UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Cláudia Filomena Bratficher Dário

Uma Metodologia Unificada para o Desenvolvimento de Sistemas Orientados a Agentes

Campinas, São Paulo

2005

Cláudia Filomena Bratficher Dário

Uma Metodologia Unificada para o Desenvolvimento de Sistemas Orientados a Agentes

Dissertação de Mestrado

Orientador:
Prof. Dr. Ricardo Ribeiro Gudwin

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais da obtenção do título de Mestre em Engenharia Elétrica.
Aprovação em 29/07/2005.

Banca Examinadora

Prof. Dr. Jaelson Freire Brelaz de Castro
CIn, UFPE

Prof. Dr. Ivan Luiz Marques Ricarte
DCA, FEEC, UNICAMP

Prof. Dr. Mario Jino
DCA, FEEC, UNICAMP

Campinas, São Paulo

2005

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

D248m

Dário, Cláudia Filomena Bratficher
Uma metodologia unificada para o desenvolvimento de sistemas orientados a agentes / Cláudia Filomena Bratficher Dário. --Campinas, SP: [s.n.], 2005.

Orientador: Ricardo Ribeiro Gudwin
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Engenharia de software. 2. Metodologia. 3. Agentes inteligentes (Software). I. Gudwin, Ricardo Ribeiro. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Titulo em Inglês: A Unified Methodology for the Development of Agent-Oriented Systems.
Palavras-chave em Inglês: Agent, Software engineering, agent oriented software engineering, AUML, MaSE, Prometheus, Tropos
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Jaelson Freire Brelaz de Castro, Ivan Luiz Marques Ricarte e Mario Jino
Data da defesa: 29/07/2005

Resumo

Este trabalho propõe uma Metodologia Unificada para o desenvolvimento de sistemas orientados a agentes. A elaboração desta metodologia foi realizada a partir de um estudo do papel do agente dentro da Engenharia de Software e da análise de diversas metodologias orientadas a agentes encontradas na literatura, enfocando-se principalmente em três destas: MaSE (*Multiagent Systems Engineering Methodology*), Prometheus e Tropos, além da linguagem de modelagem AUML (*Agent Unified Modeling Language*). A Metodologia Unificada proposta visa aproveitar o que há de melhor nestas metodologias, buscando elementos comuns a todas elas, de modo análogo ao que ocorreu com a metodologia unificada (RUP – *Rational Unified Process*) em sistemas orientados a objetos. Para validar a Metodologia Unificada e analisar as demais metodologias, um estudo de caso foi modelado. A Metodologia Unificada se mostrou eficiente no projeto, documentação e construção de sistemas multi-agentes, sendo considerada uma metodologia detalhada e mais completa por cobrir os estágios de especificação de requisitos, de análise e de projeto no desenvolvimento de software orientados a agentes.

Palavras-chave: Agente, Engenharia de Software, Engenharia de Software Orientada a Agentes, AUML, MaSE, Prometheus, Tropos.

Abstract

This work proposes a Unified Methodology for the development of agent-oriented systems. The methodology was elaborated based on a study of agent's role within software engineering and the analysis of different agent-based software development methodologies found in the literature, focusing in three main ones: MaSE (*Multiagent Systems Engineering Methodology*), Prometheus and Tropos, in addition to the modeling language AUML (*Agent Unified Modeling Language*). The Unified Methodology aims at taking advantage of the best from each methodology, searching for common elements among them, in an effort similar to what happened with the Unified Methodology (RUP - *Rational Unified Process*) in object-oriented systems. To validate the Unified Methodology and analyze the other methodologies, a case study was developed. The Unified Methodology has shown to be efficient in the design, documentation and construction of multi-agent systems. We conclude it to be a detailed and more complete methodology, covering requirements specification, analysis and design stages of agent-based software development.

Key-words: Agent, Software Engineering, Agent Oriented Software Engineering, AUML, MaSE, Prometheus, Tropos.

Ao meu marido Benedito, ao meu filho Vitor
e aos meus pais.

Agradecimentos

Agradeço,

a Deus, pelo milagre da vida, que é tão maravilhosa...

ao meu marido Benedito, que me completa, por todo seu amor, sua paciência e sua compreensão, sem ele não seria possível chegar até aqui;

ao meu filho Vitor, a alegria da minha vida, por esses momentos que não pudemos ficar juntos, mas que com certeza nos fortalecerão por toda a vida;

ao meu pai e minha mãe, pelo amor que sempre me deram, por me guiarem e servirem de exemplo de vida, ao apoio e incentivo aos meus estudos, principalmente nesse período. Sem falar das orações...

em especial, à minha avó Almerinda, protetora no céu e na terra, e ao meu tio Zezinho, por sempre acreditarem em mim e nos meus estudos;

a todos os meus familiares, aos meus irmãos Rafael e Daniel, à minha tia Cleonice que sempre me deram estímulo e apoio ao longo da minha vida;

ao meu orientador, Prof. Dr. Ricardo Ribeiro Gudwin, pelos conhecimentos que comigo compartilhou e principalmente pelo incentivo e apoio durante todo o tempo em que estive envolvida no trabalho de pesquisa e redação dessa dissertação. A ele devo muito, muito mais do que ele possa imaginar;

à Universidade Estadual de Campinas, onde dedico parte de minha vida contribuindo para o desenvolvimento intelectual e profissional;

às pessoas que me desafiaram, pois elas me ajudaram a crescer.

Há duas formas para viver sua vida: uma é acreditar que não existe milagre. A outra é acreditar que todas as coisas são um milagre.

(Albert Einstein)

Sumário

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

AOSE	Agent Oriented Software Engineering
ACL	Agent Communication Language
AFIT	Air Force Institute of Technology
AIP	Agent Interaction Protocol
AOS	Agent Oriented Software
AUML	Agent Unified Modelling Language
BDI	Belief, Desire and Intention
BMSA	Best Matching Search Algorithm
CA	Communication Act
CNP	Contract-Net Protocol
EC	Engenharia do Conhecimento
ESOA	Engenharia de Software Orientada a Agentes
FIPA	Foundation of Intelligent Physical Agent
GAIA	Generic Architecture for Information Availability
JADE	Java Agent DEvelopment Framework
JDE	Jack Development Environment
MaSE	Multiagent Systems Engineering
NFR	Non-Functional Requirements
OMG	Object Management Group
ONToolkit	Object Network Toolkit
OA	Orientada a Agentes
OO	Orientada a Objetos
PAT	Provincia Autonoma di Trento
PDT	Prometheus Design Tool
RP-Net	Resource Processing Network
RPNToolkit	Resource Processing Network Toolkit
RUP	Rational Unified Process
SMA	Sistemas Multi-Agentes
SNToolkit	Semionic Network Toolkit
UML	Unified Modelling Language

Capítulo 1

Introdução

1.1 Contexto e Motivação

O desenvolvimento de sistemas de software utilizando agentes, denominados de Sistemas Multi-Agentes (SMA), tem crescido como um novo paradigma de computação, abrangendo uma grande diversidade de aplicações, variando desde pequenos sistemas, como por exemplo, filtros de e-mail, até sistemas mais complexos e de missão crítica, como controle de tráfego aéreo [Wooldridge 1995]. Além disso, o termo agente é objeto de pesquisa nos mais diversos campos como Psicologia, Sociologia e em muitas áreas da Ciência da Computação, mais especialmente na área de Inteligência Artificial. A importância crescente dos Agentes e da Engenharia de Software Orientada a Agentes, tornou-se a principal motivação para a realização deste trabalho.

Devido a essa grande diversidade de aplicações e ao seu surgimento recente, foi constatado neste trabalho que o conceito de agentes é abordado muitas vezes de maneira inadequada na literatura, sem rigor formal e sem fundamentações, sendo utilizado meramente para designar um componente de um sistema de computação. Dessa forma, foram surgindo inúmeras definições do termo agente, com várias diferenças e particularidades, o que tem gerado grande confusão. Franklin e Graesser [Franklin 1996] condensam algumas definições de agentes elaboradas por diversos autores. Wooldridge e Jennings [Wooldridge 1995] estabeleceram um conjunto mínimo de características necessárias para identificar um agente:

- **autonomia:** os agentes operam sem a necessidade de intervenção humana ou outra qualquer e têm um certo controle sobre suas ações e estados internos;
- **reatividade:** os agentes percebem seu ambiente e respondem prontamente a mudanças que nele ocorram;
- **pró-atividade:** os agentes não simplesmente reagem em resposta ao ambiente, mas são capazes de exibir condutas baseadas em metas, tomando a iniciativa, em relação a suas próprias ações;
- **habilidade social:** os agentes interagem com outros agentes por meio de uma linguagem de comunicação de agentes (ACL – *Agent Communication Language*).

Além das divergências com relação às definições de agentes, existem também dúvidas com relação ao enfoque dado para o conceito do termo agente, ou seja, será que o conceito de agentes é apenas um enfoque diferente no projeto de sistemas complexos, ou realmente pode ser considerado como um paradigma distinto para a interação entre componentes de um sistema? Esses problemas conceituais também serviram de motivação para este trabalho.

Um outro aspecto tratado nesta dissertação é situar o agente dentro da Engenharia de Software. O principal papel da Engenharia de Software é fornecer modelos e técnicas que

facilitam a construção de sistemas, contribuindo para a diminuição do custo do desenvolvimento do software e para o aumento da qualidade. À medida que aumenta a complexidade dos sistemas de software a serem desenvolvidos, novas abordagens da Engenharia de Software são propostas. Em termos das abstrações usadas para gerenciar a complexidade, é possível perceber uma contínua evolução desde a abstração de procedimentos, passando por tipos de dados abstratos, objetos, componentes e, mais recentemente, agentes. Assim, a Engenharia de Software Orientada a Agentes (ESOA) é uma das mais recentes contribuições na Engenharia de Software. Seu principal objetivo é criar metodologias e ferramentas que possibilitem o desenvolvimento e manutenção de software baseado em agentes [Tveit 2001].

Existem vários benefícios comparados com as abordagens de desenvolvimento existentes, em particular, a habilidade dos agentes representarem uma abstração de alto nível. Segundo Jennings [Jennings 2000a] as técnicas que adotam uma abordagem orientada a agentes são bem adaptadas para o desenvolvimento de sistemas complexos pelas seguintes razões:

- **decomposição:** as decomposições da orientação a agentes são um caminho efetivo para particionar um sistema complexo em subsistemas menores, tornando-os mais fáceis de gerenciar;
- **abstração:** as abstrações da orientação a agentes são uma abordagem natural para modelar sistemas complexos;
- **organização:** a filosofia orientada a agentes para identificar e gerenciar os relacionamentos organizacionais é apropriada para lidar com as dependências e interações que existem em sistemas complexos.

Porém, um dos obstáculos fundamentais para que a tecnologia de agentes se torne uma abordagem de desenvolvimento de software aceita efetivamente nas indústrias é a falta de uma metodologia madura para o desenvolvimento de software orientado a agentes [Dam 2003]. O sucesso e a expansão de Sistemas Multi-Agentes não requerem somente novos modelos e tecnologias, mas também novas metodologias para suportar o desenvolvimento de sistemas mais robustos, confiáveis e reutilizáveis [Zambonelli 2001]. Embora existam algumas metodologias da Engenharia de Software Orientado a Agentes, elas não estão totalmente padronizadas e não cobrem completamente todas as fases de desenvolvimento de software.

A necessidade de uma metodologia uniformizada e consolidada, incluindo modelos que facilitem o processo de desenvolvimento de software, foi uma motivação relevante desse trabalho.

1.2 Objetivos

O objetivo principal desse trabalho é a proposta de uma metodologia unificada para o desenvolvimento de sistemas orientados a agentes, obtida a partir do estudo feito nas metodologias MaSE, Prometheus e Tropos. Propomos reunir pontos fortes de algumas metodologias estudadas com o intuito de aproveitar as vantagens combinadas de cada uma delas ao mesmo tempo em que é possível proporcionar maior objetividade ao processo de desenvolvimento de sistemas. Além desse objetivo, os objetivos secundários são:

1. proposição do conceito de agente como um modelo de computação (interação) distinto: durante os estudos foi diagnosticado que o termo "agente" é utilizado hoje de maneira indiscriminada na literatura, causando muitas confusões sobre seu real significado. Sua proposição como um modelo de interação distinto visa deixar mais claro o que seja ou não um agente, e com isso padronizar melhor sua utilização dentro da Engenharia de Software.
2. proposta de um novo modelo de representação para interação entre agentes: a partir da nova proposição do conceito de agente, propõe-se um novo artefato de documentação para modelar a interação entre agentes.
3. avaliação de diferentes metodologias de desenvolvimento de software orientadas a agentes (MaSE, Prometheus e Tropos) e da linguagem AUML: um estudo de caso foi modelado para cada uma dessas metodologias, o qual serviu de base para a proposição de uma nova metodologia.
4. estudo de caso para a validação da metodologia unificada: investigar a possibilidade de utilização da metodologia para modelagem de um Sistema Multi-Agentes para Busca e Recuperação de Informações.

1.3 Organização da dissertação

Esse trabalho apresenta-se estruturado da seguinte maneira:

- **Capítulo 2 "Agentes":** neste capítulo é apresentada uma discussão teórica sobre agentes. Inicialmente, são abordados os conceitos fundamentais relacionados a agentes encontrados na literatura, incluindo as principais características, tipos de classificações, distinção entre agentes e objetos e a importância do uso de agentes no desenvolvimento de sistemas. Também é apresentada a proposição do conceito de agente como um Modelo de Interação distinto. Em seguida, são abordados os sistemas multi-agentes e alguns exemplos de aplicações utilizando agentes.

- **Capítulo 3 "Engenharia de Software Orientada a Agentes":** a abordagem orientada a agentes está crescendo como um novo paradigma de computação, tornando-se necessário o uso de modelos e técnicas específicos que facilitem a construção de sistemas que utilizem agentes. O objetivo do Capítulo 3 é situar o papel do agente dentro da Engenharia de Software, abordando as particularidades dos agentes que justifiquem a Engenharia de Software Orientada a Agentes.

- **Capítulo 4 "Metodologias da Engenharia de Software Orientada a Agentes":** neste capítulo é feita uma revisão das metodologias da Engenharia de Software Orientada a Agentes (MaSE, Prometheus e Tropos) e da linguagem AUML. Esse estudo serviu de base para a construção da metodologia unificada, proposta neste trabalho.

- **Capítulo 5 "Proposta de um Novo Modelo de Representação de Interação":** a interação dos agentes é um ponto importante nos Sistemas Multi-Agentes, necessitando de artefatos que representem adequadamente essa interação. Os diagramas existentes nas metodologias da Engenharia de Software Orientada a Agentes apresentam

limitações para representar adequadamente a interação dos agentes. Portanto, neste capítulo é apresentada uma proposta de um novo modelo de representação de interação do agente, utilizando a ferramenta RP-Net.

- **Capítulo 6 "Metodologia Unificada para o Desenvolvimento de Sistemas Orientados a Agentes":** este é o capítulo de descrição da metodologia unificada proposta. São apresentados os critérios utilizados para a seleção dos artefatos que compõem a metodologia, bem como uma avaliação das metodologias estudadas no Capítulo 4. São detalhadas todas as fases da metodologia, incluindo seus artefatos e atividades.

- **Capítulo 7 "Estudo de Caso":** para validar a metodologia proposta, foi modelado um Sistema Multi-Agentes de Busca de Informações Científicas. No exemplo apresentado, tratou-se de utilizar todos os artefatos disponíveis na metodologia unificada, servindo de guia no desenvolvimento de sistemas orientados a agentes utilizando tal metodologia.

- **Capítulo 8 "Conclusão":** apresenta um resumo dos principais pontos abordados no trabalho, enfatizando-se as contribuições apresentadas, bem como sugerindo as perspectivas futuras da continuidade do trabalho, como possíveis experimentos, aplicações da metodologia e questões a serem exploradas, envolvendo o aperfeiçoamento da metodologia apresentada no presente trabalho.

- **Apêndice A:** apresenta os artefatos da metodologia Prometheus.

- **Apêndice B:** apresenta a modelagem do estudo de caso utilizando as metodologias MaSE, Prometheus e Tropos.

Capítulo 2

Agentes

2.1 Introdução

Este capítulo apresenta alguns conceitos fundamentais sobre agentes encontrados na literatura, necessários para a compreensão deste trabalho, incluindo algumas definições de agentes, suas principais características, classificações e algumas aplicações.

A teoria de agentes está sendo considerada uma área de pesquisa muito importante e seu desenvolvimento tem crescido muito ultimamente. O principal motivo desse interesse é o seu potencial no desenvolvimento de sistemas complexos, constituindo um paradigma de software apropriado para aplicações em ambientes abertos, heterogêneos e distribuídos como, por exemplo, a Internet. Outra razão para o crescimento das pesquisas realizadas em agentes está relacionada ao fato de um agente ser considerado uma abstração de mais alto nível (quando comparado, por exemplo, a um objeto), definido em termos de comportamentos e não em termos de atributos ou métodos, tornando-se uma maneira mais natural e intuitiva para a modelagem de diversos tipos de problemas, conforme discutiremos na seção 2.8.

Atualmente, os agentes têm sido usados em uma grande diversidade de aplicações e estão sendo adotados em várias sub-disciplinas da tecnologia da informação, incluindo redes de computadores, engenharia de software, inteligência artificial, interação homem-máquina, sistemas móveis, sistemas de controle, comércio eletrônico, entre outros.

A Seção 2.2 apresenta os principais conceitos de agentes encontrados na literatura. A Seção 2.3 caracteriza os agentes, incluindo suas propriedades e seu ambiente. Na Seção 2.4 é detalhada a proposição do conceito de agente como um Modelo de Interação distinto. Na Seção 2.5 são abordadas as classificações dos agentes. A Seção 2.6 inclui uma discussão a respeito dos termos agentes e objetos. A Seção 2.7 detalha os sistemas multi-agentes. Na Seção 2.8 são abordadas as questões sobre a importância do agente, em que situações eles devem ser utilizados e exemplos de algumas aplicações de agentes. E finalmente, na seção 2.9 é apresentado um resumo do capítulo.

2.2 Conceitos

O que é um agente? Está é uma pergunta que a comunidade de pesquisadores de agentes está tentando definir há anos. A origem do conceito de um agente, como uma entidade organizacional, data de meados de 1970, quando foi utilizada por Carl Hewitt na sua pesquisa de Modelo de Ator [Hewitt 1977]. Hewitt propôs o conceito de um objeto interativo, independente e com execução concorrente, que ele chamou de Ator [Nwana 1996a]:

> "é um agente computacional que tem um endereço de e-mail e um comportamento. Os atores se comunicam pelo envio de mensagem e executam suas ações concorrentemente" [Hewitt 1997].

Desde então, os pesquisadores têm apresentado uma variedade de definições, muitas até conflitantes, cada uma tentando explicar o uso que se faz da palavra agente. Essas definições estão associadas a diferentes pontos de vista e dependem muito da funcionalidade fornecida pelo agente em questão. Embora existam consensos sobre algumas características dos agentes, ainda não existe uma definição universalmente aceita do termo "agente". A dificuldade em encontrar uma definição única está na importância que diferentes características têm em diferentes domínios de aplicações. Por exemplo, em algumas aplicações a capacidade de aprendizagem do agente é imprescindível e em outras não é importante, às vezes até indesejável. Para Nwana [Nwana 1996a], uma das razões que dificultam a formulação de uma definição padrão é que até mesmo dentro da comunidade de software, a palavra agente é um termo usado em diferentes ramos de pesquisa e desenvolvimento. Além disso, outros sinônimos e especializações da palavra agente estão sendo criados: *softbots* (software *robots*), *taskbots* (robôs que realizam tarefas), *knowbots* (*knowledge-based robots*), assistentes pessoais; o que tem contribuído para aumentar a confusão na definição do termo.

A seguir serão apresentadas algumas definições de agentes elaboradas por diversos autores, sumarizadas por Franklin e Graesser [Franklin 1996]:

- "Um agente é qualquer coisa que pode ser vista percebendo um ambiente por meio de sensores e atuando no mesmo por meio de atuadores" (Russel e Norvig) [Franklin 1996].
- "Agentes autônomos são sistemas computacionais que habitam um ambiente complexo e dinâmico, sensoreiam e atuam autonomamente neste ambiente, realizando desta maneira uma série de metas e tarefas para as quais foram projetados" (Pattie Maes - MIT Media Lab) [Franklin 1996].
- "Um agente é uma entidade persistente de software dedicada a um propósito específico" (Smith, Cypher e Spohrer - Apple - KidSim) [Franklin 1996].
- "Agentes inteligentes realizam continuamente três funções: percepção das condições dinâmicas de um ambiente, ação de modo a afetar condições do ambiente e raciocínio para interpretar percepções, realizar inferências e determinar ações" (Barbara Hayes-Roth - Stanford) [Franklin 1996].
- "Agentes inteligentes são entidades de software que realizam um conjunto de operações em nome de um usuário ou outro programa com certo grau de independência ou autonomia, e desta maneira empregam algum conhecimento ou representação das metas e/ou desejos do usuário" (IBM's Intelligent Agent Strategy) [Franklin 1996].
- "Um agente é um sistema de hardware e/ou software que goza das seguintes propriedades: autonomia, habilidade social, reatividade, pró-atividade" (Wooldridge e Jennings) [Franklin 1996].
- "Agentes autônomos são sistemas capazes de uma ação autônoma e propositada no mundo real" (Brustoloni e Franklin) [Franklin 1996].
- "Um agente autônomo é um sistema que é parte de um ambiente, estando situado dentro dele, e sente e age sobre este ambiente, no tempo, de acordo com seus próprios

propósitos, de modo a alterar o que sentirá no futuro" (Stan Franklin e Art Graesser) [Franklin 1996].

- "Um agente é um sistema de computador situado em um ambiente, sendo capaz de agir de maneira autônoma em seu ambiente para atingir seus objetivos" (Wooldridge) [Wooldridge 1999a].

2.3 Características dos Agentes

Wooldridge e Jennings [Wooldridge 1995] apresentam duas noções gerais para o conceito de agente, utilizadas para caracterizar um agente. A primeira delas é chamada de agência fraca e a segunda, chamada de agência forte, é normalmente mais discutida entre os pesquisadores da área.

A agência fraca utiliza o termo agente para denotar qualquer hardware ou sistema de computação baseado em software, que apresente as seguintes características:

- **autonomia:** os agentes operam sem a necessidade de intervenção humana ou outra qualquer, e têm um certo controle sobre suas ações e estados internos.
- **reatividade:** os agentes percebem seu ambiente (que pode ser o mundo real, um usuário via uma interface gráfica de usuário (GUI), uma coleção de outros agentes, a Internet ou uma mistura de todos estes) e respondem prontamente a mudanças que nele ocorram.
- **pró-atividade:** os agentes não simplesmente reagem em resposta ao ambiente, mas são capazes de exibir condutas baseadas em metas, tomando a iniciativa, em relação a suas próprias ações.
- **habilidade social:** os agentes interagem com outros agentes (e possivelmente com humanos) por meio de uma linguagem de comunicação de agentes (ACL).

A agência forte deve apresentar, além das características da agência fraca, as noções de estados mentais, ou seja, um agente é descrito como um sistema de computação que, além de apresentar as propriedades identificadas na noção fraca, deve ser definido ou implementado utilizando-se conceitos que usualmente são aplicáveis aos seres humanos, como conhecimentos, crenças, intenções e obrigações.

Outras características dos agentes geralmente consideradas no contexto de agência incluem [Gudwin 2001, Murch 1999, Wooldridge 1999a]:

- **racionalidade:** é a suposição de que o agente sempre agirá para alcançar suas metas e nunca contra seus objetivos, pelo menos na medida em que suas crenças o permitam.
- **benevolência:** é a suposição de que os agentes não têm objetivos contraditórios e que todo agente tentará sempre responder ao que lhe foi perguntado.
- **veracidade:** é a suposição que um agente não comunicará informações falsas de maneira intencional.
- **robustez:** o agente deve ser capaz de lidar com erros, recursos escassos e dados incompletos.

- **auto-gerenciabilidade:** deve ser capaz de realizar a gestão de seu próprio ciclo operacional, ou seja, iniciar e cessar seu comportamento de acordo com critérios próprios.
- **continuidade:** deve ser um processo que roda continuamente, ou seja, tem um ciclo operacional próprio.
- **aprendizagem:** deve utilizar suas experiências prévias para aprender e adaptar-se a mudanças no ambiente.
- **mobilidade:** deve ser capaz de se transportar entre máquinas participantes de uma rede de computadores.
- **rastreabilidade:** o agente deve poder ser monitorado, apesar de independente, para o caso de agentes móveis.
- **personalidade:** deve ter individualidade, ou seja, a capacidade de diferenciar-se de outros agentes iguais a ele.
- **emocionabilidade [1]:** deve poder exibir "estados emocionais" que caracterizem seu estado diante das metas que visa cumprir e do estado atual do ambiente.
- **credibilidade:** deve poder proporcionar a ilusão de ser um ser vivo, com o qual o usuário se comunica.

