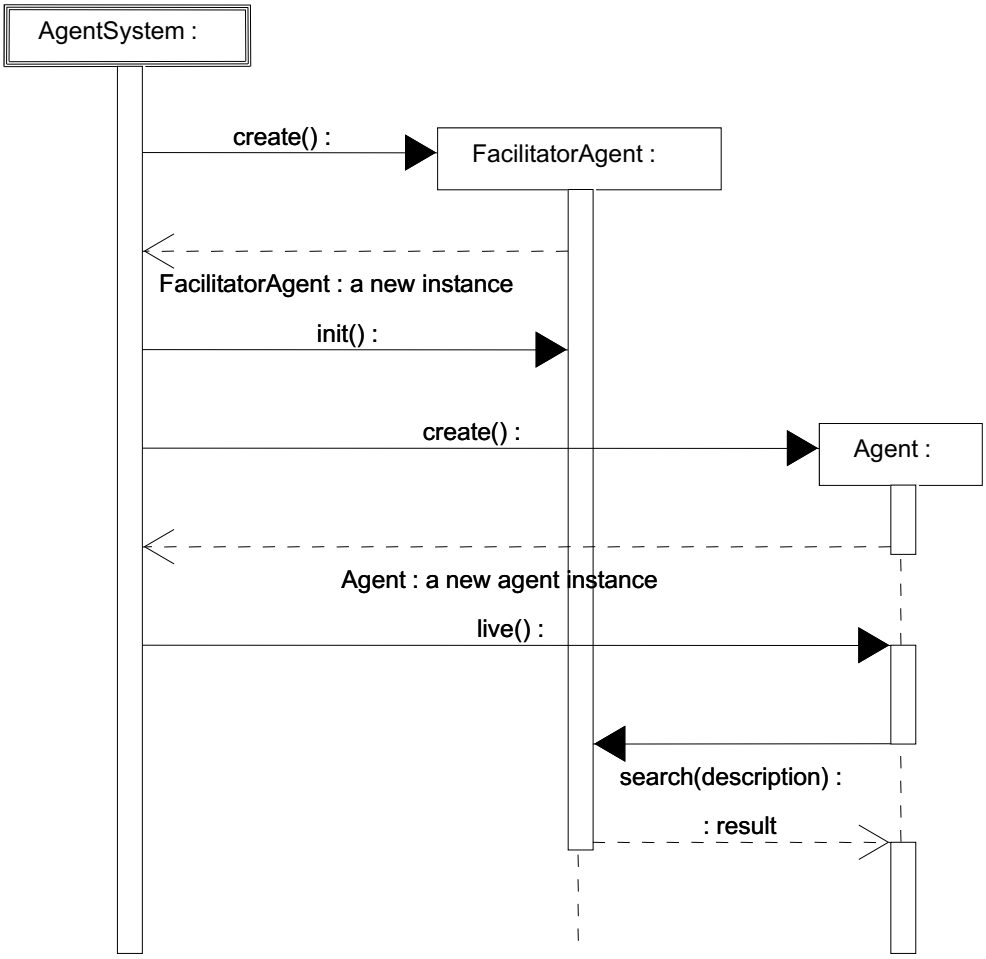


Figura A.48: Diagrama de seqüências para o padrão *Facilitator* na plataforma Grasshopper.

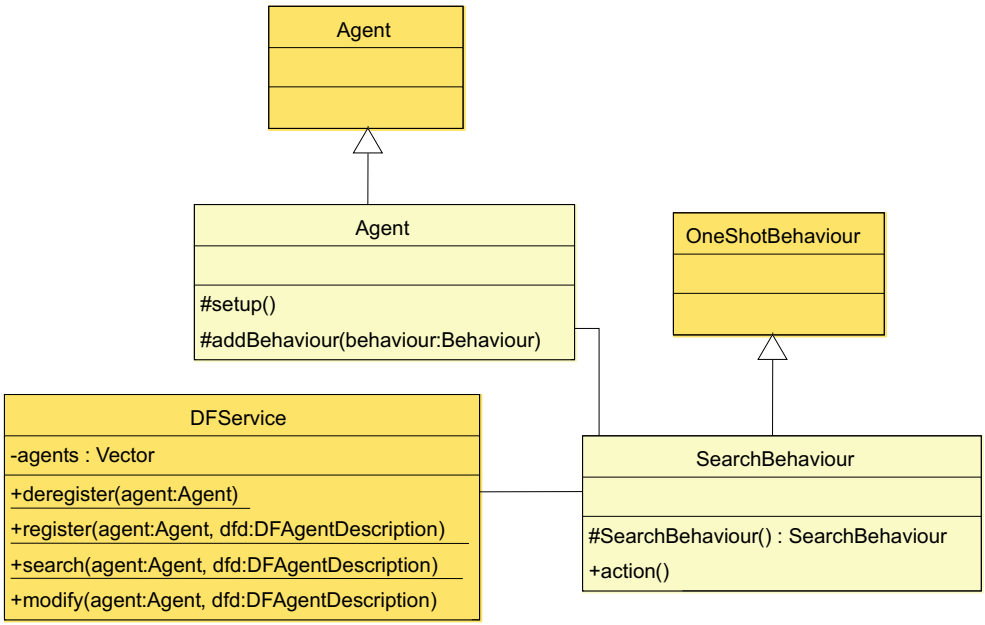


Figura A.49: Diagrama de classes para o padrão *Facilitator* na plataforma JADE.

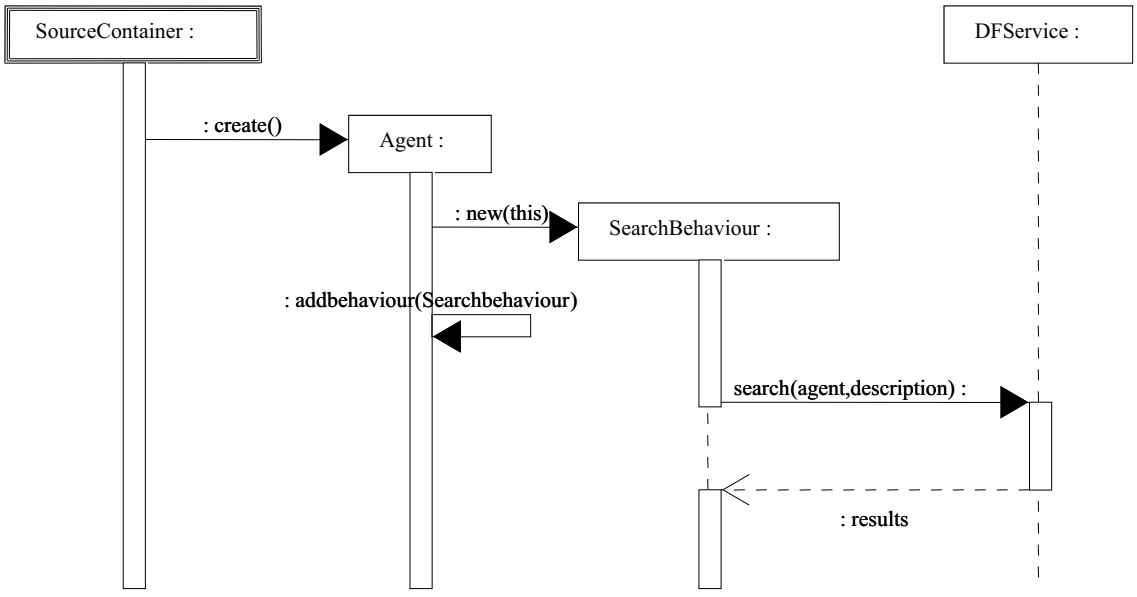


Figura A.50: Diagrama de seqüências para o padrão *Facilitator* na plataforma JADE.