A escolha de quais características devem estar presentes em um agente depende principalmente da funcionalidade que o projetista pretende dar ao seu agente. Um agente não precisa possuir todas estas características, embora suas capacidades estejam diretamente associadas à presença delas.

Além das características necessárias para identificação do agente, Gudwin [Gudwin 2001] identifica quatro requisitos mínimos para que um sistema de software possa ser chamado de um agente:

- **ambiente:** todo agente deve ter um ambiente, mesmo que este ambiente não seja um espaço bi ou tridimensional.
- **sensores e atuadores**: que permitam ao agente coletar dados do ambiente e atuar sobre ele.
- **ciclo operacional:** todo agente deve possuir um ciclo operacional no qual os passos de percepção e ação são executados continuamente.
- **objetivo(s):** é a razão que baliza o comportamento do agente, ou seja, todo agente age em função de uma meta.

A Figura 2.1 ilustra um agente prototípico, situado em um ambiente, interfaceado por sensores e atuadores, por meio dos quais coleta dados no ambiente e atua sobre ele, respectivamente.

[1] A questão se os agentes podem ou não expressar emoções está sendo discutida intensamente pela comunidade científica, como exemplo: Gmytrasiewicz [Gmytrasiewicz 2001], Sarmento [Sarmento 2004], Sloman [Sloman 2001] e Bates [Bates 1994].

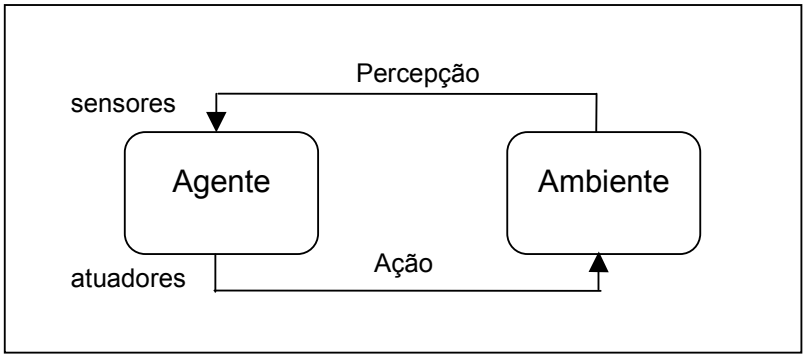


Figura 2.1: Exemplo de um agente (sensores, atuadores e ambiente).

2.3.1 Ambiente dos Agentes

Um fator importante que afeta a determinação das características do agente e a sua complexidade é o ambiente em que ele está inserido. Russel e Norvig [Russel 1995] sugerem uma extensa classificação do ambiente dos agentes:

- **acessível x inacessível:** um ambiente é acessível ao agente se seus sensores são capazes de perceber o estado completo do ambiente, obtendo informações atualizadas e precisas.
- **determinístico x não-determinístico:** um ambiente determinístico é aquele cujo próximo estado é completamente determinado pelo estado atual e pelas ações selecionadas pelo agente, não existindo nenhuma incerteza quanto ao resultado de uma ação.
- **episódico x não-episódico:** em um ambiente episódico, as ações dos agentes são divididas em episódios. Cada episódio consiste de uma percepção e uma ação, onde a qualidade da ação depende apenas do episódio atual, não dependendo do que aconteceu em episódios anteriores.
- **estático x dinâmico:** um ambiente é dinâmico se o seu estado pode mudar enquanto o agente está deliberando. No ambiente estático, o agente não precisa ficar observando o mundo enquanto está decidindo as suas ações.
- **discreto x contínuo:** um ambiente é discreto se existe um número limitado de percepções e ações claras e distintas. No ambiente contínuo, é impossível numerar todos os estados possíveis.

Quanto mais acessível for o ambiente, mais simples será projetar e construir os agentes que nele operam, pois é possível identificar o estado completo do ambiente, obtendo informações atualizadas sempre que necessário. Neste contexto não está sendo considerada a complexidade da tarefa a ser realizada pelo agente.

O ambiente não-determinístico é uma grande fonte de complexidade em um agente, pois impõe sérias restrições à sua capacidade de realizar previsões relacionadas ao seu estado futuro. O mesmo acontece com os ambientes não-episódicos, pois a ação do agente depende de sua

capacidade de armazenar a qualidade das ações realizadas anteriormente. Os ambientes dinâmicos também afetam a complexidade do agente, pois o agente não pode assumir que o ambiente permanecerá inalterado se não realizar a ação durante o intervalo de tempo estabelecido. Ou seja, neste tipo de ambiente a ação realizada em um intervalo de tempo diferente do pré-estabelecido poderá gerar resultados totalmente distintos. Um ambiente estático é muito mais simples, pois uma vez recolhidas as informações do ambiente, o agente pode assumir que este ambiente não será modificado, a não ser pelas suas próprias ações. A última característica dos ambientes diz respeito a estes serem discretos ou contínuos. Os ambientes discretos são mais simples para os agentes do que os contínuos, principalmente pois nesses últimos os agentes estarão sempre tomando suas ações a partir de informações aproximadas.

Desta forma, percebe-se que os ambientes mais complexos são os inacessíveis, não-determinísticos, não-episódicos, dinâmicos e contínuos. Quanto maior a complexidade do ambiente do agente, mais específicas devem ser as características atribuídas a ele.

2.4 Agentes como um Modelo de Interação

Como vimos anteriormente, o conceito do que seja um "agente" é às vezes por demais vago e impreciso, o que permite sua utilização em situações onde seu emprego poderia até ser questionável. Isso decorre de uma conceitualização por demais intuitiva do que seja um agente, que não caracteriza necessariamente esse conceito como envolvendo uma entidade computacional distinta. Para muitos autores, chamar um determinado módulo de software, componente de um sistema, de um "agente" acaba sendo meramente uma conveniência terminológica, ou uma estratégia de "marketing" que visa apenas se apropriar do halo de modernidade associado ao uso de tal termo. Com o intuito de se obter uma definição mais técnica, que evidencie os agentes como um paradigma computacional distinto, fizemos uma análise dos assim chamados "modelos de computação" relatados na literatura, classificando-os de acordo com os tipos de interação existentes em cada caso. Dessa forma, propomos aqui que o uso legítimo do termo "agente" envolve um modelo distinto de computação e mais do que simplesmente uma abstração diferente para abordar um problema, os agentes podem ser entendidos como um paradigma independente de programação.

Chang [Chang 1997] define um "Modelo de Computação" como a semântica de interação entre os módulos ou componentes de um sistema. Na literatura existem diversos tipos de modelos de computação, tais como Processos Seqüenciais, Máquinas de Estados Finitos, Máquina de Turing, *Dataflow*, Redes de Processos, Simuladores de Eventos Discretos, Linguagens Síncronas, Redes de Petri, Sistemas Heterogêneos, Passagem de Mensagens, entre outros. A discussão em torno dos "Modelos de Computação" é bastante complexa e sofisticada, indo além do escopo que se pretende neste trabalho. Por uma questão de simplicidade, e atendendo nossos objetivos aqui, restringimos nossa abordagem a um dos sub-tópicos desta discussão, que trata dos "Modelos de Interação". A proposta que apresentamos a seguir, adaptada das propostas envolvendo diferentes "modelos de computação", classifica os modelos de interação entre subsistemas da seguinte maneira:

- Modelo do Tipo *Dataflow*,
- Modelo do Tipo Passagem de Mensagens e
- Modelo de Busca por Mensagens

2.4.1 Modelo de Interação do Tipo Dataflow

O modelo de interação mais simples é o modelo do tipo *Dataflow*. Neste modelo de interação, as saídas de um componente estão diretamente acopladas às entradas de outros componentes do sistema e a interação é realizada de maneira síncrona.

O principal objetivo desse tipo de modelo é extrair o paralelismo naturalmente existente na composição de um sistema, adotando-se a filosofia de que existe sempre algum dado disponível nas entradas e saídas de cada subsistema, a cada instante de tempo. O processamento executado por cada componente pode ser discreto (e não contínuo) e os componentes podem até agir em diferentes instantes de tempo (sem um sincronismo perfeito), mas utilizará sempre os dados disponíveis em suas interfaces de entrada, como se esse processamento fosse contínuo e síncrono. Em outras palavras, no modelo do tipo *Dataflow*, um componente não espera pelo processamento de outros componentes, executando seu processamento com o dado que estiver disponível no instante do seu processamento. Assim, muitas instruções podem ser executadas em paralelo, desde que o sistema possua vários elementos de processamento. Outra característica deste modelo de interação é que o processamento realizado por cada componente não cessa nunca. Após realizar um processamento, segundo seus ciclos internos, cada componente coleta seus novos dados e passa a um novo processamento dos mesmos. Alguns exemplos que pertencem a essa categoria são: Redes Neurais, Programação Estruturada e *Streams*. Na Figura 2.2, temos um exemplo genérico de um sistema construído de acordo com o modelo de interação do tipo *Dataflow*. O componente 1 gera dados de saída que são utilizados pelos componentes 2 e 4. O componente 2, por sua vez, coleta esses dados e executa seu processamento, gerando os dados que serão utilizados pelos componentes 3 e 4. O componente 3 utiliza os dados disponibilizados pelo componente 2 e gera dados na sua interface de saída, os quais serão utilizados pelo componente 4. O componente 4 coleta os dados gerados pelos componentes 1, 2 e 3 e os utiliza para gerar os dados de saída do sistema.

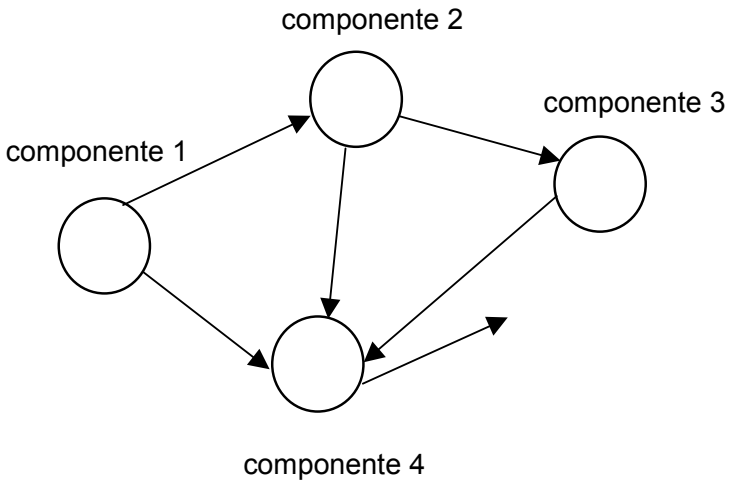


Figura 2.2: Exemplo de um Modelo de Interação do Tipo Dataflow.

Um outro exemplo mais especializado, o de uma Rede Neural, é apresentado na Figura 2.3.

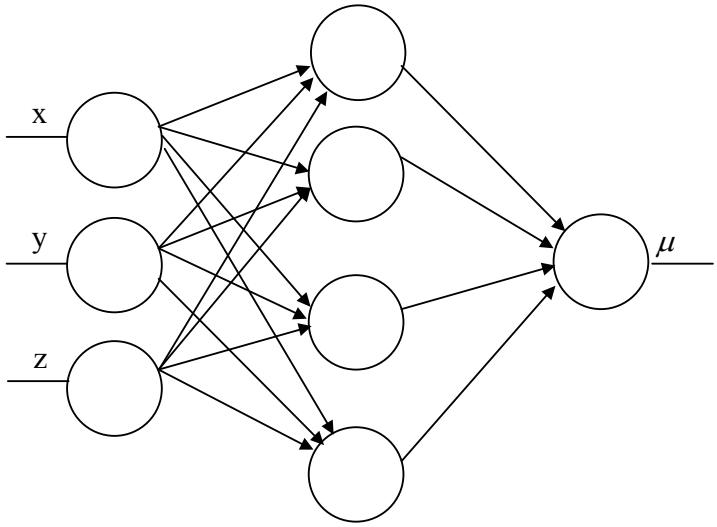


Figura 2.3: Exemplo de uma Rede Neural representando o Modelo de Interação do Tipo Dataflow.

2.4.2 Modelo de Interação do Tipo Passagem de Mensagens

O modelo de interação do tipo *Dataflow* é muito utilizado, tanto em sistemas realmente síncronos (onde existe algum sinal de sincronismo distribuído a todos os componentes), como em sistemas assíncronos (onde não existe esse sinal de sincronismo), mas que assume a disponibilidade dos dados a todo instante. Entretanto, em muitos tipos de sistema essa restrição pode ser por demais severa, devido ao total assincronismo entre o processamento dos componentes. Neste caso, a ordem em que os componentes executam seu processamento pode ser importante, o que torna o modelo *Dataflow* inadequado. Um modelo alternativo é então o modelo de interação por Passagem de Mensagens. Neste modelo, a interação entre os componentes ocorre por meio do envio de mensagens entre os diferentes componentes do sistema, sendo que não há uma conexão mais forte entre esses componentes, como ocorre no modelo *Dataflow*. Essas mensagens são eventos assíncronos que disparam um determinado comportamento por parte do componente, e podem ser mensagens síncronas ou assíncronas. Mensagens síncronas demandam uma resposta imediata ao componente que a enviou, que fica aguardando, em estado de suspensão, a resposta do outro componente. Mensagens assíncronas não demandam uma resposta imediata, sendo capturadas por filas de entrada e processadas pelo componente de destino na medida do possível. O comportamento global do sistema é totalmente assíncrono, sendo que cada componente somente executa algum processamento quando alguma mensagem lhe é direcionada, ficando normalmente em estado de suspensão quando não há mensagens. Alguns exemplos que utilizam essa modalidade de interação são: Programação Orientada a Objetos, Sistemas de Atores, Redes de Processos e Redes de Petri. Na Figura 2.4, vemos um exemplo de um sistema genérico onde o modelo de Passagem de Mensagens é seguido.

Uma sofisticação desse tipo de Modelo de Interação é o paradigma Publish-Subscribe [Eugster 2003, Muhl 2002].

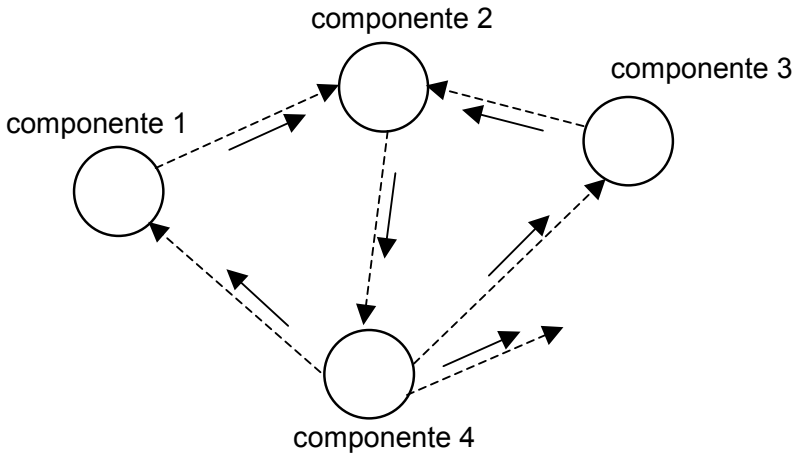


Figura 2.4: Exemplo de um Modelo de Interação do Tipo Passagem de Mensagens.

2.4.3 Modelo de Interação do Tipo Busca por Mensagens

No modelo do tipo *Dataflow*, os componentes executam seu processamento a todo instante. No modelo do tipo Passagem de Mensagens, os componentes só executam seu processamento quando chega alguma mensagem, estando inoperantes nos demais instantes de tempo. Muitas vezes, o que se desejaria era um híbrido desses dois comportamentos. Deseja-se que o sistema opere o tempo todo, mas deseja-se também a flexibilidade e a riqueza de comportamentos proporcionada pelo modelo de Passagem de Mensagens. Esse modelo híbrido é portanto o modelo de interação de Busca por Mensagens. Neste modelo de interação, os componentes de um sistema interagem também por meio de mensagens, mas ao invés de enviá-las diretamente aos seus destinatários, como ocorre no modelo de Passagem de Mensagens, aqui os componentes depositam as mensagens em *buffers* de armazenamento, onde podem ser buscadas posteriormente por seus destinatários e o componente que gerou a mensagem não precisa em princípio conhecer o componente que irá consumi-la. Da mesma forma que no modelo do tipo *Dataflow*, os componentes estão o tempo todo processando (buscando por mensagens) e, da mesma forma que no modelo do tipo Passagem de Mensagens, os componentes podem se utilizar de mensagens direcionadas a outros componentes para realizar a interação com estes. Podem ainda aproveitar-se de mensagens não direcionadas a nenhum outro componente em particular, criando uma riqueza ainda maior em seu comportamento.

Um exemplo onde pode se encontrar o uso deste paradigma ocorre nos sistemas distribuídos que utilizam espaços de tuplas para se comunicar [Figueiredo 2003]. Um espaço de tuplas tem como função criar uma abstração de memória compartilhada sobre um sistema distribuído. Processos distintos do sistema podem executar uma tarefa comum, usando o espaço de tuplas como meio de comunicação, coordenação e sincronização. Dessa forma, processos espalhados pelo sistema distribuído têm no espaço de tuplas um dispositivo de memória que é comum a todos e no qual podem ler e escrever. O espaço de tuplas tem como principal

característica o desacoplamento, ou seja, a tupla desconhece o processo que irá consumi-la. Essa propriedade distingue o modelo de espaço de tuplas do modelo de Passagem de Mensagens. No modelo de espaço de tuplas, a mensagem tem existência própria, independentemente dos processos que se comunicam através dela. Gelernter [Gelernter 1985] chamou isso de comunicação geradora (*generative communication*), no sentido que a comunicação gera uma terceira identidade independente, salientando a diferença conceitual do seu modelo com o modelo de Passagem de Mensagens.

Verificamos que este modelo de interação é intrínseco ao funcionamento de um agente (ou pelo menos de um agente como nós desejaríamos entendê-lo), pois ele busca as informações no ambiente por meio de sensores e atua no ambiente através dos atuadores, sem se preocupar com o processo que produziu a informação desejada ou que irá consumi-la. Dessa forma, propomos que para que um agente possa ser chamado propriamente de um agente, ele deve seguir o modelo de interação do tipo "Busca por Mensagens". Alguns exemplos que seguem esse modelo de interação: Sistemas distribuídos utilizando Espaços de Tuplas (e.g., *JavaSpace*), Linda [Gelernter 1985] e Agentes, de um modo geral. A Figura 2.5 apresenta um exemplo de um modelo de interação do tipo Busca por Mensagens.

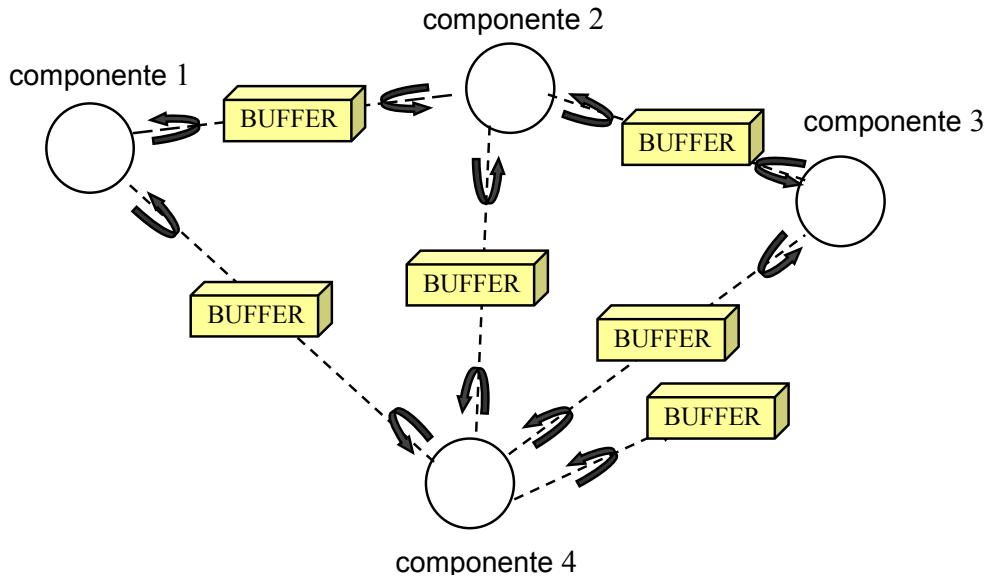


Figura 2.5: Exemplo de um Modelo de Interação do Tipo Busca por Mensagens.

2.5 Tipologia de Agentes

O número elevado de características apresentadas na Seção 2.3 permite perceber que é quase impossível implementar um agente que incorpore todas essas características, principalmente pelo fato de estarem relacionadas com o tipo de aplicação a ser realizada pelo agente. Com o objetivo de analisar melhor os vários domínios existentes da tecnologia, os pesquisadores definiram critérios para classificar os agentes.

Um critério para a classificação do agente pode ser a sua mobilidade. Segundo este critério, os agentes podem ser classificados como agentes móveis, que possuem a habilidade de

mover-se por diferentes nós de uma rede ou agentes estáticos, caso contrário. A seguir serão apresentados alguns tipos de classificações existentes na literatura.

Russel e Norvig [Russel 1995] classificam os agentes em quatro tipos:

- **Agentes reativos simples:** são agentes que reagem ao estado do mundo apenas a partir de um conjunto de regras (condição-ação) pré-determinadas. Perante a percepção de uma dada situação, selecionam e executam uma dada ação pré-especificada. Os agentes deste tipo podem ser considerados os mais simples, pois o seu comportamento é puramente reativo.
- **Agentes reativos baseados em modelos:** esses agentes possuem uma representação do estado do mundo que é atualizada dinamicamente com a percepção do agente. Desta forma, a reatividade destes agentes é condicionada pelas experiências anteriores que se encontra refletida nesse estado do mundo. Este modo de funcionamento implica que a mesma percepção ocorrida em momentos distintos e conseqüentemente estados diferentes do mundo, tenha como resultado ações diferentes.
- **Agentes baseados em objetivos:** os agentes baseados em objetivos são capazes de armazenar informações sobre os objetivos que pretendem atingir. Neste caso, a decisão pode implicar na pesquisa e planejamento prévios, antes da seleção da ação a ser executada em cada instante.
- **Agentes baseados em utilidade:** a utilidade corresponde a uma medida de satisfação do agente relativa ao cumprimento dos seus objetivos. A utilidade será mais elevada se o estado do mundo atual do agente estiver próximo de atingir os seus objetivos. Os agentes baseados em utilidade conseguem ser mais racionais, pois possuem capacidades para avaliar a utilidade da execução de uma determinada ação.

Outra possível classificação de agentes leva em consideração a estrutura interna funcional do agente [Gudwin 2001]. Neste caso, podemos ter:

- **Agentes Reativos ou Reflexivos**: apresentam um único processo de percepção-ação (reflexo). A ação é uma função direta das entradas dos sensores. Essa função pode ser um conjunto de regras condição-ação (*fuzzy* ou binária), rede neural ou função matemática (Figura 2.6). Alguns exemplos de trabalhos baseados em agentes reativos: simulação de veículos autônomos [Laugier 1992] e PENGI [Agre 1989].

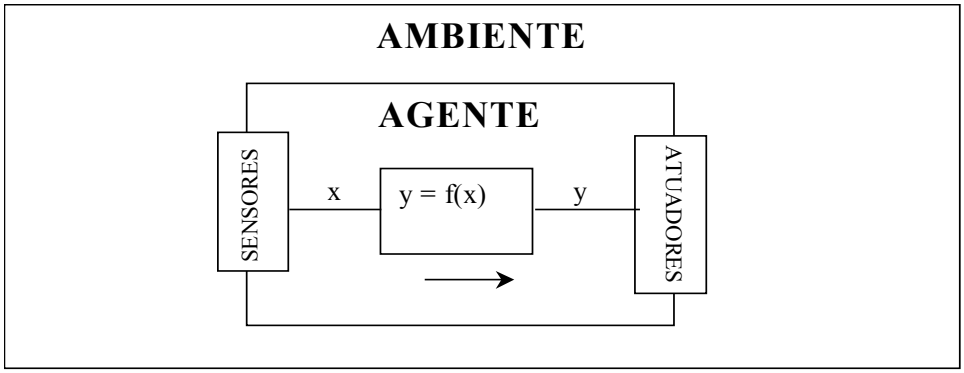


Figura 2.6: Agente Reativo ou Reflexivo.

- **Agentes Comportamentais:** apresentam processos independentes de percepção e ação. A percepção é um processo independente, controlado por um módulo de percepção. Possui um conjunto pré-definido de comportamentos, que são selecionados dependendo da percepção. O seu comportamento geral é sofisticado e possui limites em sua aplicabilidade, a qual vai depender de uma especificação adequada do conjunto de comportamentos disponíveis. O mecanismo de ação pode corresponder a diversos mecanismos reflexivos em paralelo, sendo que apenas um deles está operacional em um determinado instante (Figura 2.7). Um exemplo é a arquitetura de *subsumption* de Brooks [Brooks 1986].