A.4 Segurança

- P^s[TTOH01];
- *Sandbox*[JK99];
- *Path Histories*[JK99];
- *Partial Result Encapsulation*[JK99];
- *Mutual Itinerary Recording*[JK99];
- *Execution Tracing*[JK99];
- *Cryptography*[Gmb01];
- *Authentication*[Gmb01];
- *Access Control*[Gmb01].

A.5 Arquitetura

- *Agent Pattern*[AD98]

A.6 Recurso

- *Broker*[KKPS98].

MoProxy

- **Descrição**: Neste padrão, quando um agente precisa de um recurso, ele requisita para o concessor de recursos, indicando as permissões desejadas. Então, o concessor de recursos retorna um proxy móvel para o agente, de forma que ele acesse o recurso com as permissões desejadas, mas de acordo com as restrições do recurso. Proposto em [SSJ].

- **Motivação**: É uma forma segura de disponibilizar o acesso e utilização de recursos, através da qual se poderá definir restrições, evitando-se que os recursos sejam usados sem controle.
- **Aplicabilidade**: Sistemas que disponibilizem recursos, como banco de dado, impressoras ou memória, e que desejem controlar o acesso a estes.
- **Conseqüências**: Um nível de indireção é criado entre o agente e o recurso solicitado, podendo tornar o acesso e utilização mais lento.
- **Padrões relacionados**: *Broker*
- **Estrutura e Implementação**

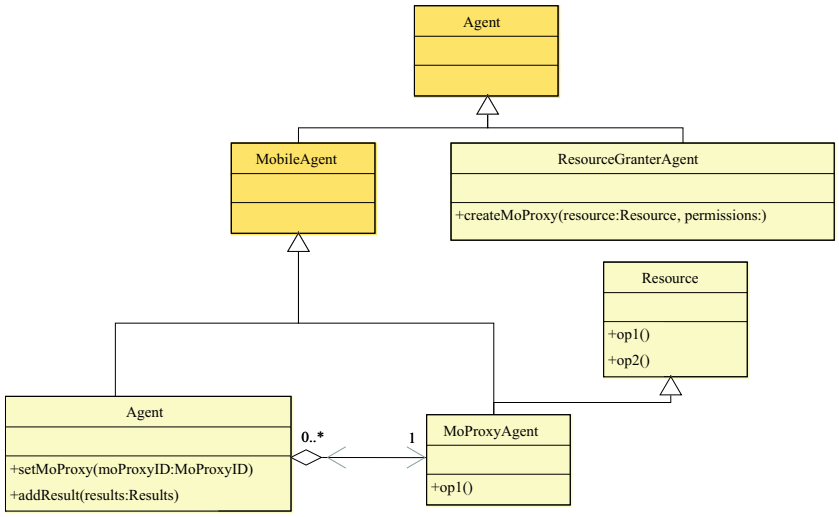


Figura A.51: Diagrama de classes independente para o padrão *MoProxy*.

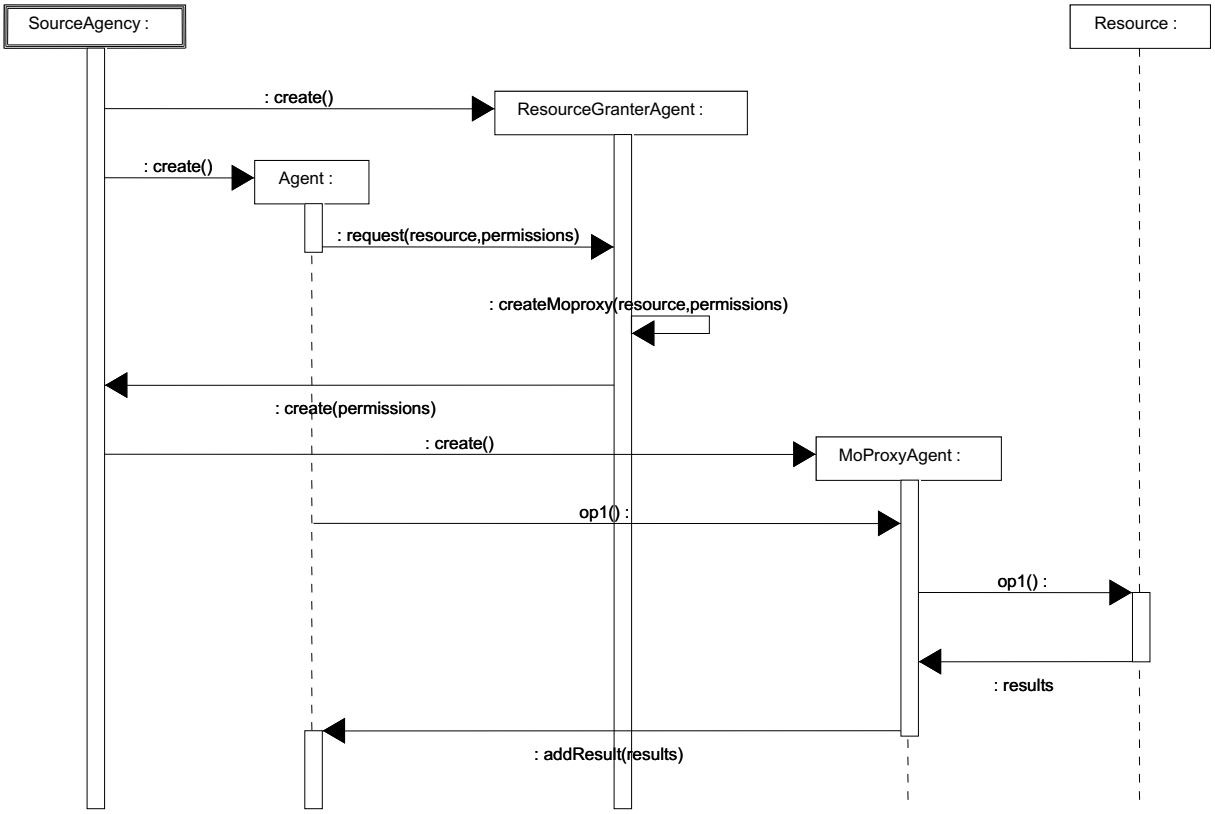


Figura A.52: Diagrama de seqüências independente para o padrão *MoProxy*.

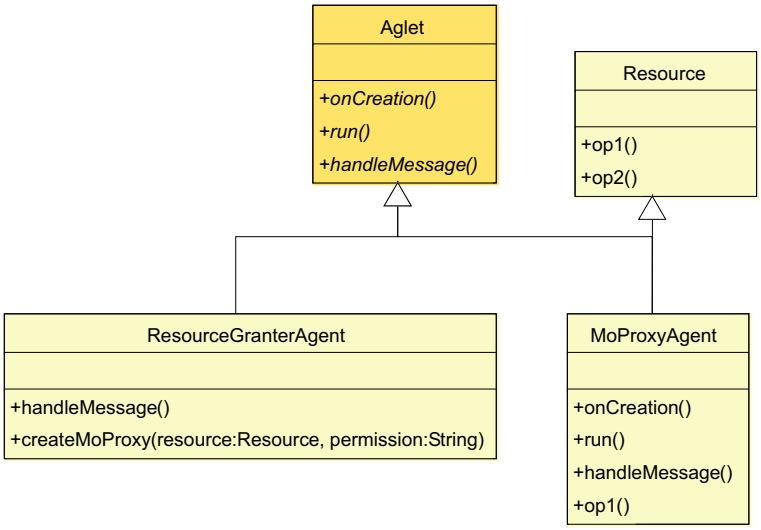


Figura A.53: Diagrama de classes para o padrão *MoProxy* na plataforma Aglets.