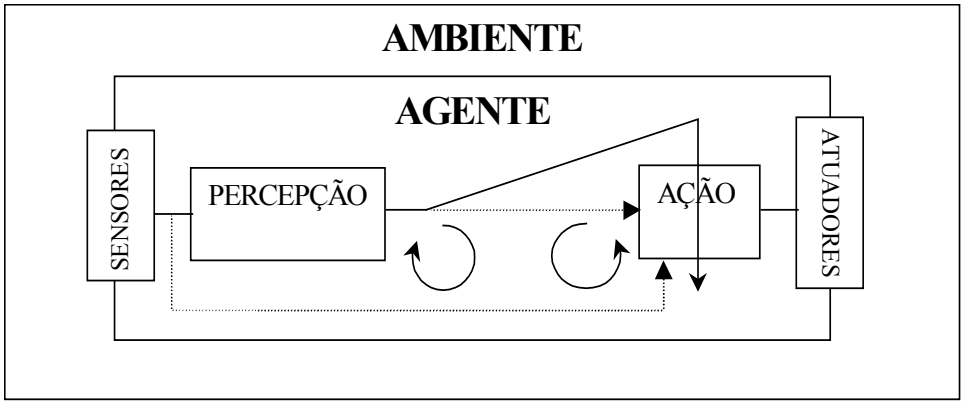


Figura 2.7: Agente Comportamental.

- **Agentes Planejadores:** apresentam processos independentes de percepção e ação, mas possuem internamente uma representação do mundo (modelo do mundo, que é conhecido a priori ou pode ser gerado dinamicamente por meio da interação com ambiente, através dos mecanismos de percepção). A ação não é gerada diretamente pela percepção, devido à existência de um mecanismo de geração de comportamentos, capaz de elaborar predições (usando o modelo do mundo) de como o mundo se comportará diante de possíveis ações tomadas pelo agente, além de um mecanismo de geração de planos (conjunto de ações a serem tomadas pelo agente) que permitirá ao agente escolher o plano que melhor satisfizer os objetivos do sistema e executá-lo definitivamente (Figura 2.8). Alguns exemplos de agentes planejadores: Internet Softbot [Etzioni 1994] e a arquitetura IRMA [Bratman 1988].

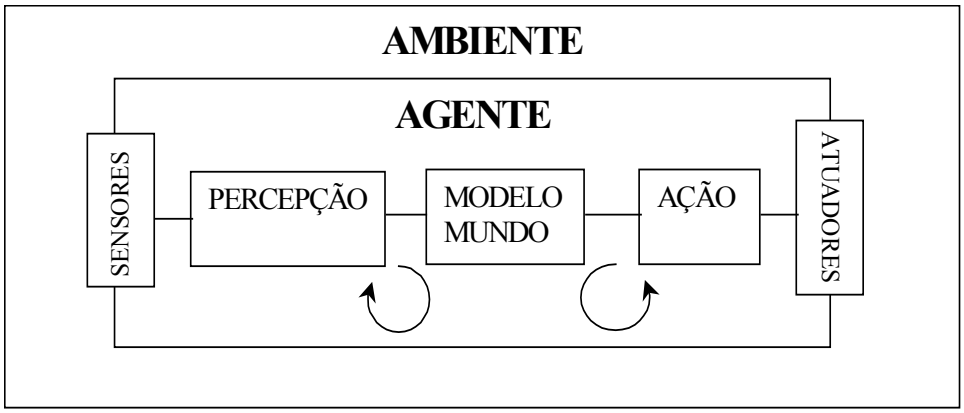


Figura 2.8: Agente Planejador.

- **Agentes Emocionais:** muito parecidos com os agentes planejadores, só que acrescido de um sistema de valor (emoções) que permitirá avaliar internamente se os objetivos do agente estão sendo cumpridos a contento. Este sistema de valor pode ser utilizado de modo a influir no planejamento de ações futuras e também pode ser atribuído a estados atuais ou a previsões de estados futuros. Existem diferentes tipos de emoções (ou de simulações de emoções); por exemplo: medo, desejo, dor, alegria (Figura 2.9). Nesta categoria estão os agentes simulados em vida artificial, tais como, Creatures [Cliff 1999] e OZ Project [Bates 1994].

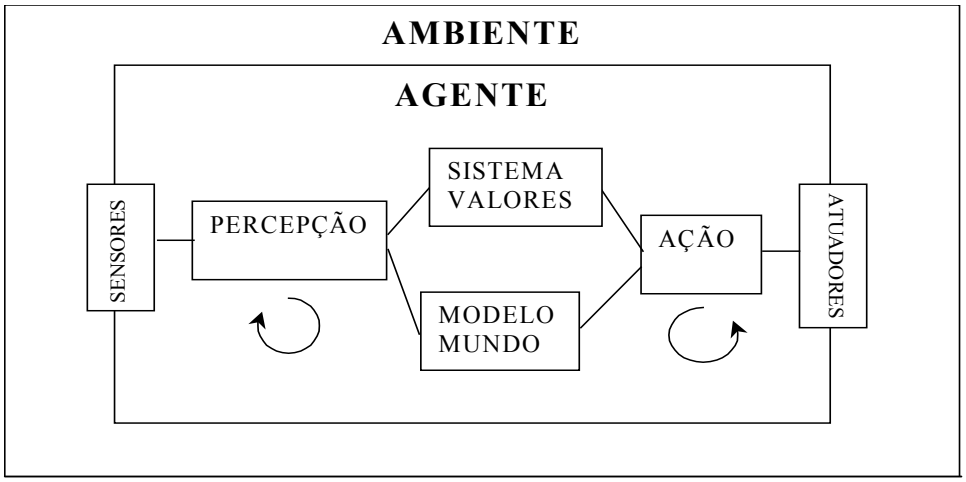


Figura 2.9: Agente Emocional.

- **Agentes Comunicativos:** utilizados em sistemas multi-agentes. Os agentes comunicativos apresentam um canal de comunicação direta para se comunicar com outros agentes. Precisam de uma linguagem de comunicação de agentes (*Agent Communication Language* – ACL) (Figura 2.10). Existe uma grande quantidade de trabalhos na literatura baseados em agentes comunicativos, tais como: Ensino a distância [Gomes 2003] e SACI [Hubner 2000].

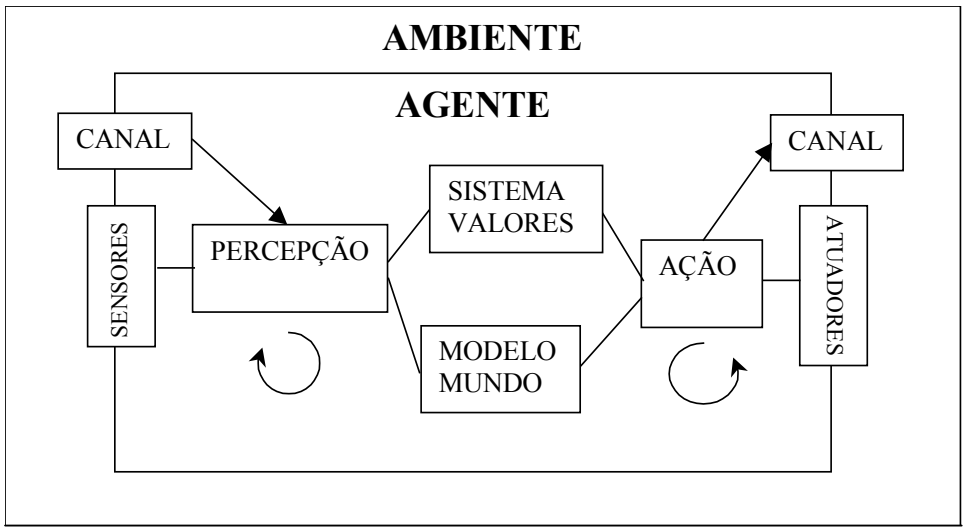


Figura 2.10: Agente Comunicativo.

- **Agentes Semióticos:** correspondem a uma sofisticação dos agentes emocionais, incorporando as funções dos agentes comunicativos. São mais sofisticados e complexos que os anteriores, pois interpretam e geram signos[2] para o ambiente. Não necessitam de canal de comunicação direta, como nos agentes comunicativos, pois o ambiente é o canal de comunicação utilizado. Os mecanismos de percepção e ação devem ser mais sofisticados (Figura 2.11). Este tipo de agente está em desenvolvimento por Gudwin [Gudwin 2001, Gudwin 2003, Gudwin 2004].

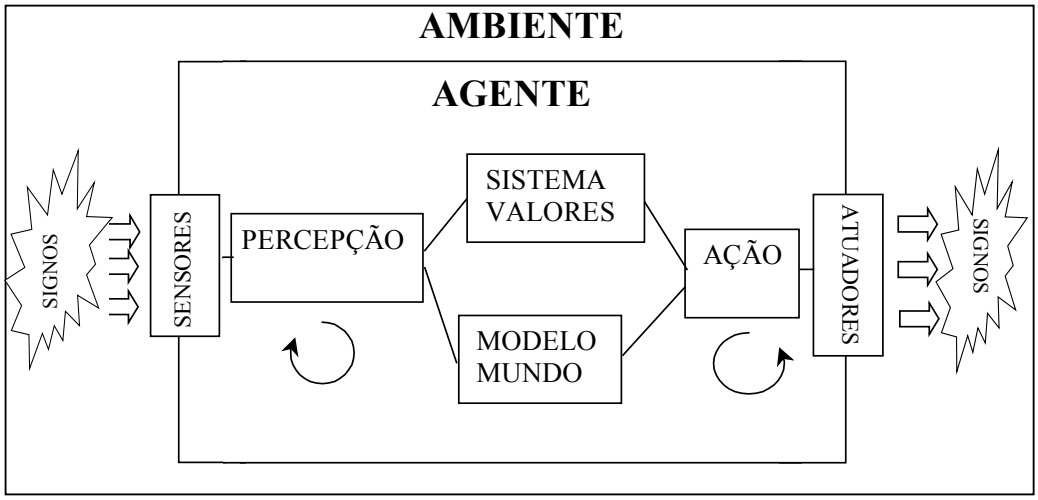


Figura 2.11: Agente Semiótico.

Outras classificações de agentes baseiam-se nas aplicações dos agentes, como por exemplo, pesquisa de informação, assistentes pessoais, controle de tráfego aéreo, comércio eletrônico, entre outros.

2.6 Agentes e Objetos

Uma das grandes dificuldades existentes na comunidade científica que estuda agentes consiste em estabelecer claramente a distinção entre os conceitos de agente e de objeto. Embora existam semelhanças óbvias entre agentes e objetos, existem também diferenças significativas entre eles. Com isso vários estudos abordam a comparação entre agentes e objetos [Gudwin 2001, Odell 2002, Wooldridge 1999a]. A seguir são apresentadas as principais características dos objetos, dos agentes e, por fim, é feita uma comparação entre eles.

[2] De uma maneira genérica, "um signo [...] é aquilo que, sob certo aspecto ou modo, representa algo para alguém." [Peirce, 1935 (CP 2.228)]. De uma maneira prática, signos são sinais existentes ou criados no ambiente (ou na memória do agente) que são utilizados com o propósito de comunicação. O conceito de signo de certa forma amplia o conceito de símbolo, ou representação simbólica, uma vez que um símbolo é um signo, mas existem outros tipos de signos, tais como os índices e os ícones.

2.6.1 Objetos

Um objeto é uma "entidade" que tem uma identidade independente do seu estado, que encapsula uma coleção de atributos (sua memória privada), o qual é capaz de manipular de acordo com um conjunto de ações bem definidas, e que é capaz de interagir com outros objetos [Fiadeiro 1991].

Gudwin [Gudwin 2001] apresenta um sumário com as características mais importantes para definir objetos:

- possuem estados internos que os descrevem;
- podem receber mensagens, a partir das quais realizam uma atividade, a qual é determinada conforme as mensagens enviadas e seus correspondentes parâmetros;
- podem ser classificados hierarquicamente em classes, que basicamente descrevem os estados internos e as mensagens que um objeto de uma determinada classe pode receber;
- possuem comportamento previsível;
- o objeto não age por si só, responde a uma requisição de serviço, ou seja, a decisão sobre a execução de uma ação pertence ao objeto que invocou o método (Figura 2.12).

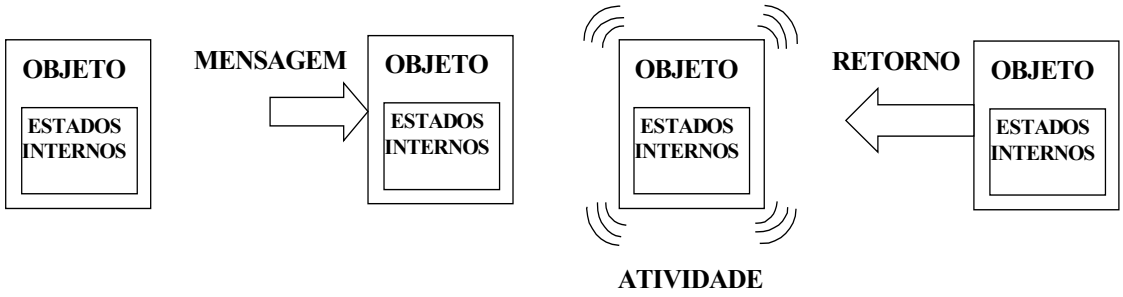


Figura 2.12: Comportamento esperado de um objeto [Gudwin 2001].

2.6.2 Agentes

Assumindo a definição genérica, dada por Wooldridge [Wooldridge 1999a], "um agente é um sistema de computador situado em um ambiente, sendo capaz de agir de maneira autônoma em seu ambiente para atingir seus objetivos".

Gudwin [Gudwin 2001] apresenta um sumário com as características mais importantes para definir agentes:

- possuem estados internos que os descrevem;
- podem extrair informações de seu ambiente por meio de sensores e atuar sobre o ambiente por meio de seus atuadores;
- possuem um ciclo operacional por meio do qual sensoreiam e atuam;
- nunca cessam suas atividades, ficando envolvidos em um loop infinito de observação do ambiente;
- comportamento pode ser imprevisível;

- um agente age por si só, não "recebe comandos", mas busca por mensagens enviadas na forma de informações do ambiente que podem influenciar o seu comportamento, ou seja, a decisão sobre a execução de uma ação pertence ao agente que recebe a requisição para executar a ação, pois é baseada no seu estado interno (Figura 2.13).

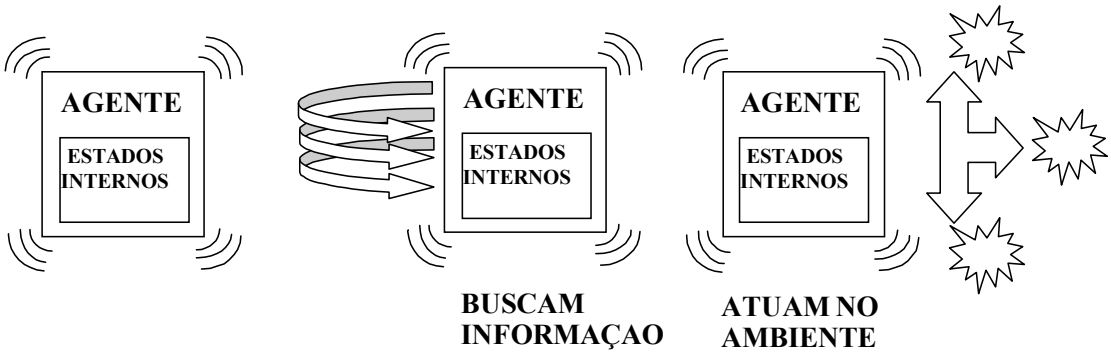


Figura 2.13: Comportamento esperado de um agente [Gudwin 2001].

2.6.3 Comparando Agentes e Objetos

As principais diferenças entre agentes e objetos estão relacionadas com os seguintes aspectos:

1. os agentes possuem capacidade de decisão autônoma: possuem controle sobre o seu comportamento. São capazes de iniciar uma ação independentemente de qualquer solicitação, podendo decidir, em cada situação, se irão ou não realizar a ação solicitada. Os objetos, por sua vez, são convencionalmente passivos. Embora possa ter autonomia sobre seu estado (dados), não possui autonomia sobre o seu comportamento, ou seja, o objeto estará sempre disponível quando invocado por um método. Os agentes têm o direito de possuir um comportamento não-determinístico, eles podem dizer "não". Por outro lado, os objetos possuem um comportamento mais previsível. Se um objeto disser "não", será considerado como uma situação de erro que não foi projetada pelo desenvolvedor. Embora alguns objetos possam ter um comportamento mais ativo, é importante destacar que a diferença fundamental está no fato de que os objetos ativos não apresentam um comportamento autônomo flexível como o de um agente. Um agente é dito autônomo na medida que sua capacidade de tomar decisões é baseada nas suas experiências e não nos conhecimentos embutidos pelos projetistas (Russel e Norvig, 1995) [Gomes 2000]. Esta distinção entre agentes e objetos é sumarizada no seguinte slogan:

"Objetos fazem de graça; agentes fazem por interesse". [Wooldridge, 1999a].

2. os agentes possuem uma forma mais complexa de interação: A interação implica na habilidade de comunicar com o ambiente e com outras entidades. Os objetos interagem através das mensagens que são enviadas pela invocação dos métodos (forma mais básica de interação). Os agentes possuem uma forma mais sofisticada de interação, pois podem estabelecer interações com outros agentes, agindo como uma sociedade de

agentes [Odell 2002]. A metodologia GAIA [Wooldridge 2000], por exemplo, utiliza o conceito de organização do sistema. Uma organização é vista como uma coleção de papéis, que se mantêm em certos relacionamentos uns com os outros. Além disso, estes papéis participam de padrões sistemáticos e institucionalizados de interação com outros papéis numa organização multi-agentes [Silva 2003].

3. os agentes possuem um comportamento reativo, pró-ativo e social: o paradigma orientado a objetos não contém qualquer referência a esses tipos de comportamento essenciais nos agentes. Desta maneira, a abordagem orientada a agentes pode ser vista como um próximo passo na evolução natural das abordagens da Engenharia de Software. Enquanto na abordagem orientada a objetos a unidade primária de abstração é o objeto, definido em termos de seus atributos e métodos, na abordagem orientada a agentes a unidade primária de abstração é o agente, definido em termos de comportamentos, podendo ser modelado como uma sociedade de entidades autônomas que interagem, possuem habilidade social e tomam decisões orientadas a metas.

4. os agentes possuem um fluxo de controle próprio: O fluxo de controle também é considerado um aspecto importante que distingue um agente de um objeto. Essa distinção baseia-se essencialmente no fluxo de controle único dos objetos *standard*, o que invalida a noção de entidade autônoma. Essa distinção diminui quando se trata de objetos ativos. Os objetos ativos são supostos terem o seu próprio fluxo de controle sem necessariamente terem a capacidade de exibir um comportamento autônomo flexível.

Embora existam várias diferenças entre agentes e objetos, algumas técnicas desenvolvidas exclusivamente para orientação de objetos (com as linguagens de programação C++, Java ou linguagem de modelagem UML) estão estendendo seus conceitos para abordarem a orientação a agentes, permitindo a utilização das duas abordagens concomitantemente no desenvolvimento de sistemas.

2.7 Sistemas Multi-Agentes

O procedimento de se decompor sistemas complexos em diferentes entidades, com o objetivo de se obter maior eficiência, não é uma abordagem recente nas ciências da computação. A Inteligência Artificial Distribuída, uma sub-área da Inteligência Artificial, tem dedicado grandes esforços na construção de uma sociedade de solucionadores de problemas ou agentes que interagem de forma a resolver um problema comum. Essa sociedade é denominada de Sistemas Multi-Agentes.

Segundo O'Hare e Jennings [O'Hare 1996], um Sistema Multi-Agentes pode ser definido como uma rede de solucionadores de problemas que trabalham juntos, cada qual com diferentes capacidades de percepção e ação do mundo, para solucionar problemas que vão além de suas capacidades individuais. Cada agente deve ter conhecimento da sua existência e da existência de outros agentes no ambiente, deve possuir conhecimentos e habilidades para executar uma determinada tarefa e deve, a priori, cooperar para atingir um objetivo global.

Além do fato de grande parte dos problemas mais freqüentemente encontrados serem inerentemente distribuídos, as principais motivações para o crescimento da área de Sistemas Multi-Agentes incluem suas habilidades em:

- solucionar problemas muito grandes para serem resolvidos por um único agente centralizado, principalmente devido às limitações de recursos;
- fornecer soluções para problemas geográficos ou funcionalmente distribuídos;
- oferecer clareza conceitual e simplicidade de projeto;
- oferecer maior velocidade, confiabilidade, atribuindo diferentes tarefas a diferentes agentes de forma que a execução seja mais rápida;
- oferecer maior robustez, pois se utilizam vários agentes, não existindo um ponto centralizado de falha no sistema.

Segundo Sichman et al. [Sichman 1992] os principais problemas encontrados na abordagem de Sistemas Multi-Agentes são:

- descrição, decomposição e alocação de tarefas: como uma tarefa complexa deve ser descrita e decomposta em sub-tarefas e como essas sub-tarefas devem ser alocadas e em que ordem devem ser executadas;
- interação, linguagem e comunicação: quais primitivas básicas de comunicação devem ser utilizadas para exprimir os conceitos semânticos envolvidos no trabalho cooperativo;
- coordenação, controle e comportamento coerente: como garantir um comportamento global coerente numa coleção de agentes, tendo cada um suas próprias habilidades/objetivos e como deveria ser projetado o controle de tal sistema;
- conflitos e incertezas: como podem ser resolvidos os conflitos que surgem, considerando que os agentes não têm informação completa sobre o seu ambiente e como dados incompletos podem ser distribuídos de forma a garantir resultados coerentes;
- linguagens e ambientes de programação: sob o ponto de vista computacional, quais linguagens de programação devem ser utilizadas e quais requisitos um ambiente de programação deve satisfazer para possibilitar o desenvolvimento e teste de Sistemas Multi-Agentes.

Essas questões têm sido levadas em consideração pela comunidade de pesquisadores de Sistemas Multi-Agentes, sendo que o foco dos estudos está no desenvolvimento de princípios e modelos computacionais para construir, descrever, implementar e analisar as formas de coordenação e interação de agentes.

2.7.1 Coordenação em Sistemas Multi-Agentes

A resolução das tarefas realizadas pelos agentes no Sistema Multi-Agentes é uma forma distribuída de resolução de problemas, envolvendo coordenação, negociação e comunicação. A coordenação é a parte central de um Sistema Multi-Agentes, pois é responsável por determinar a estrutura organizacional dos agentes, garantindo um comportamento harmônico e coerente entre

eles. Em outras palavras, ela mostra como as tarefas executadas individualmente por cada agente podem ser coordenadas para obter a funcionalidade de todo o conjunto e como os conflitos existentes entre os agentes devem ser negociados. Esses conflitos podem originar desde um simples acesso a recursos limitados até divisões complexas de tarefas entre os agentes. A definição de boas estratégias de coordenação permite que grupos de agentes executem tarefas cooperativas de forma eficiente através de decisões conjuntas sobre quais agentes devem executar uma determinada tarefa, quando e com quem devem estabelecer comunicação em cada instante.

A coordenação em um Sistema Multi-Agentes pode ser centralizada ou distribuída. Na coordenação centralizada os agentes enviam seus planos individuais para um coordenador central que irá analisar esses planos e identificar possíveis conflitos. Na coordenação distribuída não existe a figura de um coordenador, os agentes se comunicam para construir e atualizar seus planos até que os conflitos sejam removidos.

Uma técnica de coordenação muito utilizada é a Rede de Contratos (*Contract-Net Protocol* – CNP). Considerado um mecanismo de coordenação de nível elevado, o Protocolo de Rede de Contratos baseia-se no mesmo mecanismo usado nas empresas para regular a troca de bens e serviços entre si, em que os agentes podem assumir um dos dois papéis: gestor ou contratante. O funcionamento básico inicia-se quando um gestor convida outros agentes para participarem na resolução de um problema por meio de um leilão anunciado por um mecanismo de difusão. Cada agente disposto a aceitar a proposta oferece os seus recursos, retornando a oferta. Uma vez decidido pelo gestor o problema é aceito automaticamente pelo contratante vencedor. As principais vantagens no Protocolo de Rede de Contratos incluem a possibilidade de atribuir tarefas dinamicamente e permitir aos agentes a decisão sobre a participação ou não no processo. As principais desvantagens relacionam-se ao fato da detecção da dificuldade da resolução de problemas não ser efetuada pelo gestor, mas sim pelo contratante, além do fato que os anúncios para o leilão serem efetuados por difusão (broadcast), podendo gerar um *overhead* na rede.