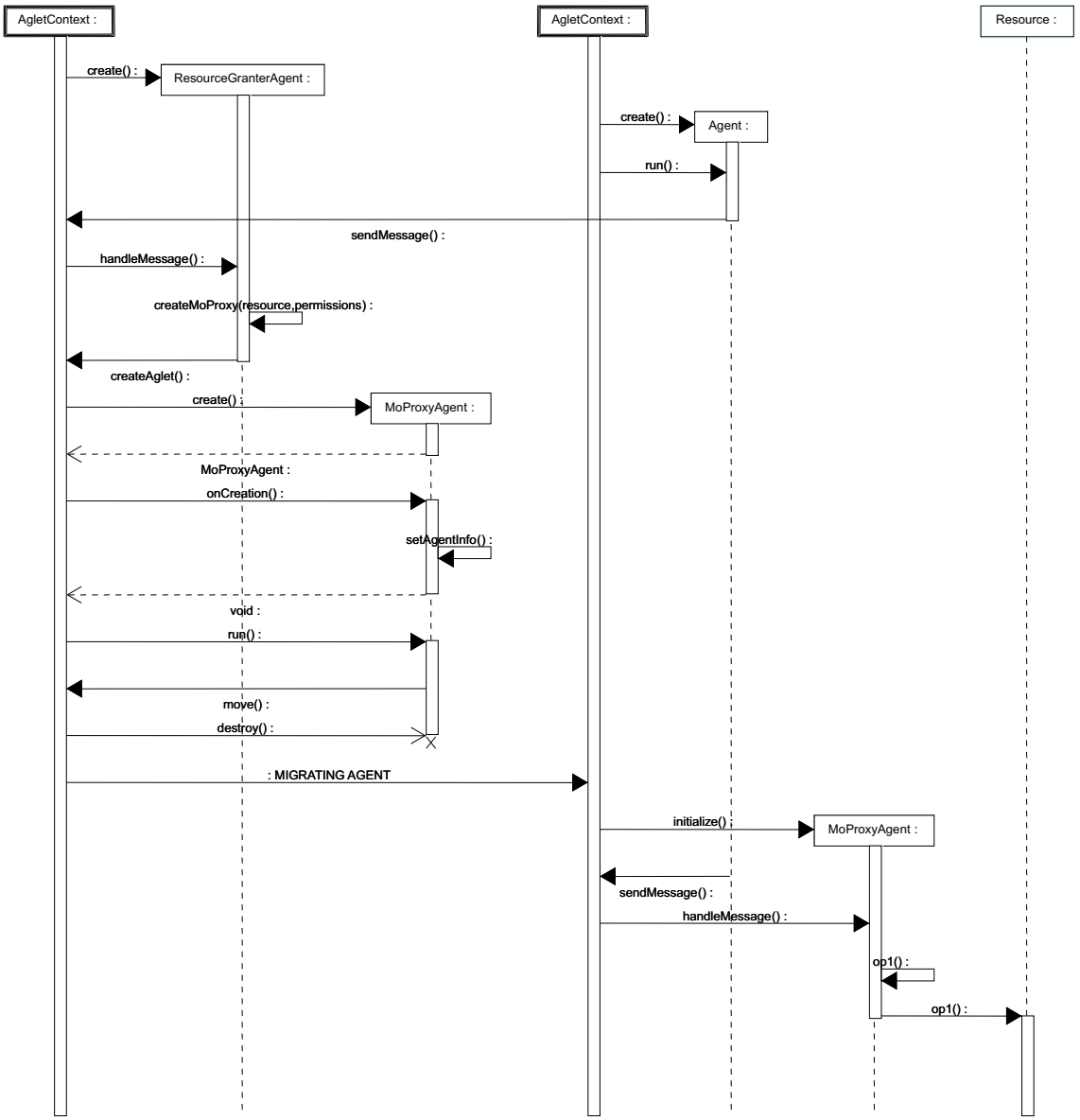


Figura A.54: Diagrama de seqüências para o padrão *MoProxy* na plataforma Aglets.

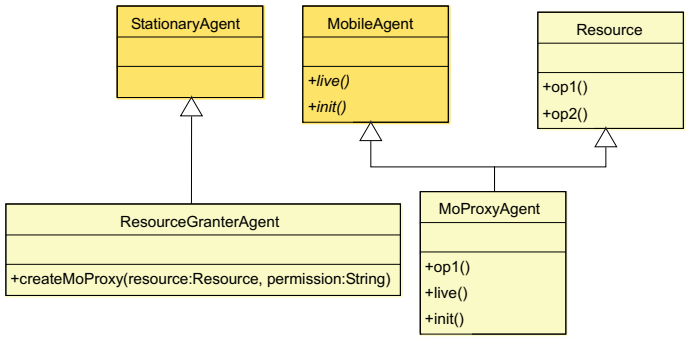


Figura A.55: Diagrama de classes para o padrão *MoProxy* na plataforma Grasshopper.

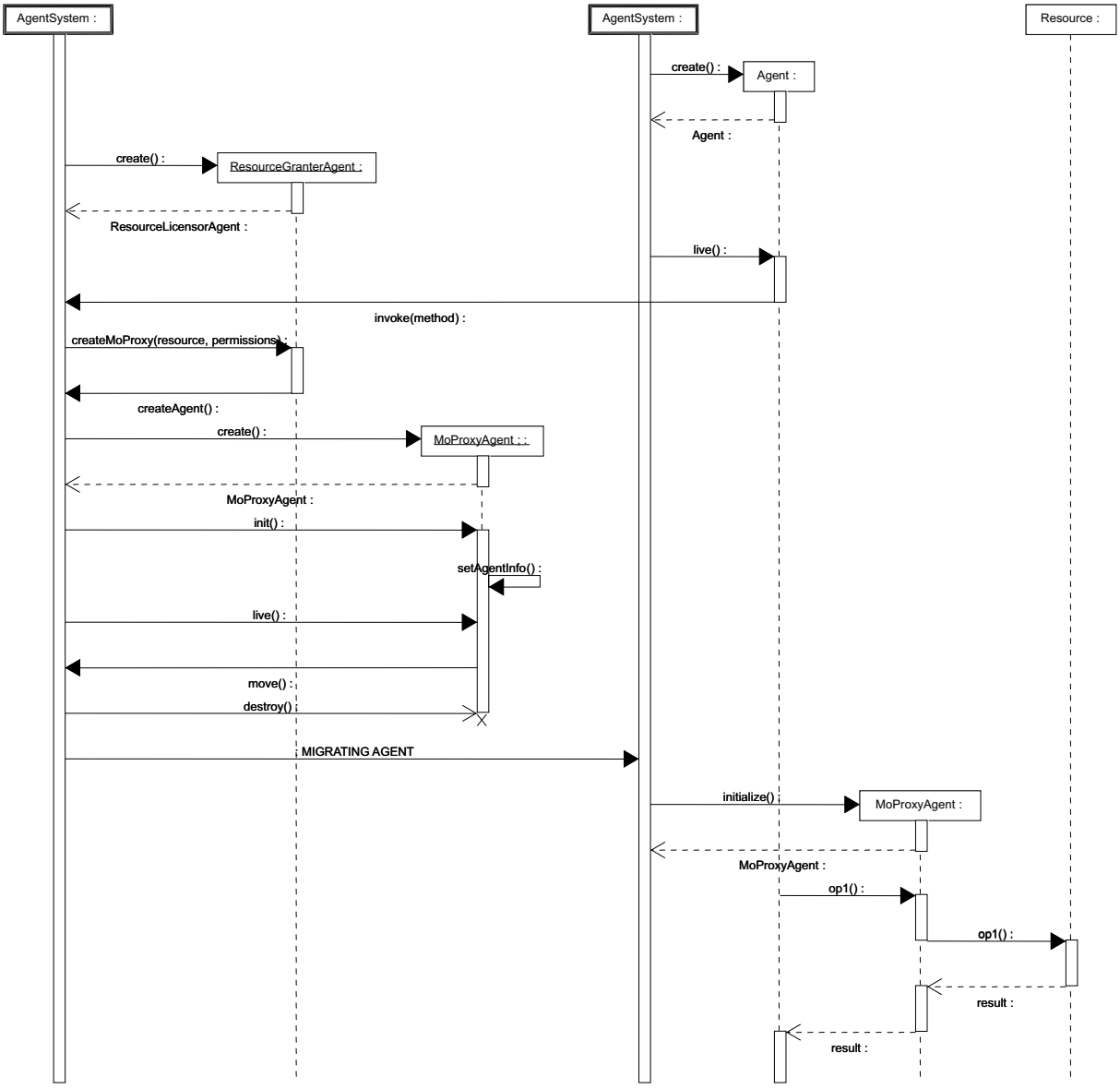


Figura A.56: Diagrama de seqüências para o padrão *MoProxy* na plataforma Grasshopper.

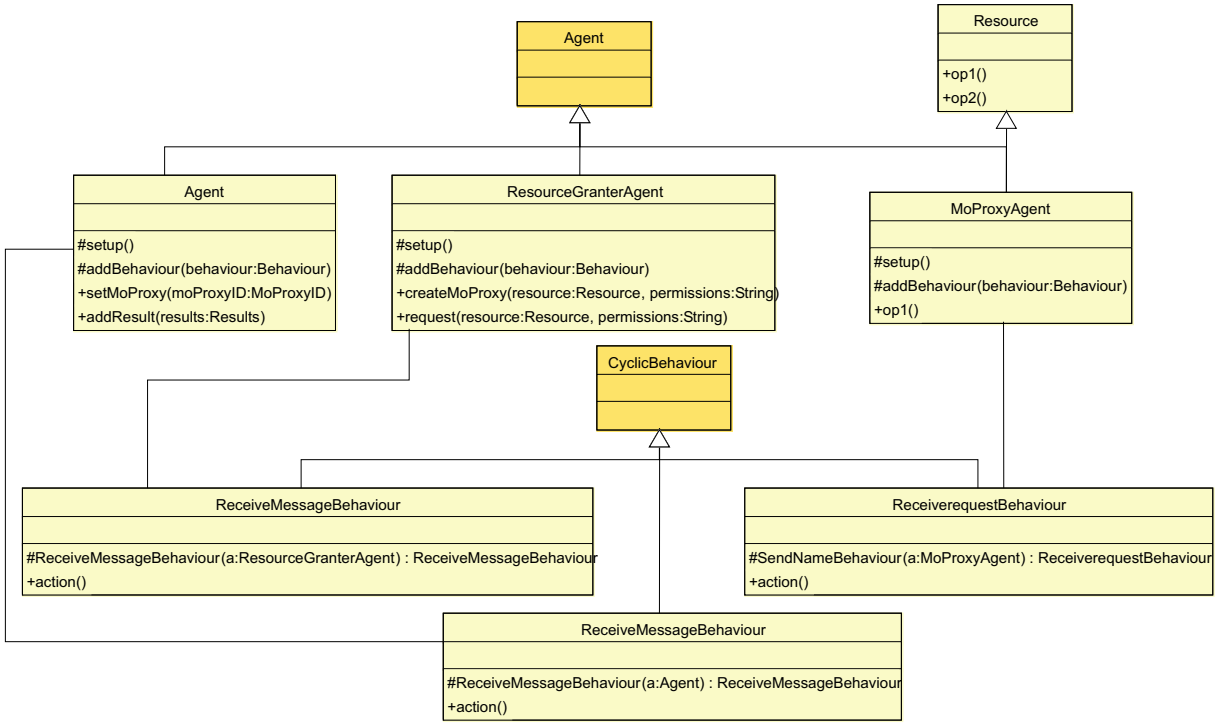


Figura A.57: Diagrama de classes para o padrão *MoProxy* na plataforma JADE.

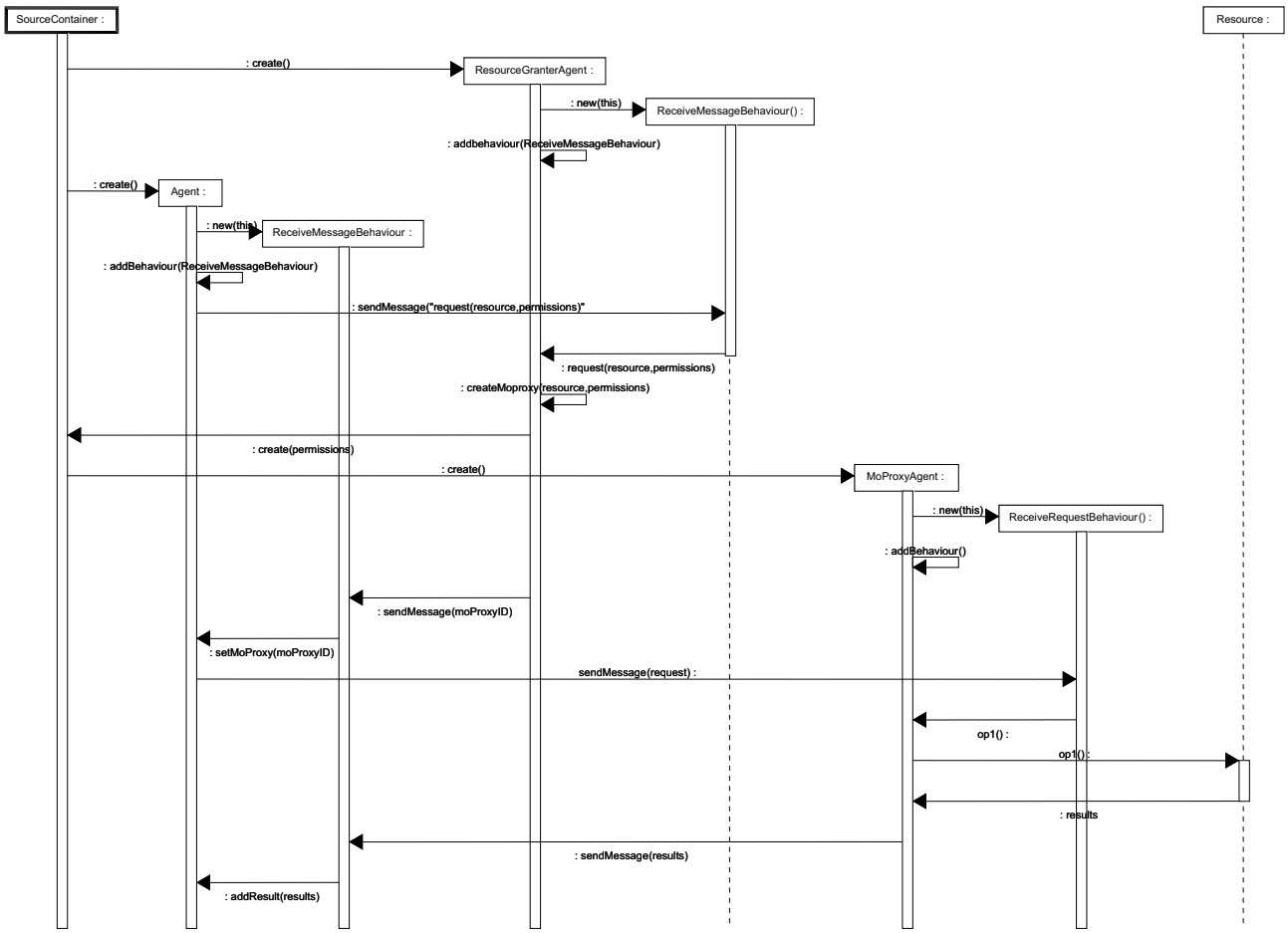


Figura A.58: Diagrama de seqüências para o padrão *MoProxy* na plataforma JADE.