2.7.2 Comunicação em Sistemas Multi-Agentes

A comunicação é outro mecanismo essencial em um Sistema Multi-Agentes. Os agentes se comunicam entre si, trocando informações sobre o estado do sistema, sobre seus planos e suas intenções. Na área de Sistemas Multi-Agentes, a comunicação é tratada a um nível mais elevado do que em outras áreas da Ciência da Computação. Os agentes precisam se comunicar uns com os outros através de uma linguagem de comunicação que seja entendida por todos. Esta linguagem é conhecida como linguagem de comunicação de agentes (*Agent Communication Language* – ACL). A maioria das ACL deriva da Teoria dos Atos Comunicativos (ou Atos de Fala - *Speech Acts*), desenvolvida pelos lingüistas na tentativa de compreender e descrever como os humanos usam a linguagem nas situações do dia a dia para cumprir seus objetivos. O conceito fundamental por trás da Teoria dos Atos Comunicativos é o de "*performativa*". Nesta teoria, argumenta-se que ao emitirmos uma mensagem, esta expressa, além de seu conteúdo primário, um ato comunicativo ou uma intenção de comunicação. Esse ato comunicativo é o que é explicitado na *"performativa"*. O KSE (*Knowledge Sharing Effort*), consórcio envolvendo empresas, universidades e centros de pesquisa, desenvolveu dois dos principais produtos utilizados para a troca e representação de informação entre sistemas:

- **Linguagem KQML (*Knowledge and Query Manipulation Language*):** é uma linguagem de comunicação entre agentes. As mensagens KQML são chamadas performativas e cada mensagem é enviada com o objetivo de gerar uma determinada ação. KQML é dividida em três níveis: conteúdo, mensagem e comunicação. O nível de conteúdo define o propósito da mensagem, o nível de mensagem contém o conjunto de performativas fornecidas pela linguagem e o nível de comunicação define o protocolo de entrega da mensagem e o seu encapsulamento. As principais vantagens da linguagem KQML estão relacionadas ao fato de ser uma linguagem padrão e por separar o domínio da performativa KQML e a semântica da mensagem. Uma das principais críticas feitas com relação à linguagem KQML está relacionada com as mudanças significativas do conjunto de performativas sofridas ao longo do tempo.

- **Formato KIF (*Knowledge Interchange Format*):** não foi desenvolvido como um formato para expressar mensagens completas, mas como um formato para expressar o conteúdo das mensagens. Baseia-se na lógica de primeira ordem com extensões para o suporte de raciocínio não monotônico e decisões. É utilizada para a representação do conteúdo das mensagens KQML. KIF também foi definido para servir como uma linguagem intermediária entre outras linguagens, nos processos de tradução entre linguagens.

Além do KQML e do KIF, existem várias linguagens relacionadas com a comunicação em Sistemas Multi-Agentes, como por exemplo, a ACL desenvolvida pela FIPA (*Foundation for Intelligent Physical Agent*). A FIPA-ACL é semelhante ao KQML, definindo essencialmente a estrutura exterior da mensagem, não obrigando a utilização de qualquer linguagem de conteúdo específica. Uma das principais vantagens da FIPA-ACL sobre a KQML consiste na existência de performativas mais adequadas à execução de processos de negociação, além de possuir um conjunto otimizado de performativas.

2.8 Por que os agentes são importantes?

Por que os agentes são importantes e em que situações eles devem ser utilizados? Essas questões devem ser analisadas cuidadosamente ao se adotar a abordagem orientada a agentes no desenvolvimento de sistemas.

Um agente representa um outro tipo de abstração de software, da mesma maneira que métodos, funções e objetos também representam abstrações de software. O agente é autônomo, percebe seu ambiente, responde às mudanças do ambiente (reativo) e possui condutas baseadas em metas (pró-ativo). Muito mais do que ser definido em termos de atributos e métodos, um agente é definido pelo seu comportamento. Ele não recebe simplesmente um conjunto de entradas que são processadas e gera uma saída, mas processa um conjunto de entradas segundo um propósito específico (toma uma decisão de maneira embasada) e é em função desse propósito que ele baliza o seu comportamento. Durante o desenvolvimento do sistema orientado a agentes, o foco do projetista passa a ser o comportamento de cada agente, avaliando as tarefas que o agente deve cumprir para atingir seu objetivo, diferentemente da abordagem orientada a objetos, onde o funcionamento do sistema é feito de maneira mecânica, com uma seqüência de operações a serem realizadas, sem a preocupação com o propósito do seu funcionamento.

Essas características dos agentes estão diretamente relacionadas com a maneira de pensar do ser humano, pois representam uma abstração de mais alto nível e uma evolução natural no desenvolvimento de uma grande variedade de sistemas complexos, justificando a adoção desse novo paradigma.

Porém, quando se diz que a abordagem orientada a agentes pode ser utilizada uma grande variedade de sistemas complexos, é necessário analisar cuidadosamente os requisitos do sistema para certificar se os agentes realmente são um mecanismo de implementação apropriado para o domínio do problema. Ou seja, não se pode considerar a abordagem orientada a agentes como uma solução universal para todos os tipos de aplicações. Existem situações em que os agentes são apropriados, porém existem outros domínios de problemas em que outras tecnologias são mais adequadas. Basicamente os agentes são adequados para aplicações que envolvem um trabalho repetitivo e contínuo e que demandam uma tomada de decisões inteligentes, tal como em sistemas de monitoração, controle de tráfego aéreo, gerenciamento de informações, comércio eletrônico, *business intelligence*, aplicações médicas, entre outros. A seguir serão examinadas três das mais promissoras áreas de agentes: agentes de interface, agentes de informação e agentes móveis.

2.8.1 Agentes de Interface

Maes [Maes 1994a] define agentes de interface como:

> "Programas de computador que empregam técnicas da inteligência artificial para fornecer assistência ao usuário para lidar com uma aplicação particular... A metáfora é a de um assistente pessoal que colabora com o usuário no seu mesmo ambiente".

Os agentes de interface também são chamados de assistentes pessoais ou agentes de usuário. Possuem o objetivo de simplificar as tarefas rotineiras realizadas por um usuário. Essencialmente o agente de interface observa e monitora as ações do usuário na interface, aprende novos "atalhos" e sugere melhores formas de desenvolver a tarefa. O agente de interface age como um assistente pessoal autônomo, o qual coopera com o usuário na realização de determinadas tarefas. Nwana e Ndumu [Nwana 1996b] identificam quatro maneiras de os agentes de interface aprenderem com seus usuários:

- observando e imitando o usuário;
- mediante recebimento de um *feedback* positivo ou negativo por parte do usuário;
- recebendo instruções explícitas do usuário;
- mediante ajuda de outros agentes.

Os agentes de interface estão aumentando cada vez mais a complexidade de suas tarefas, o que está tornando seu desenvolvimento cada vez mais atrativo. Os principais desafios no desenvolvimento deste tipo de agente estão relacionados à privacidade do usuário (em que situações o agente deve aparecer sem ser chamado? O agente pode tomar o controle da situação?), à adaptabilidade com outras áreas (projetar e implementar o comportamento apropriado a uma vasta gama de aplicações) e à interação do agente (suportar uma interação

continuada e não simplesmente diálogos localizados com o agente, ou seja, ir além de pura narrativa ou monólogo e criar conversações).

Kozierok e Maes [Kozierok 1993] elaboraram um Agente de Interface (*Calendar Agent*) que auxilia os usuários no agendamento de reuniões. O agente aprende as preferências do usuário e indica, por exemplo, o melhor dia para uma reunião.

Open Sesame! [Hoyle 1997] é um agente de interface que aprende a forma como os usuários trabalham com o computador. Ele observa as atividades do usuário e aprende quais tarefas são repetidas mais vezes e, em seguida, se oferece para executar essas tarefas repetitivas automaticamente para o usuário.

2.8.2 Agentes de Informação

O desenvolvimento dos agentes de informação tem sido favorecido pela grande quantidade de informações disponíveis na Internet e a dificuldade gerada em encontrar e indexar essas informações de forma a fornecê-las ao usuário. Esses agentes têm a função principal de gerenciar, manipular e coletar informação de pelo menos uma origem distribuída, incluindo um banco de dados tradicional e a Internet [Nwana 1996a]. Os *WebRobots*, *Spiders* ou *Wanderers* são agentes de informação utilizados para obter informações válidas para os usuários, localizar *links* inválidos, realizar a manutenção de páginas de um determinado site, filtrar o *cache* de páginas potencialmente interessantes, descobrir páginas novas e/ou conteúdo novo, entre outros.

A mobilidade não é um requisito obrigatório para o agente de informação; porém, para Nwana [Nwana 1996a] o principal problema encontrado em agentes de informações estáticos é manter seus índices atualizados em um ambiente de elevada dinamicidade. Alguns pesquisadores levantam também questões relacionadas com a ética de agentes de informação e com a dificuldade em localizar e selecionar as fontes relevantes para a descoberta da informação desejada.

Maes [Maes 1994b] desenvolveu um agente de informação denominado de NewT. O usuário indica alguns exemplos de artigos que ele gostaria e que não gostaria de receber e o agente busca na Internet somente os artigos considerados interessantes pelo usuário.

Etzioni e Weld [Etzioni 1994] descrevem um agente de informação denominado Internet Softbot. É a implementação de um agente que permite a seus usuários formular consultas em alto nível. A chance de achar informação pertinente é alta e a quantidade de informações não-pertinentes ("ruído") é relativamente baixa. É um bom exemplo de um agente orientado a meta, pois o Softbot permite a um usuário especificar o que realizar, enquanto dirige as decisões de como e onde realizá-las.

2.8.3 Agentes Móveis

Os agentes móveis representam um novo paradigma na computação distribuída. Agentes móveis são processos de software capazes de navegar em redes, interagir com outros servidores, obter a informação de interesse de seu proprietário (ou utilizar recursos não existentes em sua máquina de origem) e voltar ao seu lugar de origem tendo executado suas tarefas [Nwana 1996a]. Um agente estacionário por sua vez é executado somente no local onde ele foi originalmente executado. Se ele necessitar de informações que não estão no sistema, ele utiliza algum

mecanismo de comunicação para acessar outros agentes ou recursos. Os agentes móveis não estão ligados permanentemente ao sistema que o iniciou enquanto processo. Possui a habilidade única de se transportar de um sistema a outro em uma rede, mantendo seu contexto original antes do transporte.

A Figura 2.14(a) mostra um cliente se comunicando com um serviço remoto da rede utilizando o paradigma cliente-servidor. A Figura 2.14(b) mostra um cliente utilizando um agente móvel para fazer sua tarefa, o agente é despachado para o serviço remoto, onde ele executa localmente o serviço.

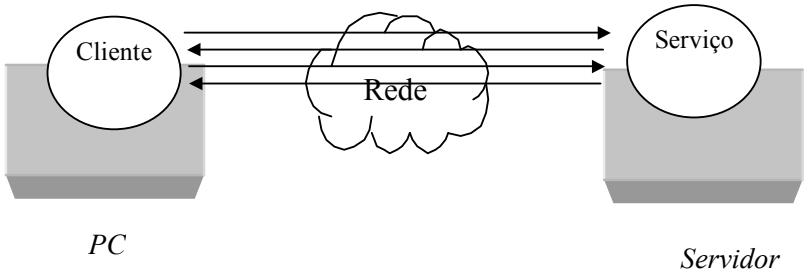


Figura 2.14 (a): Paradigma Cliente-Servidor

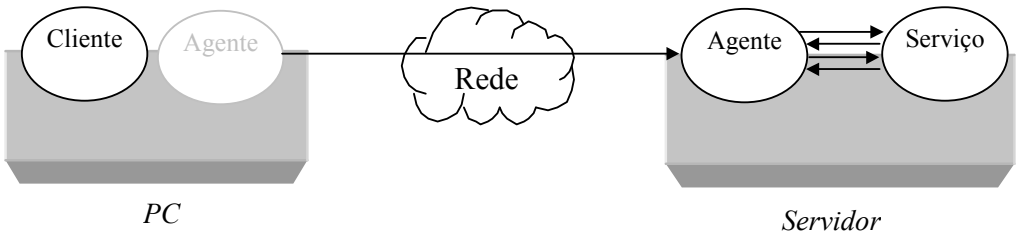


Figura 2.14 (b): Paradigma Agente Móvel

Embora a mobilidade não seja uma propriedade obrigatória de um agente, a sua existência pode trazer vários benefícios ao usuário, entre os quais podemos destacar:

- Redução do tráfego na rede e conseqüentemente do custo da comunicação: quando grandes volumes de dados estão armazenados remotamente e necessitam serem processados, ao invés de trazer os dados para serem processados localmente, o agente é enviado para processá-los remotamente.
- Comunicação assíncrona: as tarefas que demandam conexões continuamente abertas entre um dispositivo móvel e uma rede podem não ser economicamente viáveis. Os agentes móveis podem ser configurados e enviados pela rede, tornando-se processos independentes do processo que os criou. Posteriormente, o dispositivo móvel pode se reconectar para coletar o agente de volta.

Um dos principais problemas relacionados a este tipo de agente é relativo à segurança: um agente dessa categoria seria, por exemplo, um excelente disseminador de vírus dentro da rede na qual estivesse inserido.

A primeira aplicação comercial de agentes móveis foi um comunicador pessoal inteligente (Sony's Magic Link PDA). Essencialmente ele auxilia no gerenciamento dos e-mails do usuário,

telefone e *pager*, conectando-se a serviços de comunicação e mensagem, tais como o America Online e o AT&T PersonaLink Services [Nwana 1996a].

2.9 Resumo

Neste capítulo foi feita uma revisão da literatura da teoria de agentes, uma área que está sendo adotada em várias sub-disciplinas da tecnologia da informação e tem sido usada em uma grande diversidade de aplicações. O objetivo foi introduzir a noção do que é um agente e as razões de sua crescente popularidade.

Iniciou-se com um pequeno resumo dos diferentes conceitos do termo agente, bem como suas principais características e classificações. Foram analisados os modelos de computação existentes com o objetivo de se obter uma definição dos agentes, como um paradigma computacional distinto. As diversas classificações de agentes utilizadas atualmente também foram apresentadas.

Um aspecto fundamental abordado neste trabalho foi a proposição do conceito de agente como um modelo de interação distinto, mais especificamente estamos situando o agente como um Modelo de Interação do Tipo Busca por Mensagens. Também foi apresentada a comparação entre agentes e objetos. Essa comparação é necessária para a compreensão da Engenharia de Software Orientada a Agentes. Em seguida, abordou-se a importância dos agentes e em quais situações eles devem ser utilizados.

Foi apresentado um breve estudo sobre os Sistemas Multi-Agentes e finalmente, foram examinadas três das mais promissoras áreas na utilização de agentes: agentes de interface, agentes de informação e agentes móveis.

No próximo capítulo, será abordada a Engenharia de Software Orientada a Agentes, ou seja, a utilização da abordagem orientada a agentes no contexto da Engenharia de Software.

Capítulo 3

Engenharia de Software Orientada a Agentes

3.1 Introdução

No Capítulo 2 foram apresentados os principais conceitos da tecnologia de agentes. Este capítulo tem como objetivo discutir como a Engenharia de Software considera o desenvolvimento de sistemas envolvendo agentes. A construção de software de qualidade para aplicações do mundo real é muito difícil. Portanto, o uso de uma metodologia de modelagem de sistemas bem definida é fundamental para o bom desenvolvimento de projetos. O principal papel da Engenharia de Software é fornecer modelos e técnicas que facilitem a construção de sistemas, contribuindo para a diminuição do custo do desenvolvimento do software e para o aumento da qualidade. Com este objetivo, vários paradigmas foram propostos na literatura, como por exemplo, análise estruturada, orientação a objetos, arquiteturas de software, padrões de projetos, dentre outros. Esta evolução incremental dos paradigmas visa tornar a Engenharia de Software mais gerenciável bem como permite aumentar a complexidade das aplicações a serem construídas.

As técnicas de desenvolvimento de software orientadas à análise funcional ou à modelagem orientada a objetos são insuficientes em muitos casos, pois não capturam a flexibilidade e o dinamismo necessários nos sistemas em desenvolvimento. Muitos pesquisadores estão desenvolvendo novos métodos e abordagens para tentar solucionar esses problemas. Uma das mais promissoras abordagens do momento é a chamada Engenharia de Software Orientada a Agentes (*Agent Oriented Software Engineering* - AOSE) [Ciancarini 2001, Wooldridge 2002, Giunchiglia 2003, Giorgini 2004, Odell 2005].

A abordagem orientada a agentes está crescendo como um novo paradigma de computação e tem sido recomendada no desenvolvimento de sistemas complexos e distribuídos, principalmente por oferecer um maior nível de abstração se comparada com outras abordagens. Os agentes de software possuem autonomia, habilidade social, se comunicam, coordenam e cooperam para atingir seus objetivos. Como a modelagem de agentes na Engenharia de Software é uma área relativamente nova, atualmente os projetistas de software ainda não podem explorar todos os benefícios oferecidos pelo paradigma de agentes devido à ausência de notações diagramáticas, metodologias e ferramentas de projeto conhecidas para o desenvolvimento de sistemas orientado a agentes [Bergenti 2000]. Neste sentido, a comunidade de pesquisadores da Engenharia de Software Orientada a Agentes está interessada na elaboração de metodologias que suportem a construção de sistemas multi-agentes.

A Seção 3.2 apresenta uma visão geral da Engenharia de Software. Na Seção 3.3. discute-se o uso de agentes no desenvolvimento de sistemas complexos. A Seção 3.4 inclui uma reflexão a respeito da Orientação a Agentes e a Orientação a Objetos. A Seção 3.5 descreve o ciclo de vida do desenvolvimento de software e sua relação com o paradigma de agentes. Na Seção 3.6 são apresentadas algumas das metodologias de desenvolvimento de software orientadas a

agentes, bem como suas classificações. A Seção 3.7 aborda alguns cuidados que se deve ter com a utilização do paradigma de agentes na Engenharia de Software. E finalmente, é apresentado um resumo do capítulo.

3.2 Engenharia de Software

O principal papel da Engenharia de Software é fornecer modelos e técnicas que facilitem a construção de software de qualidade, de maneira produtiva. Dessa maneira, as metodologias da Engenharia de Software oferecem métodos, descrições, técnicas e ferramentas para as fases do ciclo de vida do desenvolvimento de sistema.

> "Engenharia de Software é a aplicação de uma abordagem sistemática, disciplinada e quantificada para o desenvolvimento, operação e manutenção de software" [IEEE 1990].

Desde a década de 70, muitas pesquisas têm sido feitas com o intuito de facilitar e diminuir o custo do desenvolvimento de sistemas de software complexos, resultando em vários paradigmas da Engenharia de Software. No início, as aplicações eram mais simples e os programadores escreviam seus programas em uma linguagem muito próxima à linguagem de máquina. Em seguida, a programação passou a ser estruturada, composta por procedimentos e sub-rotinas que permitiram ao programador desenvolver seus códigos de uma maneira mais organizada, viabilizando sistemas mais complexos. As aplicações foram se tornando cada vez mais complexas e com maior qualidade. Foi quando surgiu a orientação a objetos, com o objetivo de diminuir a lacuna entre o mundo real e os sistemas de software. Devido ao seu maior nível de abstração e suas propriedades, a orientação a objeto tornou-se bastante popular na Engenharia de Software, tanto no meio acadêmico como nas indústrias. Entretanto, a crescente complexidade de sistemas distribuídos, com módulos autônomos e desacoplados, acabou por demandar a criação de um novo paradigma, que ofereça um maior nível de abstração na forma de pensar sobre as características e os comportamentos dos sistemas de software, do que o paradigma da orientação a objetos. Para atender a essas novas necessidades surgiu a Engenharia de Software Orientada a Agentes (ESOA).

> "Engenharia de Software Orientada a Agentes é a aplicação de agentes na Engenharia de Software em termos de fornecer meios para analisar, projetar e construir sistemas de software" [Dam 2003].

Wooldridge e Jennings [Wooldridge 1999b] afirmam que o argumento mais simples em favor dos agentes na Engenharia de Software é que eles são uma evolução dos paradigmas anteriores, representando uma outra maneira de abstrair. Segundo Dam [Dam 2003] as características dos agentes podem resultar no uso de funcionalidades avançadas na Engenharia de Software e os sistemas multi-agentes podem expandir a complexidade e qualidade das aplicações do mundo real. Wooldridge conclui que se é para a tecnologia ter sucesso, é preciso levar a sério seus aspectos da Engenharia de Software, principalmente entender as situações em que as soluções de agentes são apropriadas e utilizar técnicas de desenvolvimento fundamentadas. Da mesma maneira que existem metodologias bem definidas para o desenvolvimento de sistemas

orientados a objetos, é necessário que a orientação a agentes também contenha metodologias e técnicas que garantam a qualidade do processo de produção de software.

3.3 Uso de Agentes em Sistemas Complexos

As aplicações industriais são complexas por natureza, principalmente por serem compostas por um grande número de partes com muitas interações. Wooldridge e Jennings [Wooldridge 1999b] apresentam alguns itens sobre o que torna um software complexo: (i) a natureza do ambiente do sistema; (ii) a natureza da interação entre os sistemas; (iii) a natureza da especificação do sistema. Dessa maneira, a finalidade da Engenharia de Software é prover estruturas e técnicas que facilitem o tratamento da complexidade.

Segundo Jennings e Wooldridge [Jennings 2001], a complexidade de um sistema apresenta características importantes, dentre as quais podem-se destacar:

- a complexidade tem forma hierárquica, ou seja, um sistema é composto de sub-sistemas inter-relacionados que, por sua vez, apresentam hierarquias, e assim sucessivamente.
- a escolha de quais componentes são básicos dentro de um sistema é geralmente arbitrária, depende muito da abordagem feita pelo observador que analisa o problema.
- a evolução ocorre mais rapidamente em sistemas hierárquicos do que em não-hierárquicos do mesmo tamanho, ou seja, sistemas complexos resultarão do processo evolutivo de sistemas simples mais rapidamente se existirem formas intermediárias.
- é possível realizar a distinção das interações entre sub-sistemas e dentro de sub-sistemas, o que permite tratar sub-sistemas quase como se fossem independentes um do outro.

Com base nessas características, Jennings e Wooldridge [Jennings 2000a] definem uma visão canônica de um sistema complexo (Figura 3.1). A natureza hierárquica do sistema é representada na figura através de setas de direcionamento, os subsistemas são representados por retângulos e os componentes dentro dos sub-sistemas por círculos. As interações, freqüentes ou não, podem ocorrer entre subsistemas, entre componentes de um subsistema ou entre componente de um subsistema com outro subsistema.

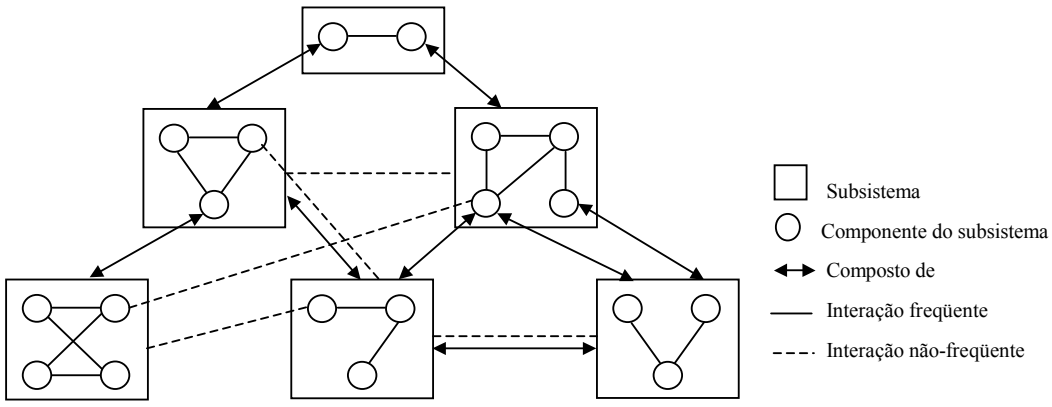


Figura 3.1: Visão de um Sistema Complexo [Jennings 2000a].

Com base nessas características da complexidade, a abordagem orientada a agentes torna-se adequada no desenvolvimento de sistemas complexos devido aos seguintes fatores:

- **decomposição:** é técnica mais básica para resolução de grandes problemas (também conhecida pelo jargão "dividir para conquistar"). Consiste de maneiras efetivas de particionar o espaço do problema de um sistema complexo em problemas menores e, potencialmente, mais facilmente gerenciáveis. Os sistemas complexos consistem de vários sub-sistemas organizados de maneira hierárquica, os quais trabalham juntos para atingir o objetivo do sistema. Em cada sub-sistema existem vários componentes que atuam com o objetivo de atingir sua funcionalidade. Uma forma de modularizar um sistema complexo é representar os componentes autônomos como vários agentes que interagem para atingir seus objetivos.