Apêndice B

Formalização de Padrões de Projeto para Agentes Móveis

B.1 Padrão *Master-Slave*

O modelo principal do padrão *Master-Slave*, assim como o modelo da rede e da agência de destino são similares aos modelos apresentados nas Figuras 4.7, 4.8 e 4.9. No modelo da agência de origem (Figura B.1), é representada a criação do agente escravo pelo mestre (transição *CreateSlave*). O agente mestre tem uma lista de agências que os agentes escravos devem visitar. Sempre que um agente escravo é criado (função *createSlave(masterAgent*), ele recebe uma agência para visitar, e esta agência é retirada da lista do agente mestre (*removeFisrtAgency(masterAgent)*). Note que a transição *CreateSlave* irá disparar enquanto existirem agências na lista do agente mestre (guarda *[itinerarySize(masterAgent)>0]*). Após sua criação, o escravo migra para a agência indicada pelo mestre, onde executa a tarefa.

Quando o agente escravo retorna para a agência de origem (transição *ReceiveAgent*), ele envia os resultados da tarefa para o agente mestre (transição *ReceiveResult*), e se destrói. O agente mestre recebe o resultado (função *addData(masterAgent,getData(agent))*), e espera pelos outros agentes escravos. Note que o agente somente é recebido pela agência, isto é, a transição *ReceiveAgent* somente dispara, se o estado do agente for *executing*, e sua localização atual for a agência de origem (guarda *[(isExecuting(agent)), isSource(agent)]*).

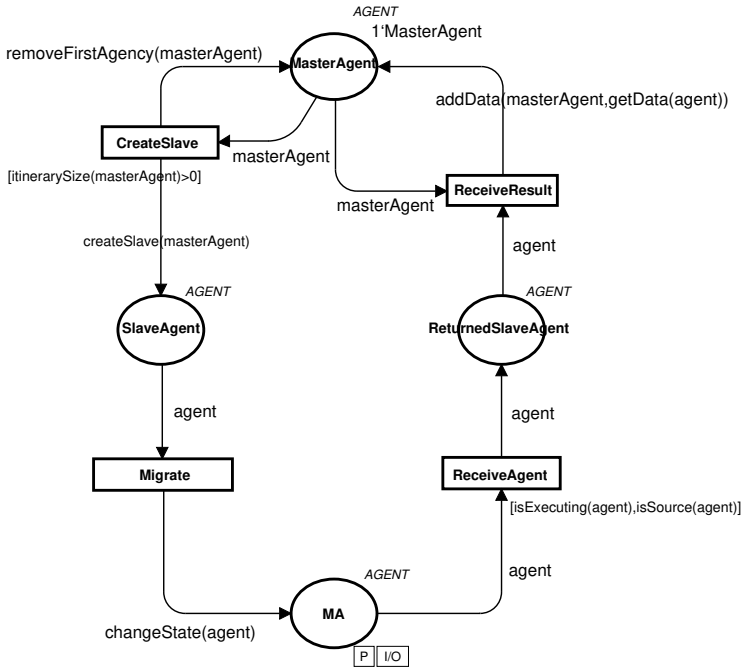


Figura B.1: Modelo da agência de origem para o padrão *Master-Slave*

B.2 Padrão *Meeting*

A estrutura principal para o modelo do padrão *Meeting* é similar ao mostrado na Figura 4.7. No modelo da agência de origem (Figura B.2), existe um agente (lugar *Agent*) e seu objeto *meeting* (lugar *Meeting*). Quando o agente vai migrar, ele solicita o local do encontro ao objeto *meeting* (transição *GetMeetingPlace*), e migra para o encontro juntamente com o seu objeto *meeting*. A função *getMeetingPlace(agent,meeting)* define o destino do agente, e a função *changeState(agent)* altera o estado do agente de *executing* para *migrating*. Note que a guarda *[match(agent,meeting]* garante que o agente somente interage com o seu objeto *meeting*.

O modelo para a agência de destino (Figura B.3 representa o local do encontro. Sempre que um agente chega à este local (transição *arrive*), ele solicita ao objeto *meeting* para ser registrado no encontro (transição *ready*), e espera pela confirmação no lugar *MeetAgtReady*. Então, o objeto *meeting* registra o agente no gerenciador de encontros (*Meeting Manager*) e, após o registro, o agente está habilitado para trocar mensagens (lugar *at_Meeting*). Note que o registro do agente é realizado pela transição de substituição *AddAgent*, que é detalhada mais a frente.

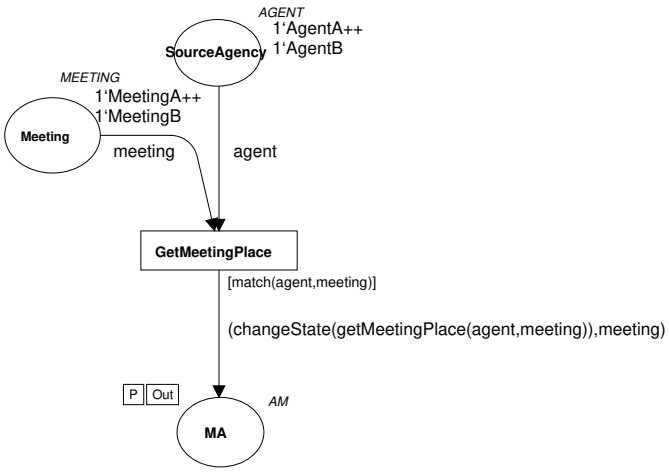


Figura B.2: Modelo da agência de origem para o padrão *Meeting*

Estando no encontro, o agente pode trocar mensagens com outros agentes no encontro (substitution transition *Messages*), ou sair do encontro. Quando deseja sair do encontro, o agente solicita ao objeto *meeting* para ser descadastrado do encontro (transição *leave*, e espera pela confirmação no lugar *MeetingAgentMigrate*. Então, o objeto *meeting* descadastra o agente do encontro (transição de substituição *deleteAgent*), e está pronto para migrar com o agente. Note que o lugar *Meeting* e *Meeting_* (lugares de fusão - rotulados com um FG) são os mesmos lugares, duplicados por questões de desenho.

No modelo que detalha a transição de substituição *AddAgent* (Figura B.4) é mostrado o registro do agente no encontro. Na transição *meet*, o agente recebe, do gerenciador do encontro, a lista dos agentes que já estão no encontro (função *setIDList(agent,meetmng)*). Então, o gerenciador do encontro notifica todos os agentes no encontro da chegada de um novo agente. Esta notificação é realizada pela transição *meetWith*, e cada agente notificado adiciona o identificador do novo agente em sua lista (função *addId(agent,id_new)*). Após notificar todos os agentes, o gerenciador do encontro retorna para o lugar *MeetingManager* (transição *finishMW*).