- **abstração:** é a técnica que visa ignorar os aspectos não relevantes para o propósito em questão, tornando possível uma focalização nos pontos principais. O desenvolvimento de sistemas complexos envolve uma coleção de componentes que precisam ser tratados como uma entidade única dependendo do nível de abstração. Em um primeiro nível de abstração, os sub-sistemas de um sistema complexo podem ser tratados como uma unidade conceitual única, representada através de uma sociedade de agentes. Em um nível menor, cada sub-sistema é composto por vários agentes que se comunicam para atingir seus objetivos.

- **organização:** trata de identificar e gerenciar os inter-relacionamentos entre os componentes de resolução do problema. A filosofia orientada a agentes para identificar e gerenciar os relacionamentos organizacionais é apropriada para lidar com as dependências e interações que existem em sistemas complexos, uma vez que esses sistemas envolvem uma variedade de relacionamentos organizacionais. Esses relacionamentos são importantes porque permitem que vários componentes individuais possam ser agrupados em um agente. Além disso, os mecanismos flexíveis, as estruturas, bem como o conceito de sociedade, presentes em relacionamentos organizacionais, são pontos fundamentais no paradigma de agente.

A Figura 3.2 mostra uma visão geral de um sistema multi-agentes formando uma analogia com a Figura 3.1, onde os agentes formam um relacionamento organizacional representando sub-sistemas inter-relacionados.

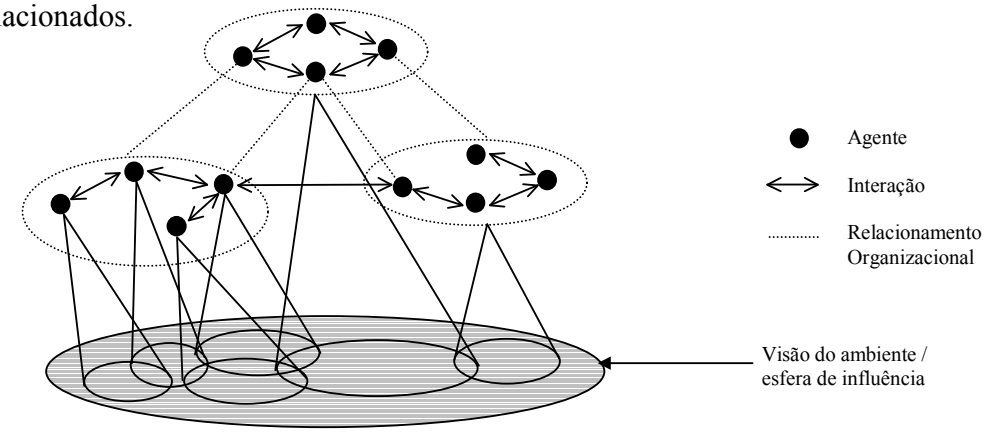


Figura 3.2: Visão de um Sistema Multi-Agentes [Jennings 2000a].

3.4 Orientação a Agentes X Orientação a Objetos

A abordagem orientada a agentes (OA) pode ser vista como um próximo passo na evolução natural das abordagens da Engenharia de Software. Isso não quer dizer que a abordagem orientada a agentes irá substituir as técnicas de orientação a objetos, de padrões de projeto (*design patterns*) ou de componentes (*component-ware*). Mais precisamente, ela deverá ser vista como uma abstração computacional de nível mais alto. É nítida a evolução dos paradigmas computacionais, que no início tinham suas bases determinadas pelas arquiteturas das máquinas e que atualmente são abstrações relacionadas efetivamente ao domínio do problema. A abordagem orientada a agentes é talvez a maneira mais natural de caracterizar muitos tipos de problemas.

O paradigma de orientação a objetos (OO) permite aos analistas, projetistas, clientes e usuários pensarem no software como uma analogia aos objetos físicos do mundo real com os quais eles têm familiaridade. Muitos autores não consideram apropriado utilizar os padrões da metodologia orientada a objetos (OO) para o desenvolvimento de aplicações orientadas a agentes (OA). O principal motivo é que o conceito de "agente" é diferente do conceito de "objeto", discutidos na Seção 2.6, além do que é necessário um maior nível de abstração quando se fala em agente. Assim, a unidade de abstração fundamental da Engenharia de Software Orientada a Objetos é o objeto, enquanto que na Engenharia de Software Orientada a Agentes a unidade primária de abstração é o agente.

Foram feitos estudos com o objetivo de verificar a existência de um paralelo entre os paradigmas OO e OA que permita a aplicação das metodologias OO para a construção de um sistema multi-agentes. Um exemplo é a adaptação da linguagem UML (*Unified Modelling Language*) proposta por Odell *et al*. [Odell 2000] para atender o paradigma orientado a agentes, denominada de AUML (*Agent Unified Modelling Language*). A UML é uma linguagem de modelagem desenvolvida para dar suporte ao desenvolvimento OO, sendo reconhecida como um padrão pela OMG (*Object Management Group*) e por grande parte dos engenheiros de software, tendo como principal vantagem a sua capacidade de acomodar todo o ciclo de vida de desenvolvimento de software. O objetivo de Odell *et al.* foi estender o UML para atender algumas características da abordagem orientada a agentes. Outros pesquisadores da área de desenvolvimento de sistemas multi-agentes têm dedicado grande esforço em propor extensões da UML com a finalidade de acomodar muitos dos aspectos dos agentes. Entretanto, como esta proposta é bastante recente, é natural que ainda não exista consenso sobre a sua real adequação para a modelagem orientada a agentes.

3.5 Ciclo de vida do desenvolvimento do software

3.5.1 Ciclo de vida clássico

O conceito de ciclo de vida de desenvolvimento do software surgiu em função da necessidade de uma padronização para o conjunto de atividades que constituem o processo de desenvolvimento do software. Os modelos do ciclo de vida descrevem as etapas do processo de desenvolvimento de sistemas e as atividades a serem realizadas em cada etapa. A definição dessas etapas e atividades possibilita prover marcos e pontos de controle para a avaliação do

sistema desenvolvido. O estudo do processo de desenvolvimento de software provocou o surgimento de várias propostas de ciclo de vida. O ciclo de vida clássico, também chamado de modelo cascata, tem um lugar definido e importante no trabalho da Engenharia de Software. Apesar de não mais ser recomendada pelas técnicas modernas, muitas técnicas iterativas utilizam as fases do modelo cascata como sub-fases internas de suas metodologias. Ele contém as seguintes fases (Figura 3.3) [Pressman 2002, Sommerville 2003, Yourdon, 1990]:

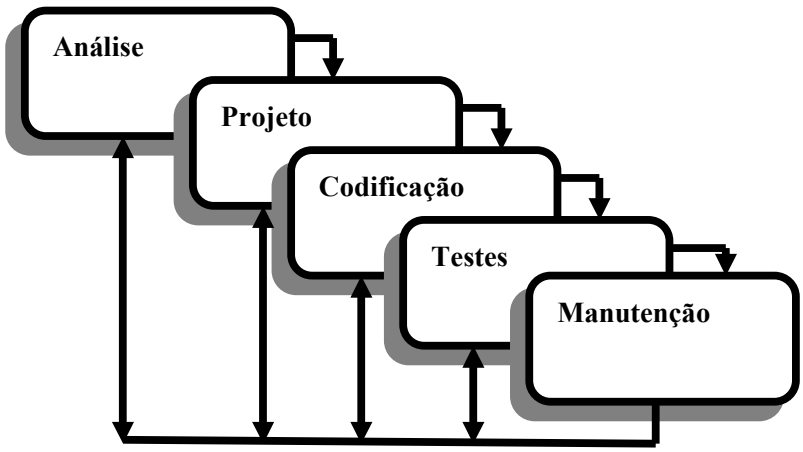


Figura 3.3: Fases do desenvolvimento de um sistema – Ciclo em Cascata.

- **Análise:** também denominada de análise de requisitos. Constitui a etapa de modelagem do problema para uma representação padronizada, resultante do processo de levantamento dos requisitos. Nesta fase são estabelecidos os requisitos do sistema a ser desenvolvido, permitindo ao engenheiro de software compreender o domínio do problema em questão. Esta fase termina quando se tem uma descrição completa do comportamento do sistema a ser construído. Os requisitos e a modelagem conceitual do sistema são documentados e devem ser revisados pelo usuário.

- **Projeto:** é um processo com vários passos que se concentra em quatro atributos importantes: estrutura de dados, arquitetura de software, detalhes procedimentais e definição da interface. Esta fase tem como objetivo traduzir os requisitos definidos na fase anterior em uma representação de software que possa ser implementado. O projeto é documentado e torna-se parte da configuração do software, sendo que quanto mais detalhado, mais clara será a fase de codificação.

- **Codificação:** envolve procedimentos de mapeamento do modelo de projeto para uma solução física, ou seja, a tradução do projeto para uma linguagem de máquina. Os programas são codificados, as bases de dados criadas e os módulos do software integrados.

- **Testes:** inicia-se após a geração do código. Serve para avaliar a qualidade do software desenvolvido, pois representa uma revisão final das fases de análise, projeto e construção. Existem diferentes estratégias de testes, sendo que as duas mais conhecidas são os testes bottom-up e top-down. A bottom-up começa por testar os

módulos pequenos de forma individual, também chamada de teste de unidade. Já a top-down presume que os módulos de execução de alto nível do sistema foram desenvolvidos, mas que os módulos de baixo nível existem apenas como simulações ou "stubs". Além disso, existem diferentes tipos de testes, entre eles podemos citar: testes funcionais (verificar se o sistema executa corretamente suas funções normais); testes de recuperação (verificar se o sistema pode recuperar-se adequadamente de vários tipos de falha, no caso de sistemas de tempo real) e testes de desempenho (verificar se o sistema pode manipular o volume de dados e transações recebidas especificadas no modelo de implementação do usuário e se apresenta o tempo de resposta necessário).

- **Manutenção:** a manutenção torna-se necessária devido a erros encontrados, ou por novas mudanças, ou por acréscimos funcionais exigidos pelo cliente. A manutenção de software reaplica cada uma das etapas precedentes do ciclo de vida a um programa existente e não a um novo.

Um processo de desenvolvimento de software muito utilizado atualmente é o *Rational Unified Process* (RUP) [Rational 2001]. O RUP provê uma abordagem genérica de projeto de Engenharia de Software para toda a vida do software. É uma abordagem muito utilizada atualmente principalmente porque cobre desenvolvimentos de diferentes tamanhos e complexidades, além da existência de ferramentas de automação para muitas de suas fases. Pode ser utilizado em várias áreas de aplicação, de diferentes tamanhos e em diferentes organizações, sempre levando em conta a natureza evolutiva e incremental do desenvolvimento do software. A Figura 3.4 apresenta o ciclo de vida de desenvolvimento do RUP, incluindo suas fases, interações e disciplinas.

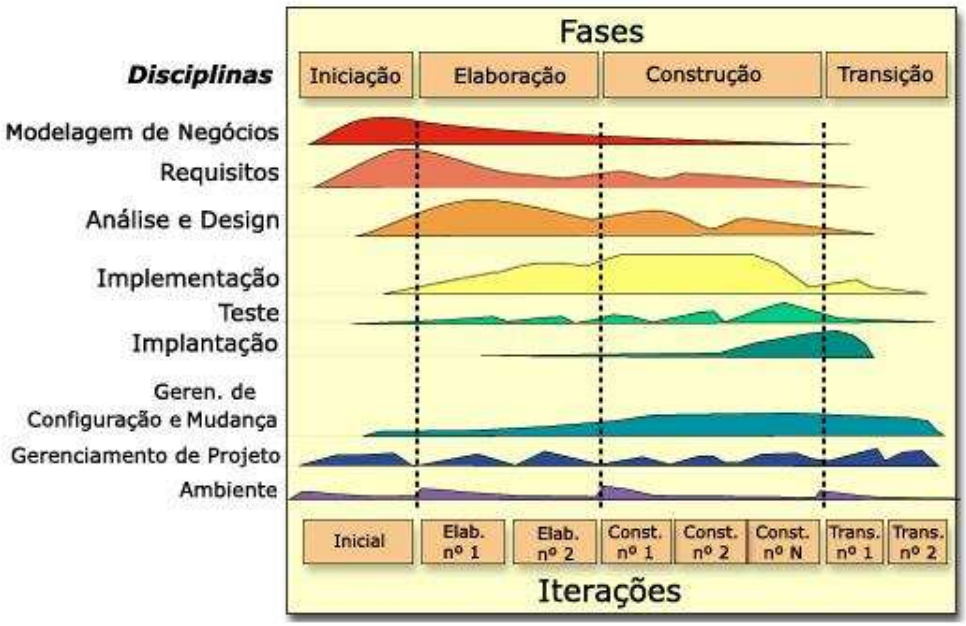


Figura 3.4: Ciclo de vida de desenvolvimento do RUP

O RUP é um processo iterativo, que pode ser analisado em duas dimensões temporais. A dimensão horizontal representa o tempo em larga escala, ao longo de diversas iterações e mostra os aspectos do ciclo de vida de um processo. A dimensão vertical representa as disciplinas do

processo, ou seja, o tempo dentro de uma mesma iteração, que agrupa logicamente as atividades da Engenharia de Software de acordo com a sua natureza. Ao longo das iterações passa-se por várias fases: Iniciação (termo adotado na tradução oficial da Rational para português), Elaboração, Construção e Transição. Cada uma dessas fases consiste de uma ou mais iterações de seqüência de atividades: Requisitos, Análise, Projeto, Implementação e Teste. Além disso, o RUP captura a maioria das boas práticas no desenvolvimento de software moderno, tais como: desenvolver software iterativamente, gerenciar requisitos, usar arquiteturas baseadas em componentes, modelagem visual do software, verificar continuamente a qualidade do software e controlar as mudanças do software.

3.5.2 O ciclo de vida na abordagem de agentes

Por se tratar de uma abordagem recente, não existe uma padronização para o ciclo de vida do desenvolvimento de sistemas orientados a agentes. As metodologias existentes definem as fases desse desenvolvimento de acordo com as seus artefatos e atividades. Por exemplo, a metodologia MaSE, que será apresentada na Seção 4.3, aborda as fases de análise e de projeto. A metodologia Prometheus, que será apresentada na Seção 4.4, aborda as fases de especificação do sistema, de projeto arquitetural e de projeto detalhado. E a metodologia Tropos, que será apresentada na Seção 4.5 aborda as fases de requisitos iniciais, de requisitos finais, de projeto arquitetural, de projeto detalhado e de implementação.

A seguir, são apresentadas algumas características do ciclo de vida do desenvolvimento do sistema orientado a agentes, de uma forma mais abrangente, envolvendo as fases de análise, projeto e codificação, para facilitar o entendimento:

- **Análise:** deve-se verificar o objetivo e o comportamento desejado do sistema. São necessários conceitos e ferramentas metodológicas que tratem o sistema como um todo, composto por agentes com características específicas. Os conceitos e métodos devem ser bem fundamentados para representar os aspectos dinâmicos do sistema, sua localidade, bem como, o estado mental do agente, suas crenças, seus desejos e suas intenções. Nesta fase, são identificados os papéis que os agentes devem ter para solucionar o problema e quais os recursos disponíveis.

- **Projeto:** envolve a especificação da arquitetura de software do sistema de agente e dos agentes dentro do sistema, ou seja, a arquitetura do sistema como um todo, dos agentes individuais, dos componentes e de outras estruturas dentro do agente. Transforma os objetivos de alto nível e os papéis em tipos de agentes concretos, identificando quais agentes são responsáveis por quais papéis. Transforma as interações de alto nível em protocolos de interação específicos. Heurísticas e diretrizes são necessárias para suportar o projetista durante esta fase, principalmente porque auxiliam na transformação das entidades da fase de análise em entidades computacionais.

- **Codificação:** a codificação, também conhecida como implementação, pode ser derivada da fase de projeto através da geração automática de código, ou feita manualmente. O ideal seria reusar componentes de bibliotecas de agentes, porém por se tratar de uma área recente, ainda não existem muitas bibliotecas disponíveis. Nesta

fase, são necessários métodos de rastreamento bem definidos, que verifiquem a consistência entre o modelo conceitual e sua implementação.

3.6 Metodologias orientadas a agentes

Metodologias são os meios fornecidos pela Engenharia de Software para facilitar o processo de desenvolvimento do software e, conseqüentemente aumentar a qualidade do software. Portanto, por definição, a metodologia da Engenharia de Software é um conjunto estruturado de conceitos, diretrizes ou atividades para auxiliar no entendimento do desenvolvimento do software [Dam 2003].

É necessário que, a exemplo do paradigma orientado a objetos (OO), o paradigma orientado a agentes (OA) também seja suportado por técnicas e métodos que garantam a qualidade do processo de produção de software.

Atualmente existe um grande aumento nas pesquisas envolvendo a utilização de agentes na Engenharia de Software. Os métodos, muitos na fase inicial, tentam explorar os principais conceitos de agentes nos vários estágios do ciclo de vida do desenvolvimento de software. Uma característica comum dessas pesquisas é que a maioria dessas metodologias está sendo desenvolvida a partir de outras metodologias existentes. Basicamente essas metodologias são divididas em dois grupos:

- **extensões das metodologias orientadas a objetos**: são as metodologias que estendem ou adaptam as metodologias orientadas a objetos para a orientação a agentes. Seus principais benefícios são: (i) capturam algumas similaridades entre objetos e agentes; (ii) como as metodologias orientadas a objetos são muito utilizadas, uma extensão para a orientação a agentes terá um aprendizado mais fácil e (iii) as técnicas de identificação de objetos podem ser úteis na identificação de agentes [Iglesias 1999].

- **extensões das técnicas da engenharia do conhecimento**: as metodologias da engenharia do conhecimento são úteis para a abordagem orientada a agentes, pois lidam com o desenvolvimento de sistemas baseados em conhecimento e podem modelar os estados mentais dos agentes. Seus principais benefícios são: (i) é útil para modelar o conhecimento do agente; (ii) a definição do conhecimento do agente pode ser visto como um processo de aquisição do conhecimento; (iii) a engenharia do conhecimento tem sido aplicada com grande sucesso em vários projetos de software. Entretanto, as metodologias da engenharia do conhecimento não tratam os aspectos sociais dos agentes, nem suas atitudes reflexivas ou orientadas a agentes [Iglesias 1999].

A existência de um grande número de metodologias torna-se um grande obstáculo na adoção comercial e nas indústrias. Uma abordagem consolidada, através da combinação da melhores práticas das metodologias existentes, servirá como um avanço na tecnologia de agentes.

A Tabela 3.1 mostra as principais metodologias orientadas a agentes existentes na literatura atualmente. Essas metodologias estão agrupadas de acordo com a sua categoria, ou seja, se representam extensões das técnicas orientadas a objetos (OO) ou extensões das técnicas da engenharia do conhecimento (EC).

Tabela 3.1: Metodologias da Engenharia de Software Orientada a Agentes. Adaptado de Dam [Dam 2003].

Metodologia	**Autor**	**Categoria**	**Referência**
AAII	Kinny, Georgeff & Rao	EC	[Kinny 1996, Rao 1995]
ADEPT	Jennings et al.	EC	[Jennings 2000b]
AO methodology for enterprise modeling	Kendall et al.	OO	[Kendall 1995]
AOR	Wagner	OO	[Wagner 2000, 2003]
AUML	Odell et al.	OO	[Odell 2000]
Cassiopeia	Collinot et al.	OO	[Collinot 1996]
DESIRE	Brazier et al.	OO	[Brazier 1997]
GAIA	Wooldridge et at.	OO	[Wooldridge 2000, Zambonelli 2003]
OPEN Agents	Debenham & Hendersom	OO	[Debenham 2002]
MaSE	DeLoach & Wood	OO	[DeLoach 2001, DeLoach 2003, Wood 2001]
MAS-CommonKADS	Iglesias et al.	EC	[Iglesias 1996]
MASSIVE	Lind	OO	[Lind 2000]
MESSAGE	Caire et al.	OO	[Caire 2001]
NFR	Chung	EC	[Chung 2000]
PASSI	Cossentino et al.	OO	[Burrafato 2002]
Prometheus	Padgham & Winikoff	OO	[Padgham 2002a, 2002b, 2002c, 2005]
Styx	Bush et al.	OO	[Bush 2001]
Tropos	Mylopoulos et al.	EC	[Bresciani 2002, Bresciani 2004, Castro 2002, Giunchiglia 2002, Mylopoulos 2002, Perini 2001, Sannicolo 2002].

3.6.1 Classificação das metodologias

Existem várias maneiras de classificar as metodologias orientadas a agentes na Engenharia de Software. Weiss [Weiss 2002] apresenta alguns possíveis critérios. O primeiro critério está relacionado às fases do processo de desenvolvimento coberto pela metodologia. A maioria das

metodologias existentes atualmente trata somente as fases de análise e projeto, poucas incluem a fase de implementação. O segundo critério é o nível de modelagem utilizado pela metodologia, enfatizando o nível intra-agente (componentes individuais do agente), o nível inter-agente (protocolos de coordenação e comunicação) e o nível supra-agente (estruturas organizacionais, normas e leis sociais). O terceiro critério está relacionado com a direção do desenvolvimento, ou seja, desenvolvimento *bottom-up* ou *top-down*. O quarto critério é a generalidade (tecnológica ou de aplicação). Algumas abordagens são menos gerais que outras, pois são projetadas para suportar arquiteturas de agentes específicos, como por exemplo, arquitetura BDI (*belief, desire and intention*) [Rao 1995]. Finalmente, o quinto critério de classificação é o nível de granularidade, ou seja, o nível de detalhe considerado pela abordagem.

3.6.2 Linguagem

Além das metodologias apresentadas anteriormente, também existem linguagens específicas para o desenvolvimento de agentes. Basicamente estas linguagens estão agrupadas em linguagens para programação orientada a agentes e linguagens para comunicação e coordenação entre agentes.

- **Linguagem para programação orientada a agentes**: A maioria dos sistemas baseados em agentes está sendo escrita em Java ou C++. Porém, estão sendo propostas novas linguagens elaboradas especificamente para atender a tecnologia de agentes. Um exemplo é a linguagem AgentSpeak(L) [Rao 1996]. Ela é linguagem baseada em regras, possuindo uma semântica operacional formal e assume que os agentes possuem intenções, desejos, eventos arquivados e regras de planos.

- **Linguagem para comunicação e coordenação**: As linguagens de comunicação de agentes mais comuns são: KQML (*Knowledge Query and Manipulation Language*), talvez a mais utilizada; FIPA-ACL (FIPA *Agent Communication Language*), é uma linguagem de comunicação muito influenciada pela ARCOL (ARTIMIS *Communication Language*). Juntos FIPA-ACL, ARCOL e KQML praticamente estabelecem um padrão para as linguagens de comunicação de agentes. Além dessas linguagens, existem outras, como por exemplo, o AgentTalk [Kuwabara 1995], que é uma linguagem de descrição do protocolo de coordenação para sistemas multi-agentes. AgentTalk suporta customização, definição incremental dos protocolos de interação. O principal problema com as linguagens de comunicação está na necessidade de uma definição única, sem dialetos diferentes o que inviabiliza a comunicação entre sistemas multi-agentes.

3.6.3 Ferramentas de Desenvolvimento e Plataformas

Algumas ferramentas e plataformas estão disponíveis para suportar as atividades ou fases do processo de desenvolvimento de software orientado a agentes, sendo que várias foram construídas utilizando a linguagem Java. Embora a maioria dessas ferramentas e plataformas

tenha seu foco na implementação do sistema, algumas também suportam as fases de análise e de projeto.

- **Jack** [Busetta 1999] – é uma ferramenta comercial desenvolvida por *Agent Oriented Software Pty.Ltd*. da Austrália e utiliza os conceitos de BDI (*belief, desire, intention*).

- **Jade** [Bellifemine 2003] – *Java Agent Development Framework*, foi desenvolvido pela Universidade de Parma, na Itália. Consiste de conjunto de classes que implementam um sistema de gerenciamento do agente, um facilitador de diretório e um canal de comunicação de agente.

- **Zeus** [Nwana 1999] – é uma ferramenta para a construção de agentes, desenvolvida por *British Telecom Intelligent System Research Lab*. Possui vários utilitários que orientam no desenvolvimento de sistemas de agentes. É implementado como uma coleção de classes Java e pode ser dividido em três componentes: (i) biblioteca de componente de um agente (forma os blocos construtivos de agentes individuais); (ii) software de construção do agente (coleção de ferramentas acessadas através de interfaces gráficas); (iii) ferramentas de visualização da sociedade de agentes.