No modelo que detalha a transição de substituição *DeleteAgent* (Figura B.5), o agente que está saindo do encontro é removido da lista do gerenciador do encontro (transição *deleteAgent* - função *removeID(meetmng,getID(meeting))*. Então, todos os agente são notificados da saída do agente (transition *notifyDelete*).

O modelo que detalha a transição de substituição *Messages* (Figura B.6) é muito depen-

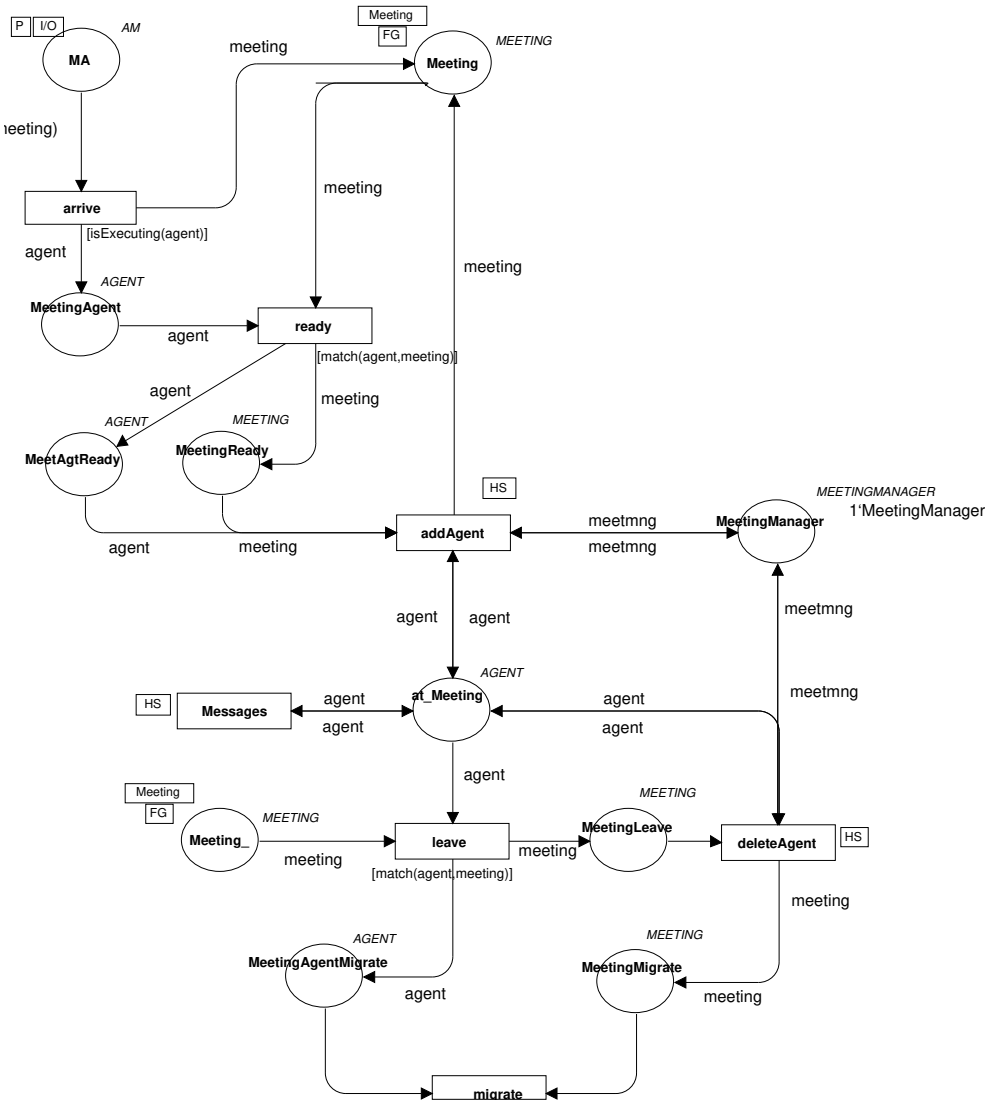


Figura B.3: Modelo da agência de destino para o padrão *Meeting*

dente da aplicação que irá utilizar este padrão. No modelo apresentado, temos uma representação de uma possível troca de mensagens entre os agentes no encontro, em que um agente envia, aleatoriamente, uma mensagem para algum agente de sua lista (transição *sendMessage*). Esta mensagem é recebida pelo agente destinatário (transição *ReceiveMessage*).

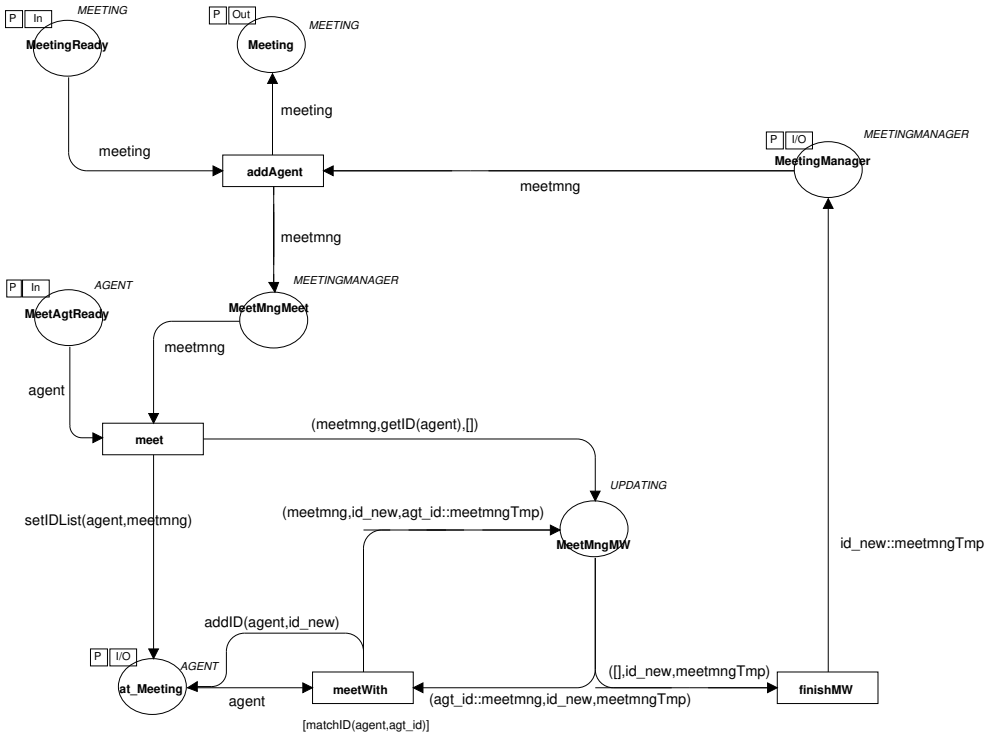


Figura B.4: Modelo para registro de agentes no encontro do padrão *Meeting*

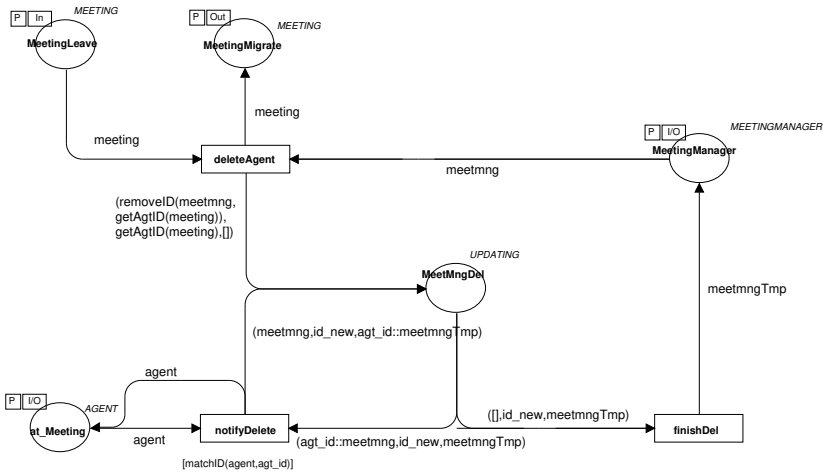


Figura B.5: Modelo para descadastramento de agentes no encontro do padrão *Meeting*

B.3 Padrão *Group Communication*

No padrão *Group Communication*, a estrutura do modelo é a mesma mostrada na Figura 4.7. No modelo da agência de origem (Figura B.7), o agente *messenger* é criado (*CreateMessenger*). Em sua criação, este agente recebe o identificador do grupo e a mensagem para enviar aos membros do grupo (função *createMsn(agent)*). Após isto, o agente *mes-*

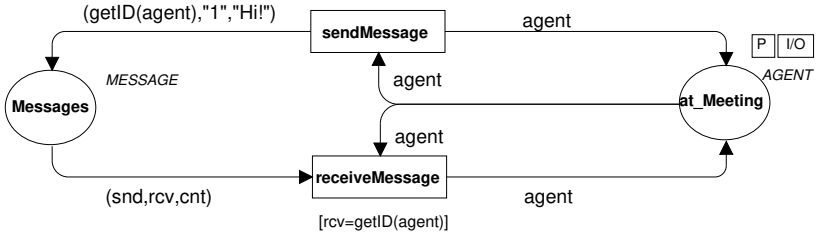


Figura B.6: Modelo da troca de mensagens para o padrão *Meeting*

senger recupera o grupo do diretório de grupos *Group Directory* de acordo com o identificador (transition *RetrieveGroup*), e os membros do grupo são adicionados ao agente (função *getGroup(msnagent,gd)*). Tendo o grupo, o agente *messenger* migra para para as agências de cada agente no grupo. O modelo da rede é similar ao da Figura 4.8.

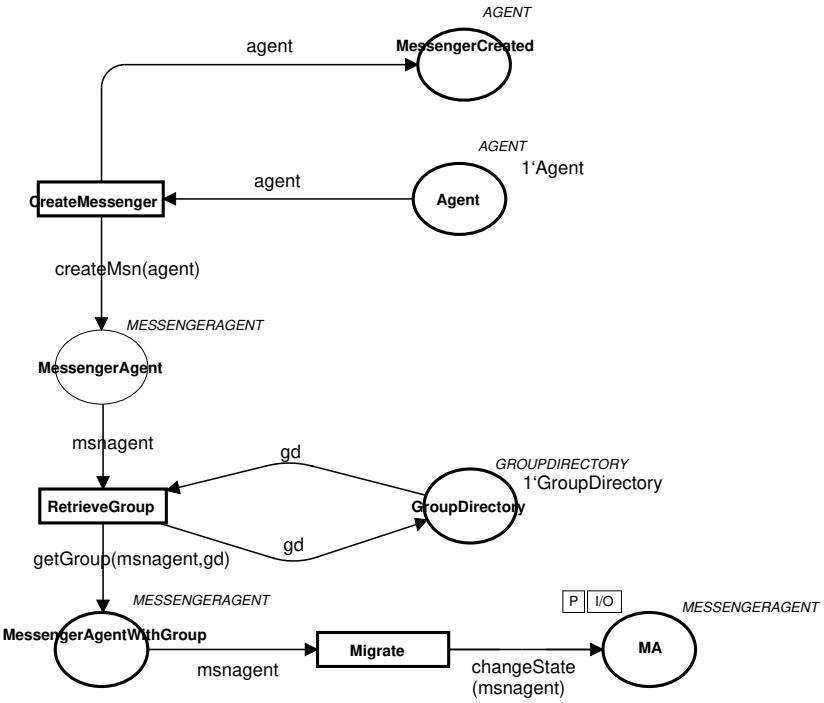


Figura B.7: Modelo da agência de origem para o padrão *Group Communication*

Após chegar à agência de destino (Figura B.8, transição *Arrive*), o agente *messenger* envia a mensagem para o agente do grupo que estiver naquela agência (transição *SendMessage*). A guarda *[checkReceiver(msnagent,agtRcv)]* garante que a mensagem será enviada somente para o agente correto. Após enviar a mensagem, o agente *messenger* migra para a agência do próximo agente no grupo (transição *Migrate*). Se não houver mais agentes no grupo, o agente *messenger* se destrói (transição *Destroy*).

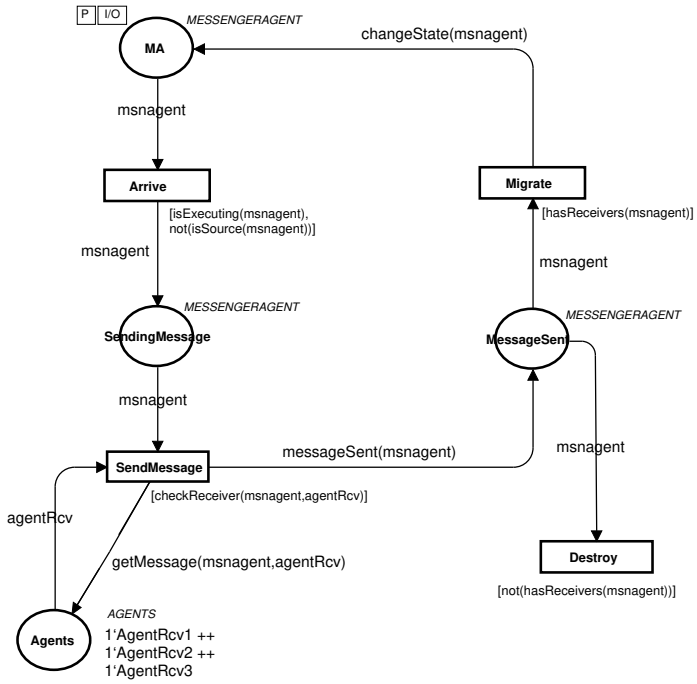


Figura B.8: Modelo da agência de destino para o padrão *Group Communication*

B.4 Padrão *Facilitator*

No modelo para o padrão *Facilitator* (Figure B.9), existe um lugar representando o *Facilitator* e um outro representando as mensagens do sistema. O *Facilitator* armazena registros contendo identificador, tipo e localização dos agentes. Uma mensagem contém um remetente, um destinatário, um tipo e um conteúdo. Os agentes podem interagir com o *Facilitator* através de mensagens, e as operações disponíveis são representadas pelas quatro transições mostradas no modelo.

Cada transição têm uma guarda indicando o tipo de mensagem que deverá processar. Na primeira (*Search*), o conteúdo da mensagem indica o identificador, localização e/ou tipo dos agentes procurados. Se um agente deseja procurar agentes do tipo "mestre"em qualquer agência, por exemplo, então o conteúdo da mensagem será ("*","mestre","*"). Quando esta transição dispara, a função *search(message,registers)* cria uma mensagem contendo os registros que casem com o critério de busca.

Na segunda transição (*Register*), o conteúdo da mensagem é um registro a ser adicionado no *Facilitator*. A função *addRegister(message,register)* adiciona um novo registro na lista do *Facilitator*.