3.7 Cuidados com a abordagem orientada a agentes

Alguns cuidados devem ser levados em consideração quando se deseja utilizar a tecnologia de agentes para a construção de um sistema de software. Embora os agentes estejam sendo utilizados em uma grande variedade de aplicações, eles não podem ser considerados como uma solução universal. Existem muitas aplicações em que os paradigmas convencionais de desenvolvimento de software são mais apropriados, como por exemplo, a abordagem orientada a objetos.

Jennings e Wooldridge [Jennings 2001] fazem um levantamento dos problemas encontrados no desenvolvimento de sistemas utilizando abordagem orientada a agentes. Este levantamento não é suficiente para garantir o sucesso de um projeto de desenvolvimento baseado em agentes, mas auxilia na detecção de algumas falhas evidentes durante a sua construção. Uma das principais origens desses problemas está simplesmente no fato que os desenvolvedores superestimam o potencial dos sistemas de agente. Alguns gerentes de projetos que propõem um projeto de agentes não têm a exata idéia sobre agentes. Eles não têm o modelo de negócio para os agentes, ou seja, não têm o entendimento de como os agentes podem ser usados para diferenciar seus produtos.

Quando se constrói uma aplicação de agentes, existe uma tentação compreensível em concentrar a atenção nos aspectos da "inteligência" da aplicação. O resultado é freqüentemente um *framework* de agente sobrecarregado com técnicas experimentais.

Outro problema que deve ser bem analisado está relacionado com a quantidade de agentes dentro do sistema. Enquanto alguns projetistas utilizam um agente para cada tarefa, outros parecem não reconhecer o valor da abordagem multi-agentes. Eles criam um sistema que não explora o poder do paradigma de agente e desenvolvem uma solução com um número muito pequeno de agentes executando várias tarefas. O resultado fica mais parecido com um programa orientado a objeto em que todas as funcionalidades estão alocadas dentro de uma única classe.

Silva [Silva 2003] apresenta alguns obstáculos identificados que devem ser superados para que a Engenharia de Software Orientada a Agentes se torne popular:

- Relacionamento do paradigma orientado a agentes e outros paradigmas: não está claro como o desenvolvimento de sistemas orientados a agentes irá coexistir com outros paradigmas de software, como a orientação a objetos.
- Metodologias orientadas a agentes: embora existam várias metodologias orientadas a agentes, há pouco consenso entre elas e nenhum acordo sobre os tipos de conceitos que a metodologia deveria suportar.
- Engenharia de escalabilidade: é preciso entender melhor como fazer engenharia de forma segura e previsível em sistemas que consistem de um grande número de agentes interagindo dinamicamente uns com os outros a fim de atingir suas metas.

3.8 Resumo

A abordagem orientada a agentes é recomendada para o desenvolvimento de sistemas distribuídos e complexos. Os softwares complexos são compostos por várias partes independentes com muitas interações. A ESOA tem como objetivo fornecer métodos, técnicas e ferramentas para facilitar o desenvolvimento desses softwares.

Neste capítulo, foram feitas uma revisão da literatura da Engenharia de Software Orientada a Agentes e uma análise da utilização de agentes em sistemas complexos. Um ponto importante apresentado foi a diferença entre a Orientação a Agentes e a Orientação a Objetos. Foi elaborada uma comparação entre o ciclo de vida clássico do desenvolvimento de software, o RUP (*Rational Unified Process*) e as fases mínimas necessárias na abordagem orientada a agentes. Foram apresentadas algumas linguagens, ferramentas de desenvolvimento e plataformas das metodologias orientadas existentes atualmente. Finalmente foram apresentados alguns cuidados necessários ao se adotar a abordagem orientada a agentes no desenvolvimento de sistemas.

Pode-se perceber que devido ao grande aumento da necessidade de ferramentas para suportar o desenvolvimento de sistemas orientados a agentes, os pesquisadores continuam se esforçando para criar técnicas cada vez mais eficientes e que atendam as novas exigências dos usuários.

Capítulo 4

Metodologias da Engenharia de Software Orientada a Agentes

4.1 Introdução

Neste capítulo apresentamos algumas das metodologias da Engenharia de Software Orientada a Agentes mais discutidas na literatura: MaSE (*Multiagent Systems Engineering Methodology*), Prometheus e Tropos, além da linguagem de modelagem AUML (*Agent Unified Modeling Language*). Essas metodologias foram selecionadas baseadas em vários fatores, como a abrangência na Engenharia de Software Orientada a Agentes, a difusão na comunidade de pesquisadores e os recursos disponíveis pelas metodologias como documentação detalhada e de fácil acesso e suporte à ferramenta. Cada uma das metodologias (e também a linguagem AUML) apresenta pontos fortes, pontos fracos e diferentes especialidades para atender aos aspectos do domínio das aplicações existentes. A AUML, uma extensão da UML, fornece ferramentas para especificar protocolos de interações entre agentes e representar o comportamento interno do agente. A metodologia MaSE é usada na modelagem de sistema multi-agentes. A metodologia Prometheus foca na construção de agentes e suas funcionalidades utilizando a abordagem BDI (*belief, desire and intention*). Tropos é uma metodologia dirigida a requisitos, onde os agentes possuem propriedades intencionais como objetivos, crenças e responsabilidades.

A Seção 4.2 apresenta inicialmente a linguagem AUML, cujos diagramas são utilizados em outras metodologias baseadas em agentes. A Seção 4.3 descreve a metodologia MaSE. A Seção 4.4 apresenta a metodologia Prometheus e a Seção 4.5, a metodologia Tropos. Na Seção 4.6 são descritos critérios relacionados à avaliação de metodologias orientadas a agentes. E finalmente, é apresentado um resumo do capítulo.

4.2 *Agent Unified Modeling Language* - AUML

Agent Unified Modeling Language (AUML) é uma extensão da linguagem de modelagem UML (*Unified Modeling Language*), proposta por Odell *et al*. [Odell 2000], concebida para dar suporte ao desenvolvimento de sistemas orientados a agentes. Essa extensão visa exclusivamente estender a linguagem UML de modo a dar suporte à especificação de protocolos de interação entre agentes, não se preocupando em fornecer artefatos para todo o ciclo de vida do desenvolvimento de um sistema orientado a agentes. Os diagramas AUML são utilizados em diversas metodologias orientadas a agentes, principalmente nas que surgiram como uma especialização das abordagens orientadas a objetos. O objetivo da AUML é fornecer uma semântica semi-formal e intuitiva através de uma notação gráfica amigável para o

desenvolvimento de protocolos de interação entre agentes, em sistemas orientados a agentes. De acordo com a AUML, a interação entre agentes pode ser definida em um modelo com três camadas: a primeira camada contendo definições gerais de protocolos de interação (ou *patterns* de interação), a segunda camada, representando as interações particulares entre agentes e finalmente, na terceira camada, detalhando-se o comportamento interno de cada agente.

No nível macro da metodologia, ou primeira camada, são modelados os protocolos de interação de agentes (AIP – *Agent Interaction Protocol*). Segundo Odell *et al*. [Odell 2000], um protocolo de interação entre agentes descreve um padrão de comunicação na forma de uma seqüência permitida de mensagens entre agentes e as restrições sobre o conteúdo destas mensagens. Os fluxos de controle e informação são considerados atos de comunicação (*communication act* – CA). Os artefatos disponíveis na AUML para a modelagem de protocolos como um todo são os pacotes e os templates. A AUML estende a UML utilizando o conceito de pacote não somente para agregar diagramas de classes, mas para agregar diagramas genéricos. Na UML pacotes representam abstrações de diagramas de classes. Na AUML, pacotes podem representar abstrações de diagramas quaisquer. Especificamente neste caso, um pacote é utilizado para representar um diagrama de seqüência, que descreve um protocolo de interação entre agentes. De uma maneira mais geral, quando os agentes são identificados pelos papéis que representam e não diretamente por seus nomes, podem-se utilizar *templates* (Figura 4.1). Nominando-se os agentes que realizam os papéis indicados no *template*, obtemos os pacotes que descrevem os protocolos de interação.

A Figura 4.1 mostra um protocolo de interação genérico, modelado na forma de um *template*. Observe-se que não são indicados os agentes participantes da interação, mas tão somente os papéis que estes assumem. Estes papéis, Iniciador e Participante, estão indicados no retângulo tracejado no canto superior direito, indicando-se que se trata de um *template*. O pacote encapsula um diagrama de seqüência, onde Iniciador e Participante indicam os agentes que irão exercer estes papéis. Além dos nomes dos papéis, o *template* especifica restrições possíveis durante a seqüência de interações (no caso, a existência de prazos) e as ações de comunicação presentes no protocolo de comunicação (chamada-de-proposta, recusa, etc).

De acordo com a segunda camada AUML, é possível representar-se não protocolos genéricos de interação como no caso da primeira camada, mas as interações particulares entre agentes. Para tal, utilizam-se Diagramas de Interação UML (**Diagramas de Seqüência** e **de Colaboração)** com algumas modificações e, em alguns casos, Diagramas de Atividade e/ou Diagramas de Estado. Dentre estas extensões, algumas são mínimas, tais como referir-se a *agentes* e não a *objetos*, e a *atos de comunicação* (*CA – Communication acts*) ao invés de *mensagens.*

Um exemplo de um **Diagrama de Seqüência** utilizado para representar a comunicação entre agentes é mostrado na Figura 4.2. De uma maneira genérica, a caixa retangular pode expressar tanto um único agente como um conjunto de agentes, executando um papel específico. De maneira alternativa, pode-se indicar somente o papel sendo exercido pelo agente e/ou o mesmo agente pode aparecer em diferentes caixas, executando diferentes papéis. A sintaxe genérica utilizada pela AUML é: nome-agente/papel: classe, onde *classe* se refere ao tipo do agente e *papel* identifica o papel ou a função específica que um agente desempenha durante a interação.

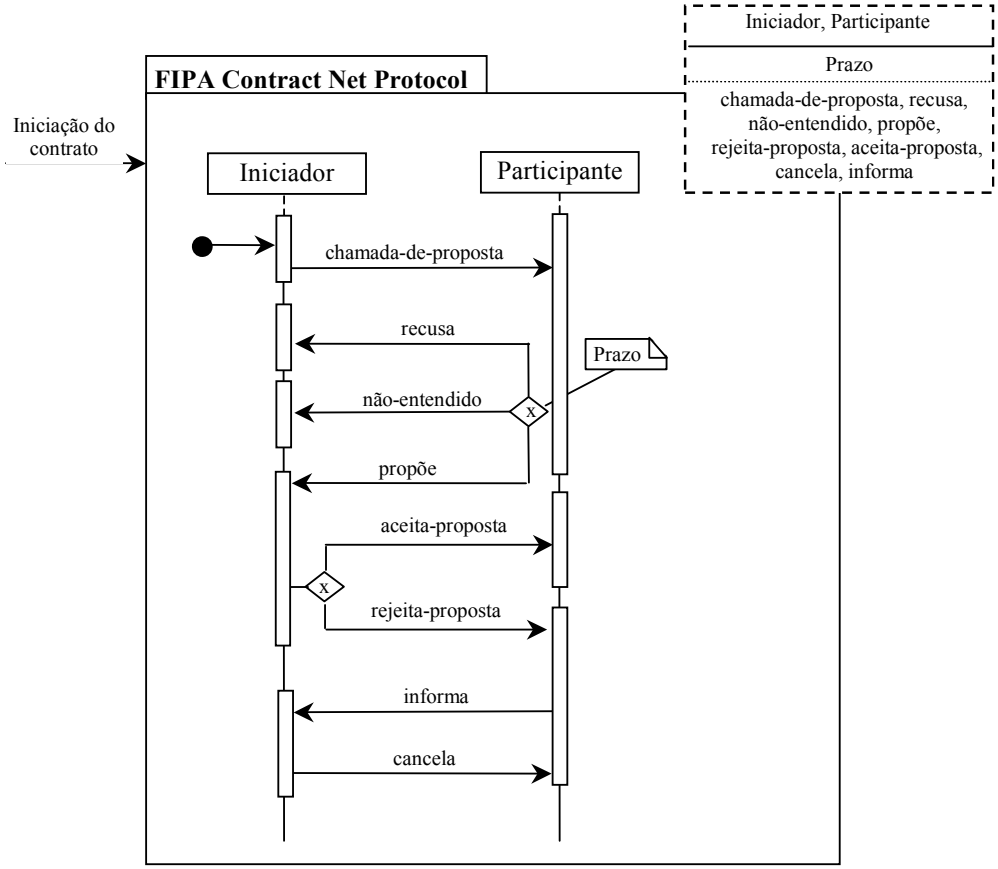


Figura 4.1: Um Protocolo de Interação entre Agentes genérico expresso como um template.

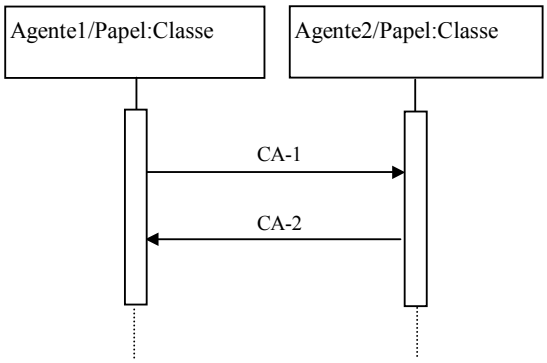


Figura 4.2: Formato Básico para a Comunicação do Agente.

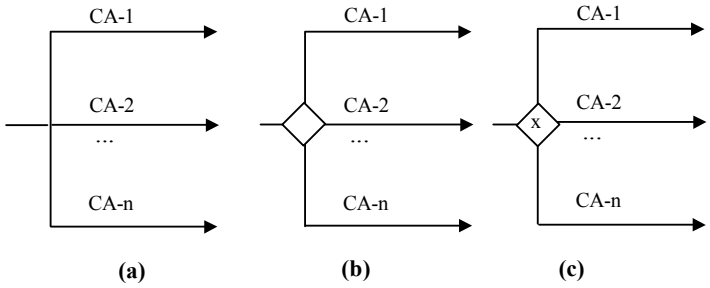


Figura 4.3: Extensões recomendadas para suportar threads concorrentes: (a) indica que todas as threads CA-1 até CA-n são concorrentes; (b) a caixa de decisão indica qual CA (zero ou várias) será enviada; (c) indica que exatamente uma CA será enviada.

AUML também estende a comunicação assíncrona e concorrente da UML, suportando *threads* concorrentes de interação. A Figura 4.3 mostra três maneiras de expressar várias *threads*. A Figura 4.3 (a) indica que todas as *threads* CA-1 a CA-n são enviadas concorrentemente. A Figura 4.3 (b) incluiu uma caixa de decisão indicando qual dos CA (zero ou mais) será enviada. A Figura 4.3 (c) indica que uma e somente uma CA será enviada.

Outras extensões foram propostas especificamente para o **Diagrama de Colaboração**. Nestas extensões, um mesmo agente exercendo papéis distintos pode aparecer em lugares diferentes, sendo que a transição de um papel para o outro é indicada por meio de arcos estereotipados. A Figura 4.4 mostra um Diagrama de Colaboração composto por três agentes: A, B e C. O agente B assume dois papéis distintos: Contratante e Analisador; o agente A assume os papéis de Cliente e Negociador e, finalmente, o agente C assume o papel de Competidor e Contratante1.

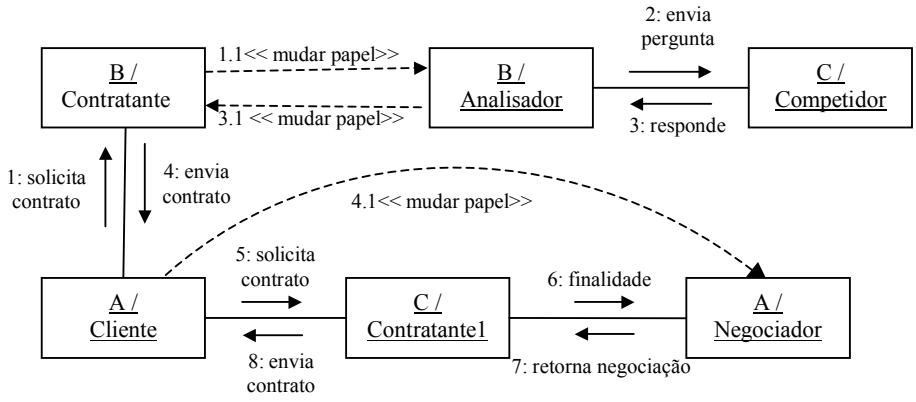


Figura 4.4: Exemplo de um diagrama de colaboração mostrando a interação entre agentes com vários papéis.

Além dos Diagramas de Interação, pode-se também utilizar **Diagramas de Atividades** para modelar as interações entre agentes, principalmente protocolos de interações complexos que envolvam processamento concorrente. Na Figura 4.5, as atividades dos agentes estão indicadas nos diferentes *swimlanes*[3]. Os agentes executam uma seqüência de atividades que resultam tanto na liquidação do pedido quanto na atualização de cota.

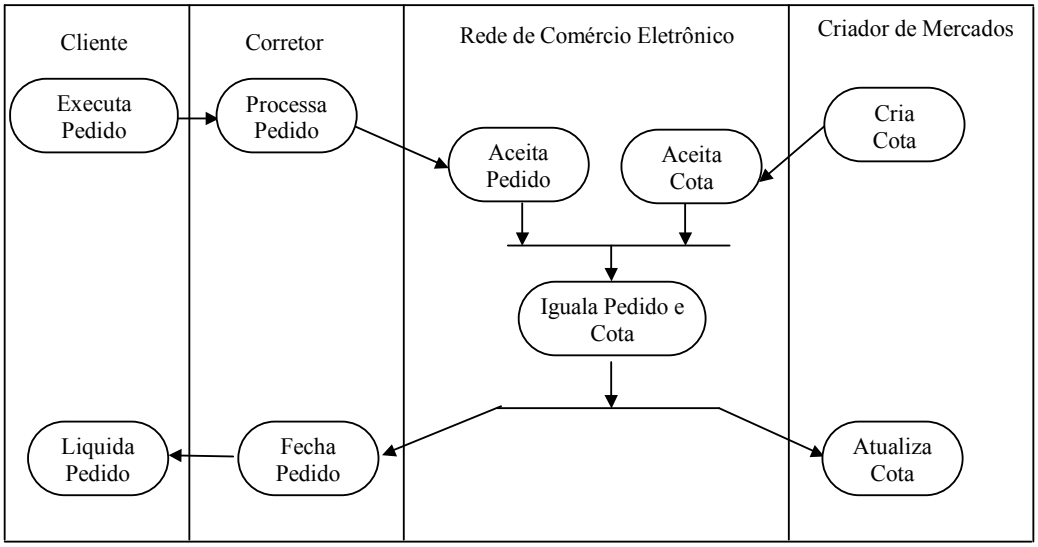


Figura 4.5: Exemplo de um diagrama de atividade que mostra um protocolo para venda entre vários agentes [Odell 2000].

Uma extensão nos diagramas de atividades proposta pela AUML, supondo que eventualmente o número de agentes pode ser maior do que as atividades realizadas por cada agente, elimina os *swimlanes*, e demonstra abaixo de cada atividade qual é o agente que a executa. Por exemplo, na Figura 4.6, o Agente Vendedor aceita, monta, envia e fecha o pedido. A execução começa no estado de partida, representado por um círculo vazado e termina no estado final, representado por um círculo preenchido.

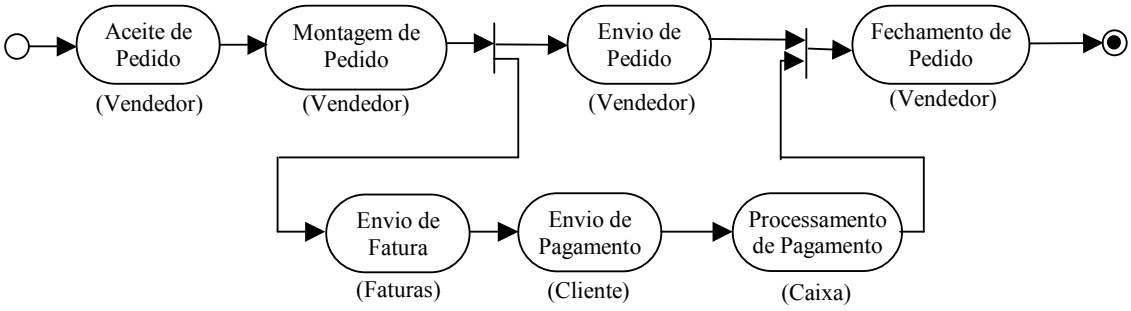


Figura 4.6: Exemplo de um diagrama de atividade com as atividades e o papel do agente [Odell 2000].

[3] *Swimlanes* são colunas que organizam as atividades de cada agente.

Diagramas de Estados são equivalentes a máquinas de estados finitos, sendo recomendados para enfatizar as restrições do protocolo. Os estados são representados por retângulos com bordas arredondadas, enquanto as transições são arcos direcionados que conectam seus estados. A Figura 4.7 mostra um Diagrama de Estados para um protocolo de Pedido. Por exemplo, se um dado pedido está no estado *Requisitado*, o agente fornecedor (B) pode comprometer a negociação requisitada, resultando em uma transição para o estado *Comprometido*. O estado *Requisitado* possui outras duas possíveis ações para o estado de *Comprometido*: o agente fornecedor (B) pode recusar o pedido e o agente cliente (A) pode confirmar. Além disso, o agente fornecedor (B) pode recusar um pedido nos estados *Proposto* ou *Requisitado*.

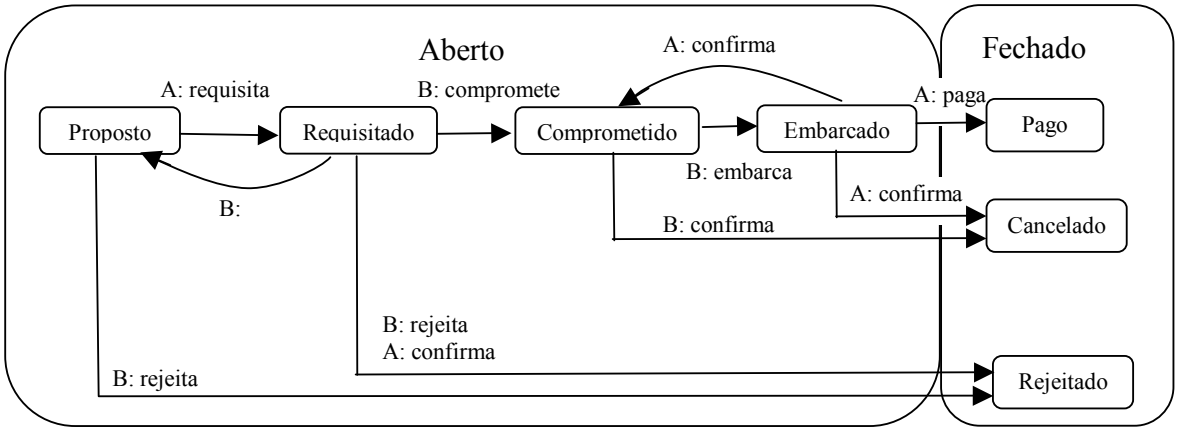


Figura 4.7: Diagrama de Estado indicando os estados e transições válidas para o protocolo Pedido – adaptado de Odell [Odell 2000].

Na última camada da AUML, são utilizados o **Diagrama de Atividades** e o **Diagrama de Estados** para detalhar o comportamento interno de um único agente, quando este estiver executando protocolos. O Diagrama de Estados adota a mesma notação utilizada na camada anterior, porém abordando os estados de um único agente. O Diagrama de Atividades descreve o funcionamento interno de um agente apresentando as etapas que o agente executa do início ao fim. Na Figura 4.8 o processamento detalhado do agente Vendedor inicia-se com o pedido e termina com a venda completada. As caixas pontilhadas representam que a atividade é realizada por um agente externo, como por exemplo, a atividade Envio de Fatura. A execução começa no estado de partida, representado por um círculo vazado e termina no estado final, representado por um círculo preenchido.

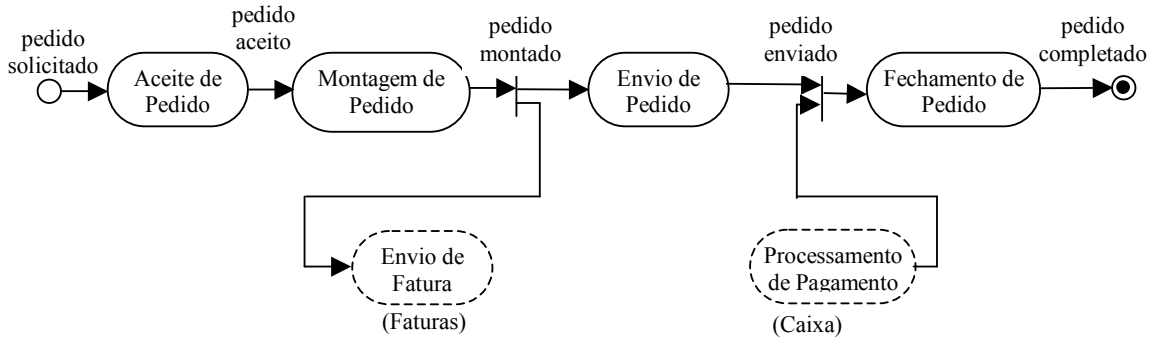


Figura 4.8: Diagrama de Atividades que especifica o comportamento do agente Vendedor [Odell 2000].