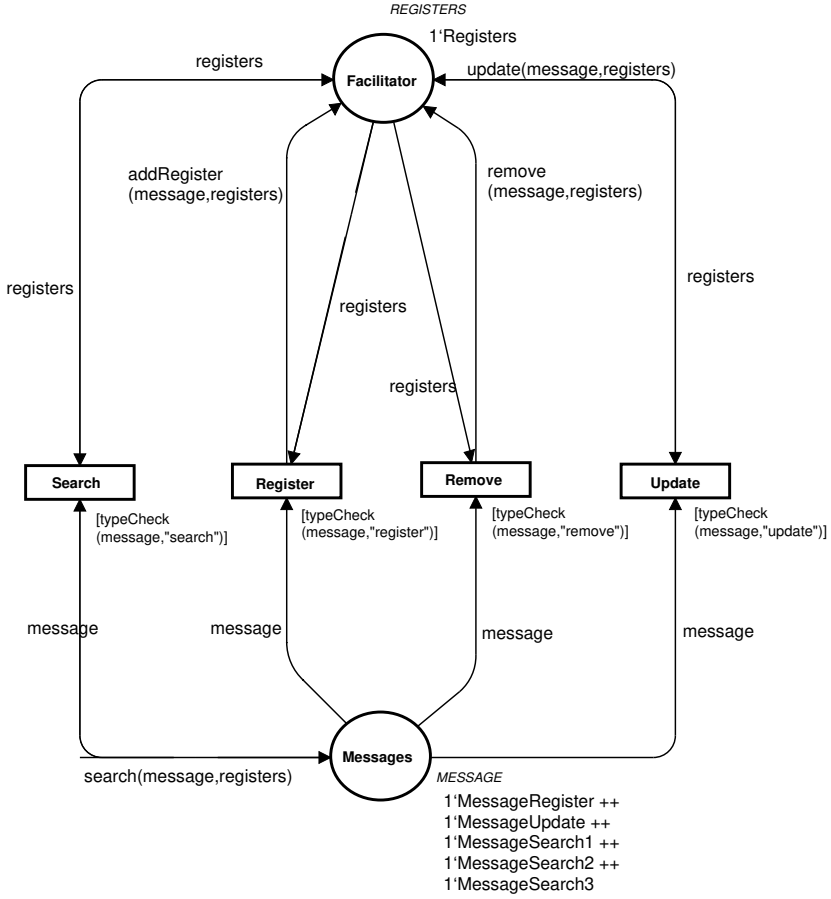


Figura B.9: Modelo para o padrão *Facilitator*

Na transição *Remove*, a função *remove(message,register)* remove o registro, cujo identificador for igual ao contido na mensagem. Se não existir registro satisfazendo esta condição, nada acontece aos registros do *Facilitator*.

Na última transição (*Update*), a função *update(message,register)* substitui o registro do *Facilitator*, cujo identificador for igual ao contido na mensagem, pelo registro contido na mensagem. Se nenhum identificador for igual, então nada acontece aos registros do *Facilitator*.

B.5 Padrão *MoProxy*

No modelo para o padrão *MoProxy* (Figure B.10), existe um agente (lugar *Agent*) que quer acessar um recurso (lugar *Resource*). Para obter este acesso, o agente solicita ao concessor de recursos (*Resource Granter*) a criação de um *MoProxy* (transição *CreateMoProxy*).

Quando esta transição dispara, o agente *MoProxy* é criado no lugar *MoProxy*, sem requisições e respostas (*([],[])*). O agente possui uma variável indicando se a criação do agente *MoProxy* já foi solicitada. Esta variável tem seu valor mudado de "y"para "n"(função *changeMoProxy(agent)*), e a transição *CreateMoProxy* somente dispara se o valor da variável for "y"(guarda *[moProxy(agent)=y]*).

O agente tem uma lista de requisições para pedir ao recurso. Então, o agente deve interagir com o *MoProxy* para poder acessar este recurso. Para isto, o agente envia uma requisição para o *MoProxy* (transição *Request*), e retira esta requisição de sua lista (função *removeRequestAgt(agent)*). Ao mesmo tempo, o *MoProxy* adiciona a requisição à sua lista (função *addRequestMP(moproxy, getRequestAgt(agent))*). Note que a transição *Request*, somente dispara se o agente tiver, pelo menos, uma requisição em sua lista (guarda *[hasRequestAgt(agent)]*).

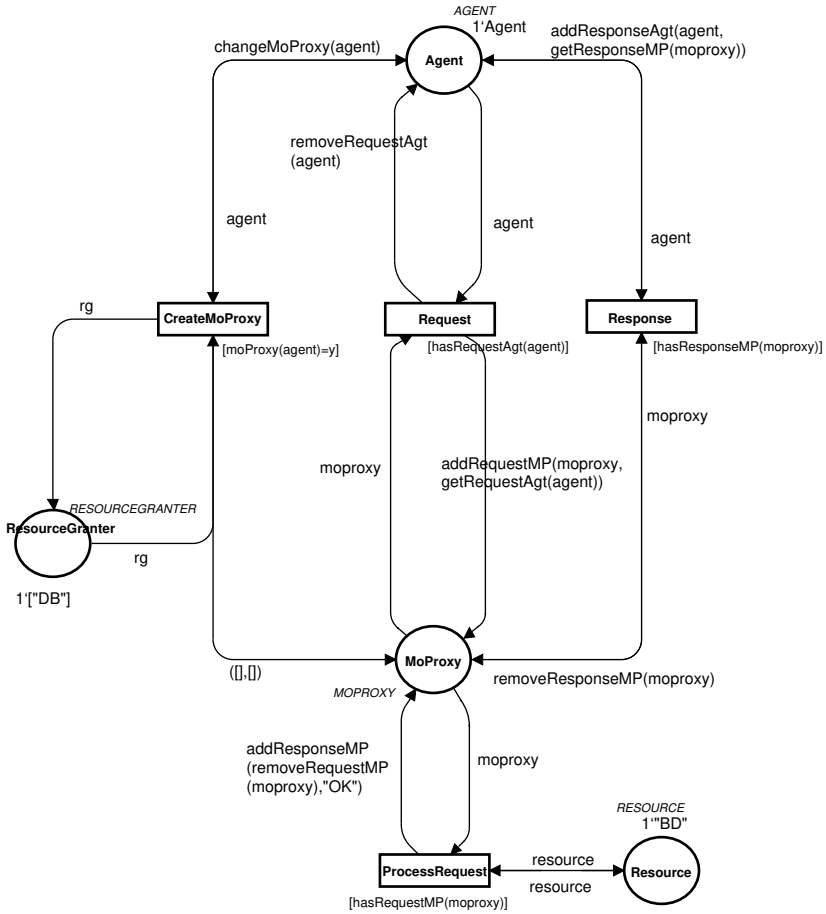


Figura B.10: Modelo para o padrão *MoProxy*

Tendo a requisição, o agente *MoProxy* pode interagir com o recurso. Quando a transição

ProcessRequest dispara, uma requisição é retirada de sua lista e uma resposta é adicionada (função *addResponseMP(removeRequestMP(moproxy),"OK")*). Note que a transição *ProcessRequest*, somente dispara se o *MoProxy* tiver, pelo menos, uma requisição em sua lista (guarda *[hasRequestMP(moproxy)]*).

Após interagir com o recurso, o *MoProxy* pode enviar a resposta para o agente (transition *Response*). Quando esta transição dispara, o *MoProxy* retira a resposta de sua lista (função *removeResponseMP(moproxy)*), e a resposta é adicionada à lista do agente (função *addResponseAgt(agent,getResponseMP(moproxy))*). Note que a transição *Reponse*, somente dispara se o *MoProxy* tiver, pelo menos, uma resposta em sua lista (guarda *[hasResponseMP(moproxy)]*). *[hasRequestMP(moproxy)]*).