4.3 Metodologia MaSE

MaSE (*Multiagent Systems Engineering Methodology*) é uma metodologia orientada a agentes, desenvolvida pela *Air Force Institute of Technology* (AFIT) [Wood 2001, DeLoach 2001, DeLoach 2003]. Na metodologia MaSE, os agentes não são considerados como sendo necessariamente autônomos, pró-ativos, etc.; são processos de software simples que interagem uns com os outros para atingir o objetivo geral do sistema. MaSE assume a existência prévia da especificação de requisitos para o início do desenvolvimento da metodologia e segue de forma estruturada até a sua implementação, combinando vários modelos pré-existentes em uma única metodologia estruturada. O objetivo principal da MaSE é guiar o projetista através de todo o ciclo de vida do software, independentemente de qualquer arquitetura multi-agentes, arquitetura de agente, linguagem de programação ou sistema de troca de mensagens. MaSE possui uma ferramenta denominada AgentTool que auxilia no processo de modelagem.

4.3.1 Fases da Metodologia

A metodologia consiste de duas fases. A fase de análise tem os seguintes passos: captura dos objetivos do sistema, aplicação de caso de uso e refinamento de papéis. A fase de projeto é dividida nos seguintes passos: criação de classes de agente, construção de conversa entre os agentes, montagem de classes de agentes e projeto do sistema. A Figura 4.9 mostra as fases da metodologia MaSE.

4.3.1.1 Fase de Análise

A fase de análise inicia na captura dos objetivos do sistema. Depois de definir os objetivos de alto-nível do sistema, os casos de uso são extraídos e em seguida, modelados no Diagrama de Seqüência, no passo de aplicação de caso de uso. Este passo define um conjunto inicial de papéis do sistema e os caminhos de comunicação. A partir das metas do sistema e dos papéis identificados nos casos de uso, esse conjunto inicial de papel é refinado e estendido e as tarefas para realizar cada objetivo são definidas no passo de refinamento dos papéis. Cada um desses passos é detalhado a seguir:

- **Captura dos objetivos:** O primeiro passo da metodologia MaSE é a captura dos objetivos do sistema como um todo. Esse passo é dividido em duas partes, sendo que primeiramente o analista identifica um conjunto de objetivos do sistema, abstraído a partir da especificação inicial do sistema. A MaSE assume a existência prévia de uma representação textual dos requisitos, não possuindo nenhuma técnica específica para o levantamento dos requisitos junto aos usuários. Em seguida, esses objetivos são organizados em uma hierarquia, pelo grau de importância dentro do sistema, levando a uma visualização semelhante a uma árvore de desdobramento, denominado de **Diagrama Hierárquico de Objetivos**. Neste diagrama, cada nível da hierarquia possui objetivos com o mesmo valor e os objetivos do nível inferior (objetivo-filho) são

necessários para satisfazer os objetivos do nível superior (objetivo-pai). Na Figura 4.10, o Diagrama Hierárquico de Objetivos possui como principal objetivo "1. Detectar e notificar ao administrador as violações do servidor". Esse objetivo por sua vez é composto de dois sub-objetivos (objetivos-filho): "1.1 Detectar e notificar ao administrador as violações no sistema de arquivos" e "1.2 Detectar e notificar ao administrador as violações de login". Cada um desses sub-objetivos é decomposto em outros sub-objetivos até que todos os objetivos do sistema sejam detalhados.

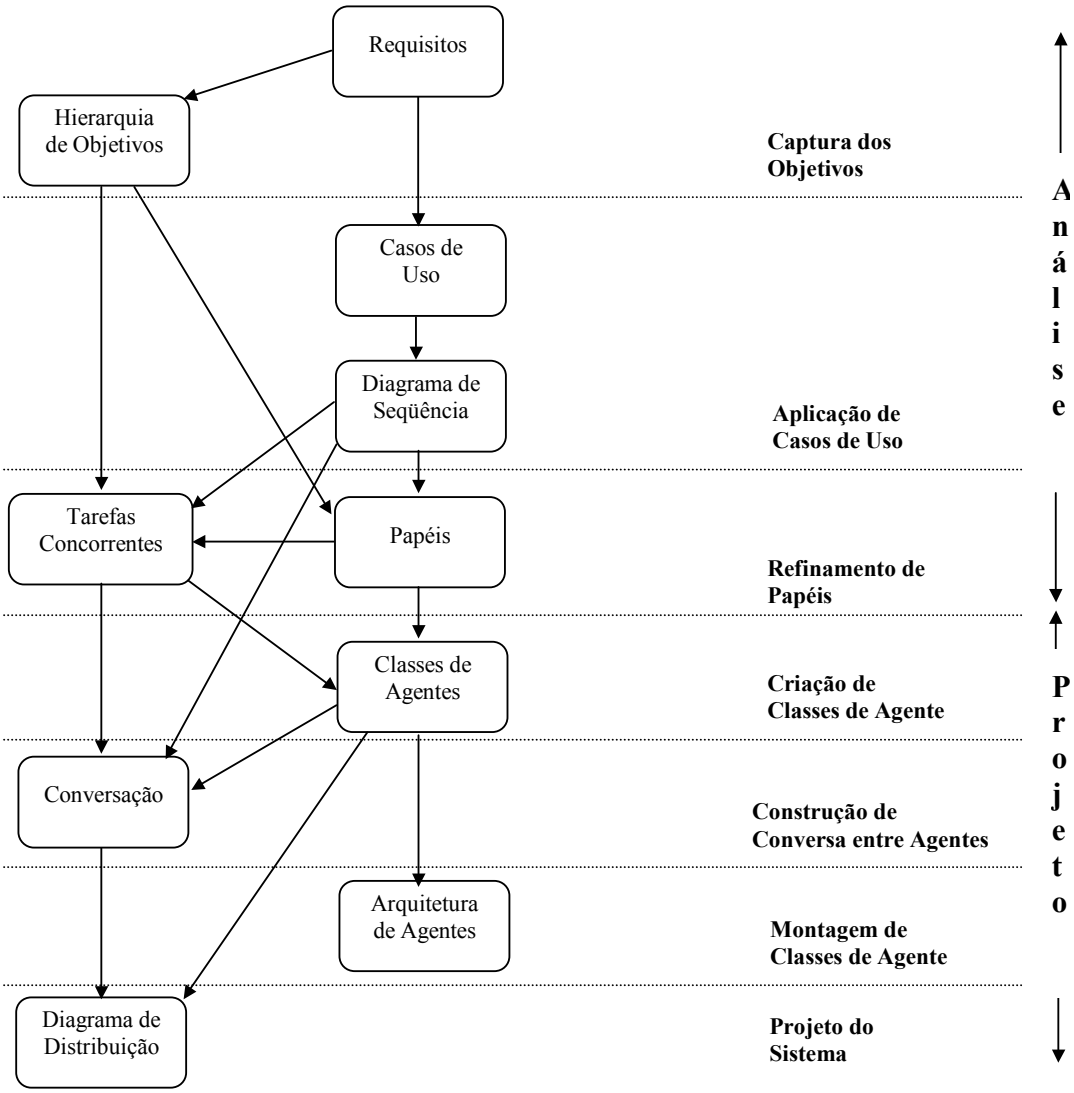


Figura 4.9: Metodologia MaSE [Wood 2001].

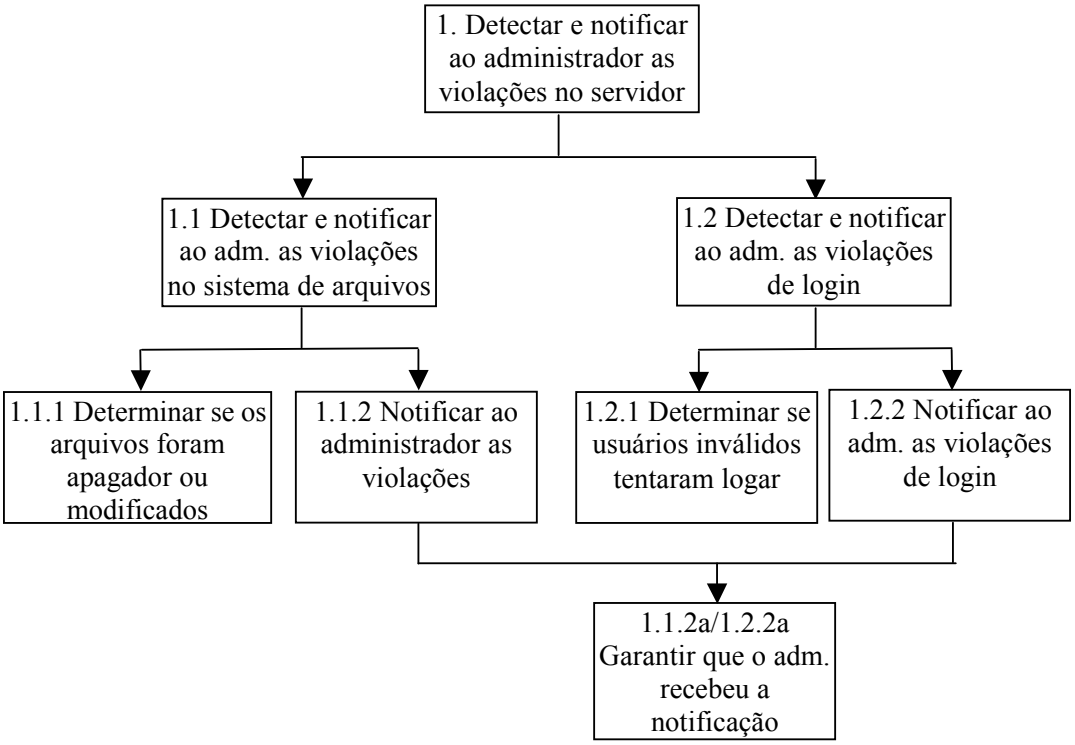


Figura 4.10: Diagrama Hierárquico de Objetivos- adaptado de DeLoach [DeLoach 2001] .

- **Aplicação de Casos de Uso:** é um passo importante na transformação de objetivos em papéis e tarefas associadas. Esta etapa consiste de duas atividades. Na primeira, são identificados os Casos de Uso, elaborados a partir dos requisitos do sistema. Os Casos de Uso são descrições narrativas de uma seqüência de eventos que define o comportamento desejado do sistema, ou seja, são exemplos de como o sistema deve se comportar em um dado caso. A Figura 4.11 mostra o exemplo de um Caso de Uso para o ator "Notificador".

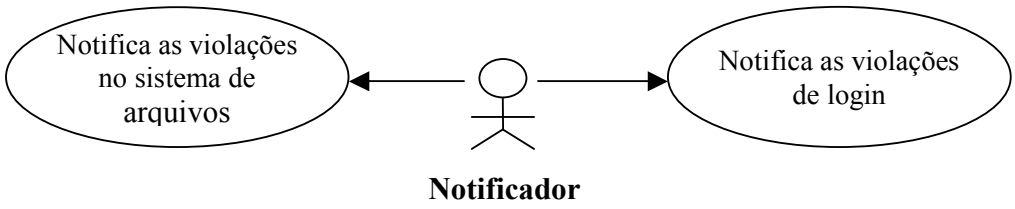


Figura 4.11: Exemplo de Caso de Uso

Na segunda atividade desse passo, são aplicados os Casos de Uso. Primeiramente, um conjunto inicial de papéis é identificado baseados nos objetivos e nos Casos de Uso. Em seguida, esses Casos de Uso são reestruturados através do **Diagrama de Seqüência**, conforme a Figura 4.12. Esse diagrama equivale ao Diagrama de Seqüência proposto na

AUML (Seção 4.2), onde as entidades representadas pelos diagramas são os papéis assumidos pelos agentes e não objetos como na orientação a objetos. Os Diagramas de Seqüência mostram a seqüência de eventos entre múltiplos papéis e determinam a comunicação mínima que deve existir entre eles. Os papéis identificados neste passo formam um conjunto inicial de papéis que serão refinados no próximo passo.

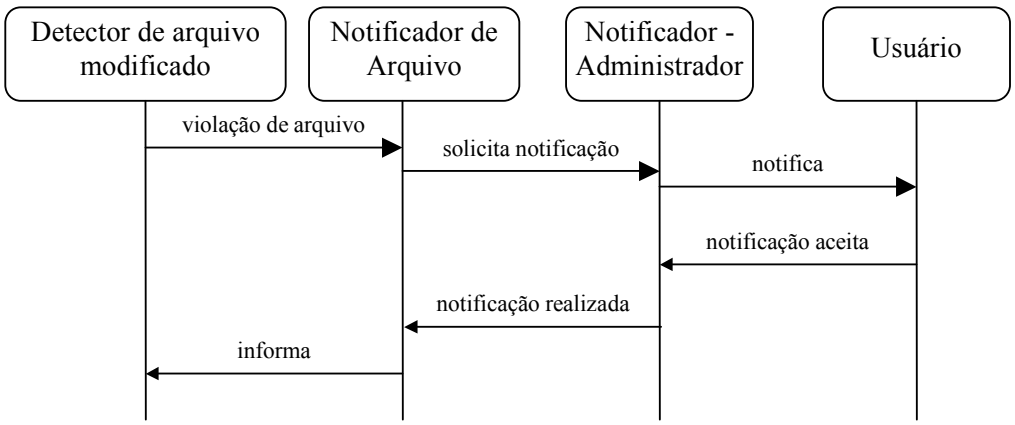


Figura 4.12: Diagrama de Seqüência [DeLoach 2001].

- **Refinamento de Papéis:** neste passo é feito o refinamento do conjunto inicial de papéis definidos no passo anterior. Esse processo de refinamento envolve renomear os papéis, decompô-los em vários papéis ou combinar com outros papéis. A principal atividade do analista neste passo é mapear os objetivos em papéis. Cada papel pode ter vários objetivos e os objetivos do sistema identificados no primeiro passo da metodologia devem ser atribuídos a papéis específicos. Essas informações são representadas no **Modelo de Papéis**, que mostra a associação entre papéis e objetivos. Os papéis são denotados por retângulos e os objetivos são representados por números que correspondem à mesma numeração do Diagrama Hierárquico de Objetivos (Figura 4.10). Por exemplo, na Figura 4.13, o papel "Detector de Arquivo Modificado" está associado ao objetivo 1.1.1 (Determinar se os arquivos foram apagados ou modificados).

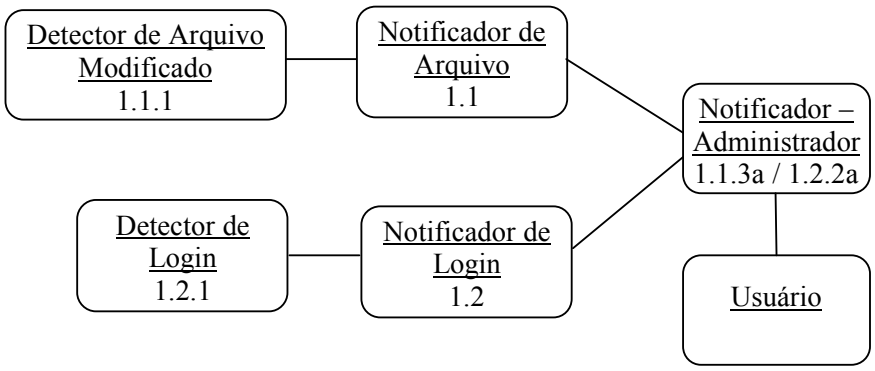


Figura 4.13: Modelo de Papéis – adaptado de DeLoach [DeLoach 2001]

Uma vez definidos os papéis, devem ser criadas as tarefas necessárias para que esses papéis possam cumprir seus objetivos. Segundo os autores, a parte mais interessante e mais difícil da metodologia é transformar os papéis em classes de agente, definindo a comunicação entre os agentes e seus comportamentos internos. Para auxiliar o analista nesse processo, é necessário primeiramente definir as tarefas de alto-nível dos papéis, as quais serão transformadas em funcionalidades específicas dos agentes. As tarefas são representadas através do **Modelo de Papéis** detalhado (Figura 4.14), o qual representa um refinamento do modelo construído no passo anterior. As formas ovais situadas embaixo de cada papel denotam as tarefas a serem executadas por este papel em específico e as linhas entre as tarefas indicam os protocolos (ou as mensagens) entre elas. Por exemplo, na Figura 4.14, o papel "Detector de Arquivo Modificado", que está associado ao objetivo 1.1.1 (Determinar se os arquivos foram apagados ou modificados), contém as tarefas de "detectar a modificação de arquivos" e "determinar a validade do arquivo".

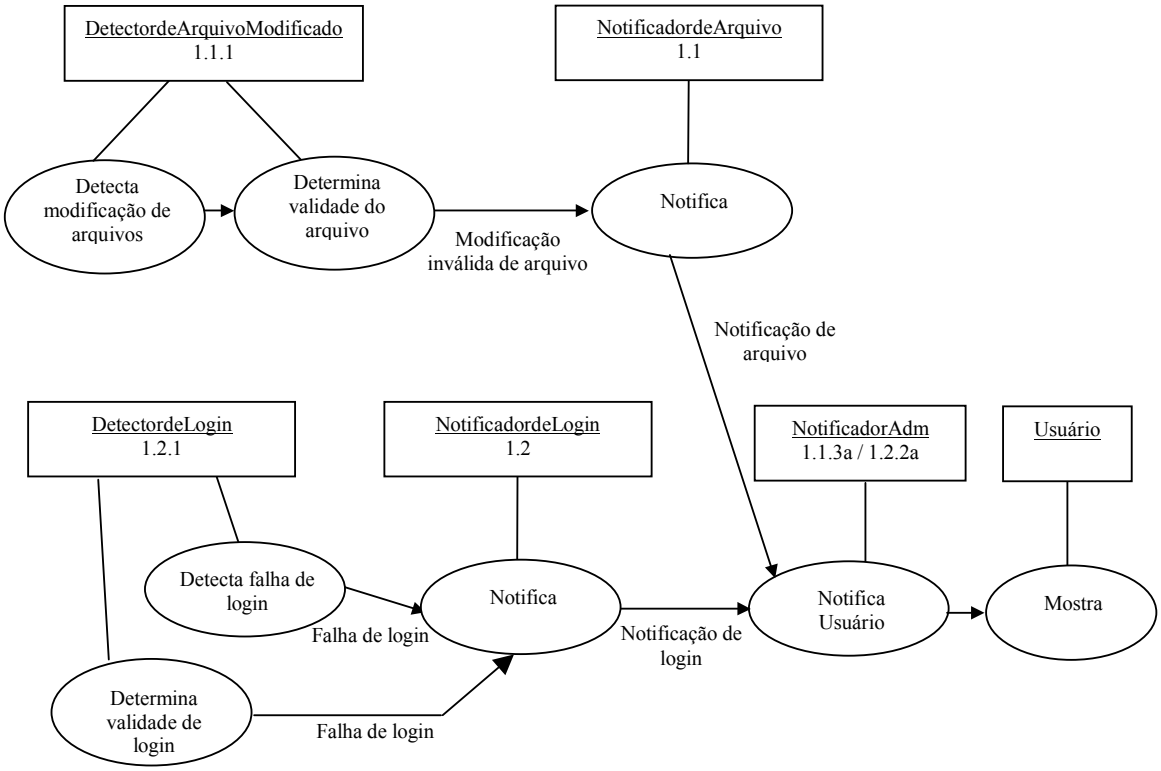


Figura 4.14: Modelo de Papéis detalhado – adaptado de DeLoach [DeLoach 2001]

Durante o passo de refinamento de papéis, pode ser necessário modelar as tarefas concorrentes. MaSE define essas tarefas concorrentes, utilizando-se de um autômato de estados finitos onde os estados descrevem o processamento que acontece no interior do papel e as transições de estado mostram a comunicação entre os papéis ou entre tarefas. O artefato utilizado pela metodologia é o **Diagrama de Tarefas Concorrentes**. A Figura 4.15 detalha a tarefa "Notifica usuário" para o papel "NotificadorAdm". A sintaxe da transição de estado é: *disparador(args1)[guard]^transmissão(args2)*, que significa que se um evento disparador é recebido com um número de argumentos args1 e com condição de guarda, então a transmissão é

enviada com um conjunto de argumentos args2 (cada parte desta sintaxe é opcional). As ações dentro de cada estado são executadas seqüencialmente e são escritas como funções [Lacey 2000].

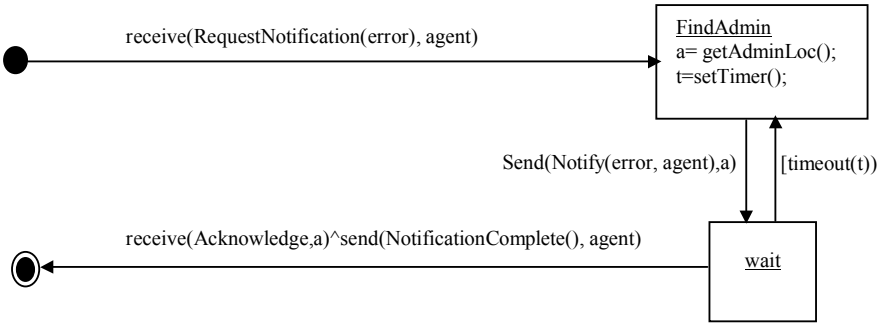


Figura 4.15: Diagrama de Tarefas Concorrentes [DeLoach 2001].

4.3.1.2 Fase de Projeto

A fase de projeto é dividida nos seguintes passos: criação de classes de agente, construção de conversa entre os agentes, montagem de classes de agentes e projeto do sistema. A seguir serão detalhados cada um desses passos:

- **Criação de classes de agentes:** a partir dos papéis gerados na fase anterior, passa-se à identificação das classes dos agentes. Estas classes devem assumir todos os papéis identificados anteriormente. O artefato gerado nesta fase é um **Diagrama de Classes de Agente**. A principal diferença entre o Diagrama de Classes de Agente e o Diagrama de Classes da UML é a semântica dos relacionamentos entre classes de agente. No Diagrama de Classes de Agente, os relacionamentos definem as conversas que são mantidas entre as classes de agentes. Uma classe de agente é representada por retângulos contendo o nome da classe de agentes e seus respectivos papéis. As setas denotam as conversas e apontam do iniciador até o outro participante da conversa, com o nome da conversa escrita próximo à seta. A Figura 4.16 mostra o Diagrama de Classes de Agentes relacionados aos papéis apresentados na Figura 4.14.

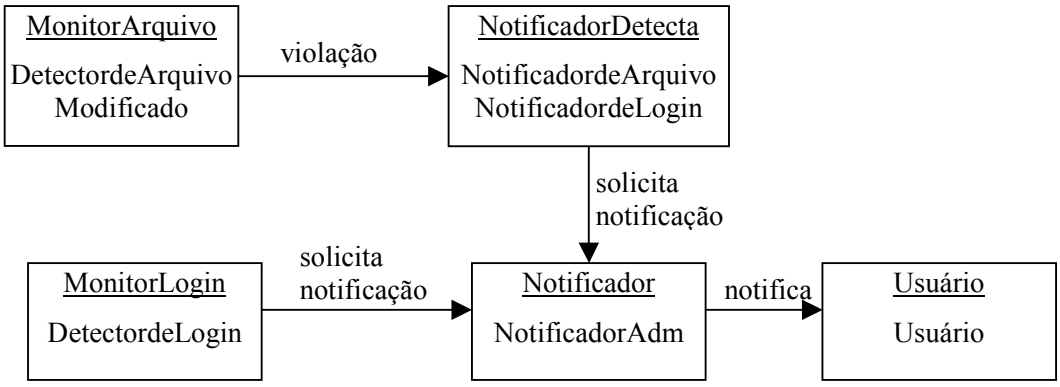


Figura 4.16: Diagrama de Classes de Agentes [DeLoach 2001].

- **Construção de conversas de agentes:** depois de determinar o número e os tipos de classes de agentes no sistema, deve-se definir as conversas entres os agentes referentes a essas classes. Segundo os autores da metodologia, este passo e o próximo devem ser executados paralelamente, pois a arquitetura do agente definida no passo de montagem de classes de agentes deve implementar as conversações e os métodos definidos neste passo. A conversação define um protocolo de coordenação entre as duas classes de agentes que participam da conversa, modelados por dois **Diagramas de Classes de Comunicação** (um para cada agente participante da conversa: iniciador e respondedor). Apesar do nome, este diagrama é uma máquina de estados finitos que define os estados de conversação entre as duas classes participantes. A sintaxe é a mesma utilizada no Diagrama de Tarefas Concorrentes: *disparador(args1)[guard]^transmissão(args2)*. A Figura 4.17 mostra um Diagrama de Classes de Comunicação para a classe do agente iniciador da conversa.

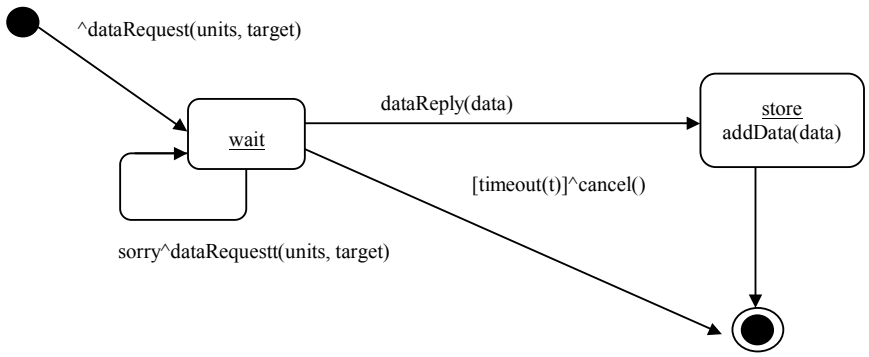


Figura 4.17: Diagrama de Classe de Comunicação [Wood 2001].

- **Montagem de classes de agentes**: neste passo é criado o interior de cada classe de agentes, ou seja, é criada sua arquitetura. Robinson [Robinson 2000] define cinco tipos diferentes de arquiteturas multi-agentes: crença-desejo-intenção (BDI), reativo, planejamento, arquitetura baseada em conhecimento e arquitetura definida pelo usuário. Cada modelo de arquitetura tem um conjunto de componentes específicos. Os autores da MaSE [Wood 2001, DeLoach 2001] apenas sugerem estas arquiteturas, mas não explicam como utilizá-las no processo de modelagem da MaSE.

- **Projeto do Sistema:** é o passo final da metodologia. As classes definidas nas fases anteriores são instanciadas e distribuídas no sistema. O artefato utilizado é o **Diagrama de Distribuição** (Figura 4.18), que define a configuração do sistema a ser implementado, mostrando a quantidade, os tipos e as localizações dos agentes dentro do sistema (por exemplo, os agentes podem ser organizados de acordo com as configurações das máquinas com o objetivo de maximizar o processamento dentro de uma rede). Os agentes são representados por quadrados tridimensionais e suas conversações são representadas por linhas conectadas entre os agentes. Os agentes que utilizam a mesma plataforma física são agrupados em quadrados pontilhados. A idéia de instanciar os agentes das classes de agentes é a mesma que instanciar objetos das classes de objetos na programação orientada a objetos.

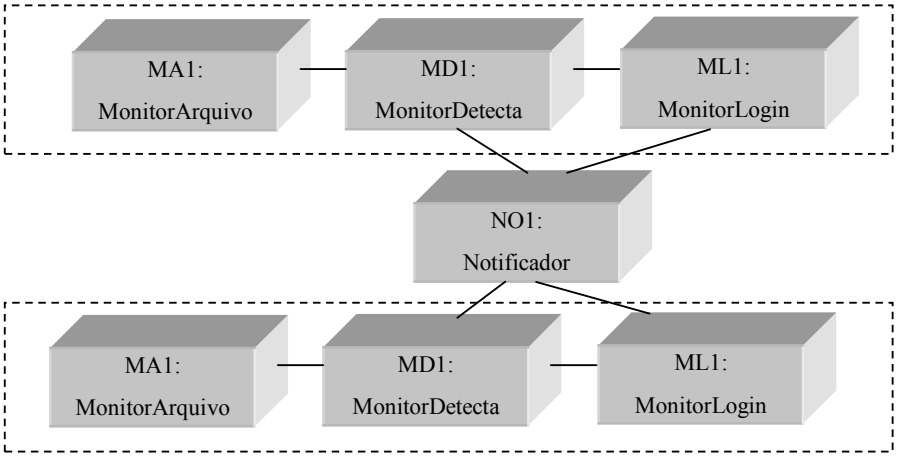


Figura 4.18: Diagrama de Distribuição para as classes de agentes da Figura 4.16.

4.4 Metodologia Prometheus

Prometheus é uma metodologia na Engenharia de Software Orientada a Agentes para especificar, projetar e implementar sistemas de agentes [Padgham 2002a, 2002b, 2002c, 2005]. Está sendo desenvolvida há vários anos em colaboração com a *Agent Oriented Software* (AOS – Estados Unidos, Reino Unido e Austrália). O nome Prometheus é uma referência a um Titã da mitologia grega, conhecido como protetor e benfeitor do homem. O principal objetivo no desenvolvimento desta metodologia é ter um processo definido, que possa ser usado no desenvolvimento de sistemas de agentes e que possa ser ensinado aos profissionais da área e aos estudantes que não possuam conhecimento em agentes. Prometheus suporta o desenvolvimento de agentes inteligentes que possuem objetivos, crenças, planos e eventos. Proporciona mecanismos de estruturação hierárquica que permite ser executado em diferentes níveis de abstração. A metodologia utiliza o conceito de capacidade, a qual pode ser composta por planos, eventos, crenças e outras capacidades que dão habilidades específicas ao agente. O agente é composto por várias capacidades e cada uma delas tem uma função específica.

4.4.1 Fases da Metodologia

A metodologia consiste de três fases: (i) especificação do sistema; (ii) projeto arquitetural; (iii) projeto detalhado. Prometheus segue a abordagem RUP (*Rational Unified Process*) aplicando o processo iterativo em cada fase. Alguns artefatos gerados na metodologia são variações da UML, como por exemplo, os casos de usos definidos na fase de especificação do sistema. A visão geral da metodologia dividida em fases é apresentada na Figura 4.19. Os artefatos produzidos em cada fase são obtidos a partir de artefatos intermediários disponíveis na metodologia. As linhas tracejadas com setas de direcionamento representam a checagem dos artefatos e as linhas contínuas indicam as derivações entre os eles.

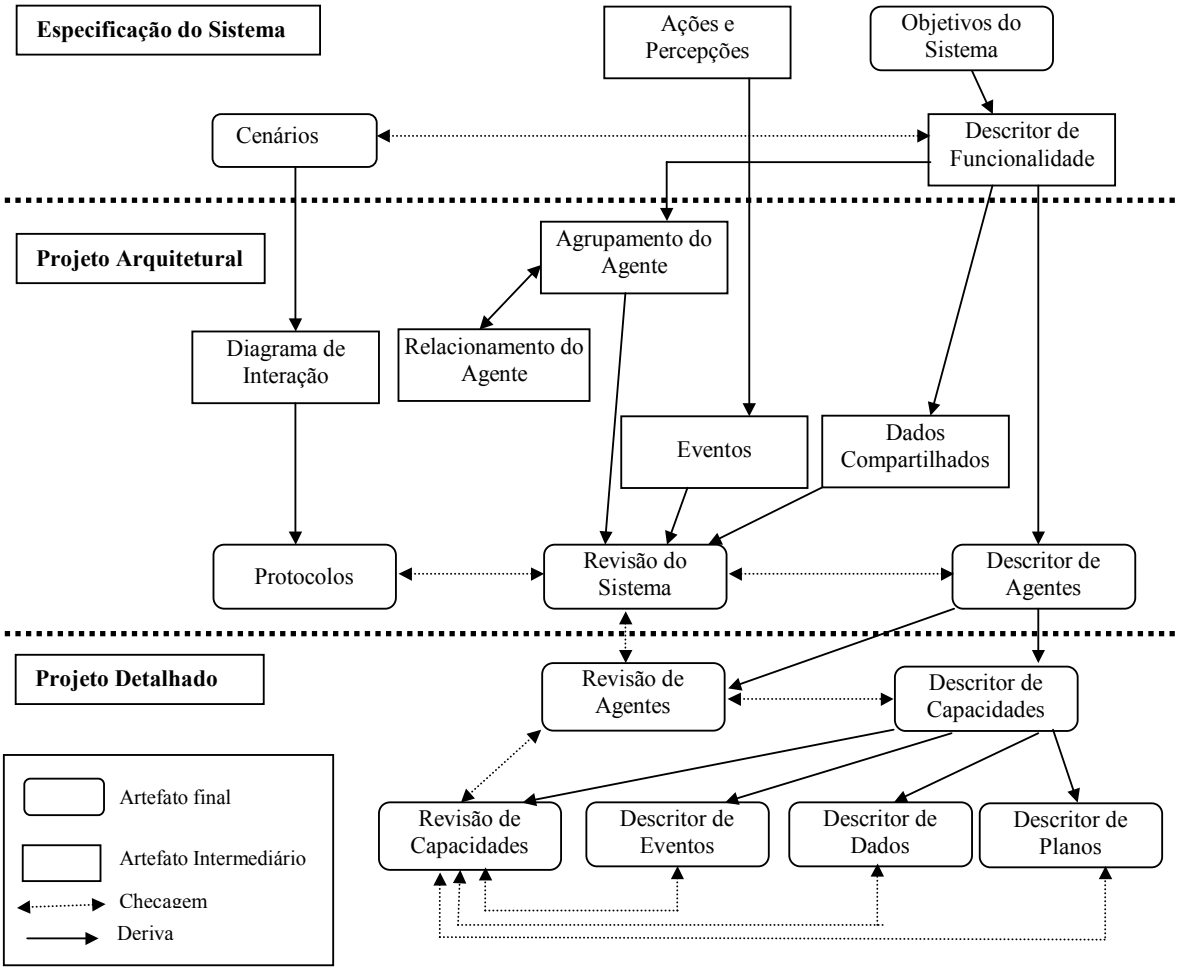


Figura 4.19: Visão geral da metodologia dividida em fases [Padgham 2002c]

4.4.1.1 Fase de Especificação do Sistema

Os objetivos desta fase são construir o modelo do ambiente do sistema, identificar os objetivos e funcionalidades do sistema e descrever os principais cenários dos casos de uso. Esta fase focaliza na identificação das funcionalidades básicas do sistema, suas entradas (percepções), suas saídas (ações) e as principais origens de dados compartilhados. O conceito de funcionalidade utilizado pelo Prometheus equivale ao conceito de papéis em outras metodologias baseadas em agentes. Neste momento, está sendo avaliado como o sistema de agentes irá interagir com o ambiente. Primeiramente, são identificados os principais objetivos do sistema e em seguida, as funcionalidades que irão atingir esses objetivos. Prometheus utiliza um formulário padrão (Anexo A, Item 1), que contém a relação dos objetivos e das funcionalidades do sistema.

Para auxiliar na identificação das funcionalidades Prometheus utiliza o **Descritor de Funcionalidade** (Figura 4.20 e Anexo A, Item 2), onde são verificadas quais informações cada funcionalidade necessita ou produz e quais são os repositórios de dados externos necessários.

Nome: Acolhedor
Descrição: Recebe um novo visitante no site (com informações personalizadas, se possível).
Percepção/eventos/mensagens: CustomerArrived(mensagem),
CustomerInformation(mensagem)
Mensagens enviadas: CustomerInformationRequest(mensagem),
CustomisedWWWPage(mensagem)
Ações: Mostrar a página WWW customizada
Dados usados: CustomerDB, CustomerOrders
Interações: CustomerManager (via CustomerInformationRequest, CustomerInformation)
OnlineInteraction (via CustomisedWWWPage, CustomerArrived)

Figura 4.20: Descritor da funcionalidade "Acolhedor" de uma livraria online adaptado de Padgham [Padgham 2002b]

Como as funcionalidades tratam os aspectos particulares do sistema é necessário definir os **Cenários do Caso de Uso** para obter uma visão holística do sistema. Os casos de uso da metodologia são similares aos da orientação a objetos. Um cenário de caso de uso típico consiste de passos que descrevem as percepções, as mensagens enviadas e as ações dos agentes. Outro artefato produzido nesta fase é o **Descritor de Cenário** (Anexo A, Item 3), constituído basicamente por uma identificação, uma descrição do cenário, o contexto em que o cenário ocorre, os passos do cenário, incluindo suas funcionalidades e possíveis variações do cenário. Todos os descritores utilizados pela metodologia Prometheus seguem o mesmo padrão do descritor de funcionalidade (Figura 4.20). Os artefatos gerados nesta fase devem ser revistos várias vezes com o cliente até atingir o consenso de todos.

4.4.1.2 Fase de Projeto Arquitetural

As três principais atividades desta fase são determinar os tipos de agentes, projetar a estrutura do sistema e definir as interações entre os agentes. Pela análise dos artefatos da fase anterior, as funcionalidades são agrupadas para determinar os agentes necessários, sempre levando em consideração os aspectos de segurança, de privacidade e a capacidade de alteração da funcionalidade durante o agrupamento. As funcionalidades são agrupadas de acordo com dois critérios. No critério de coesão, as funcionalidades que estão relacionadas (por exemplo, usam o mesmo dado) são juntadas no mesmo grupo. No critério de ligação, se existem interações significantes entre duas funcionalidades, elas podem ser agrupadas. O **Diagrama de Ligação de Dados** (Figura 4.21 e Anexo A, Item 4) é utilizado para examinar as propriedades que permitem a união das funcionalidades que utilizam o mesmo banco de dados. Na Figura 4.21, são relacionadas todas as funcionalidades (A a L) e de todos os dados identificados (1 a 6), definindo os agentes X e Y. Para a elaboração deste diagrama, são utilizados os descritores de funcionalidades desenvolvidos na fase anterior.

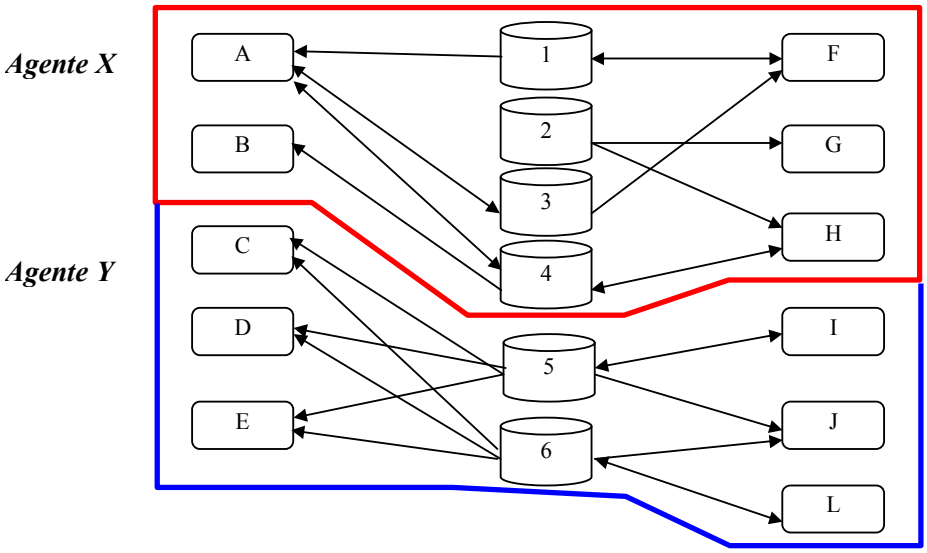


Figura 4.21: Exemplo parcial de um Diagrama de Ligação de Dados, para as funcionalidades (A a L) e os dados (1 a 6).

Para avaliar o agrupamento das funcionalidades Prometheus usa um **Diagrama de Relacionamento do Agente** (Anexo A, Item 5). Esse diagrama liga os agentes com os quais eles interagem, indicando quais funcionalidades serão agrupadas em quais agentes e fornecendo uma visão do nível do agente (tipo de agentes, número esperado de cada tipo de agente e a interação entre agentes do mesmo tipo). Definidos os agentes, o próximo passo é identificar os eventos a que o agente irá responder e as mensagens produzidas pelos agentes, formando a interface entre eles. Os artefatos utilizados para modelar estar informações são (Anexo A, Itens 6 e 7):

- **Descritor de Percepção:** composto pelo nome do identificador, descrição, origem (sensor) da percepção, estrutura, quais os eventos importantes da percepção e dados auxiliares necessários para gerar os eventos.

- **Descritor de Ação:** composto pelo identificador, descrição, parâmetros que influenciam a performance da ação, duração da ação, detecção de falha e mudanças parciais.

- **Descritor de Mensagem:** composto pelo nome, descrição, informação transportada, protocolo de comunicação, qual o agente de origem, qual o agente de destino, finalidade da mensagem.

- **Descritor de Tipo de Agente:** possui informação de alto nível do agente. É composto pelo nome do agente, objetivos, número de instâncias, capacidades necessárias, ciclo de vida do agente, inicialização, término, percepção, ação, mensagens enviadas e recebidas, interações e informações lidas e gravadas. É necessário comparar se os

descritores dos agentes estão consistentes com os descritores de funcionalidades que foram incorporadas em cada agente.

Em seguida, é necessário capturar a estrutura do sistema por meio do **Diagrama de Revisão do Sistema** (Figura 4.22 e Anexo A, Item 8). Este diagrama é considerado o artefato de projeto mais importante, pois apresenta os agentes, as percepções, as mensagens entre agentes, as ações dos agentes e o uso de arquivos de dados, representando um importante artefato de rastreabilidade da metodologia Prometheus [Padgham 2002b]. A Figura 4.22 mostra um Diagrama de Revisão do Sistema parcial de uma livraria on-line.

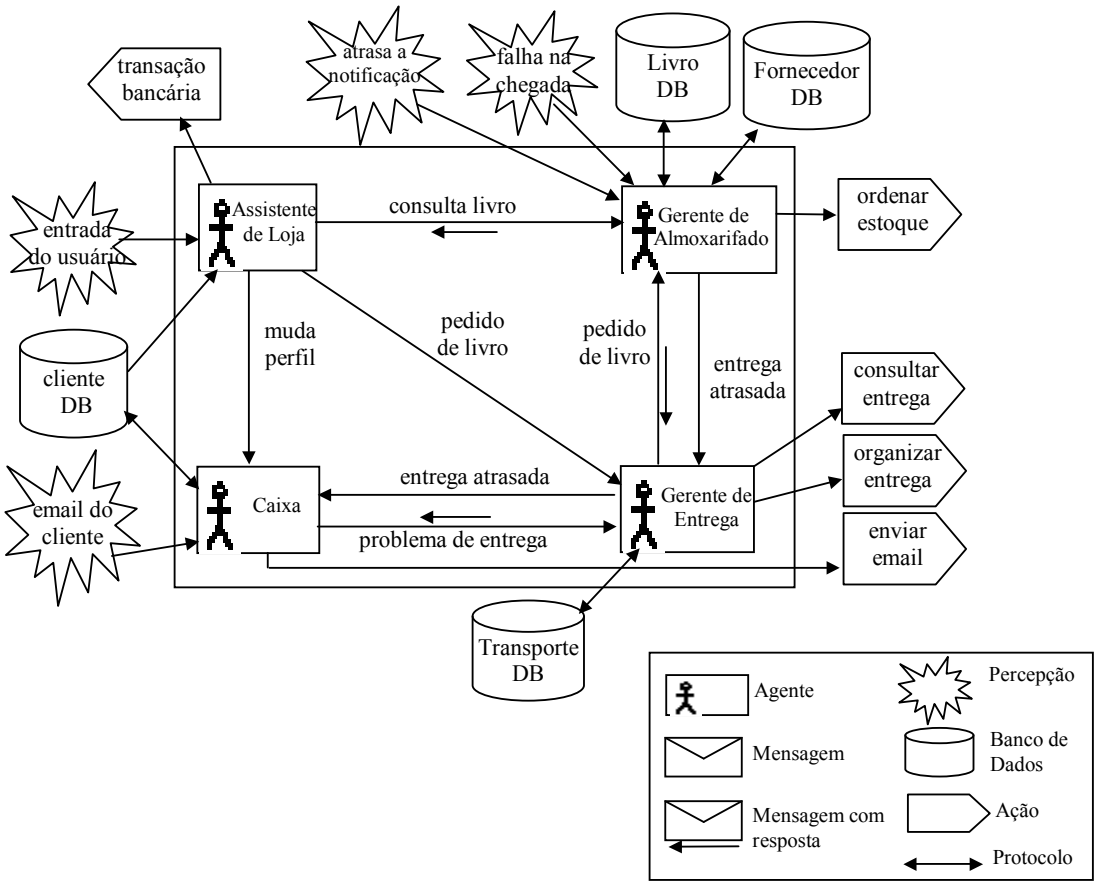


Figura 4.22: Diagrama de Revisão do Sistema parcial da livraria on-line [Padgham 2002c].

O Diagrama de Revisão do Sistema fornece somente a estrutura estática do sistema. Entretanto, nesta fase o projetista já pode capturar o comportamento dinâmico do sistema. Prometheus utiliza o diagrama de interação e os protocolos de interação (Anexo A, Item 9). Os diagramas de interação são semelhantes aos da orientação a objetos e são construídos a partir dos cenários dos casos de uso desenvolvidos na fase de especificação do sistema. A partir dos diagramas de interação são gerados os protocolos de interação que definem mais precisamente quais são as seqüências de interação válidas dentro do sistema. Deve ser feita a consistência entre os diagramas de protocolo e de interação, o diagrama de revisão do sistema e os casos de uso para verificar a integridade entre os modelos.

4.4.1.3 Fase de Projeto Detalhado

A última fase da metodologia concentra-se no desenvolvimento da estrutura interna de cada agente e em como ele realiza suas tarefas no sistema. Ela focaliza na definição das capacidades, em termos de eventos internos, planos e estruturas de dados detalhadas, para cada tipo de agente definido anteriormente. As funcionalidades da fase de especificação formam um conjunto de capacidades. As capacidades são refinadas e no nível de detalhe mais baixo, são definidos os planos, os eventos e os dados. Para modelar essas informações Prometheus utiliza o **Diagrama de Revisão do Agente** (Figura 4.23 e Anexo A, Item 10). Este é um artefato que fornece uma visão de alto nível da estrutura interna do agente. É muito semelhante ao diagrama de revisão do sistema, porém, ao invés de mostrar os agentes no sistema, ele mostra o interior do agente, incluindo o fluxo de eventos, as tarefas e os dados internos do agente. A Figura 4.23 mostra o Diagrama de Revisão do Agente "Gerente de Almoxarifado" para o sistema de livraria on-line.

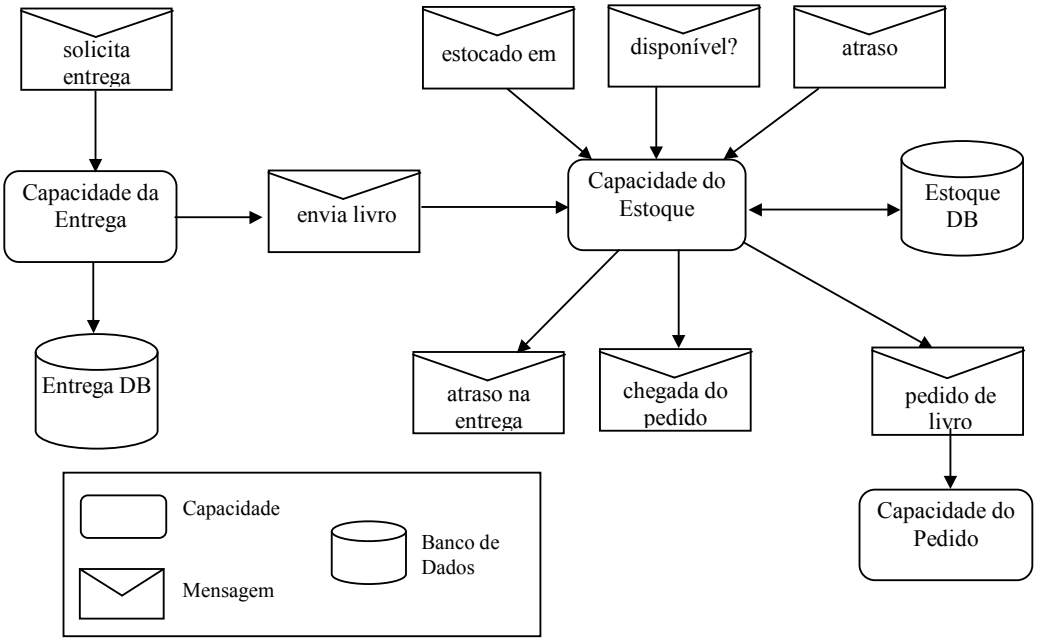


Figura 4.23: Diagrama de Revisão do Agente para o agente "Gerente de Almoxarifado" [Padgham 2002a].

Depois de ter sido elaborado o Diagrama de Revisão do Agente, Prometheus detalha cada capacidade dos agentes. O **Diagrama de Revisão de Capacidade** (Anexo A, Item 11) utiliza uma única capacidade e descreve sua estrutura interna, ilustra os dados, os eventos e os planos existentes dentro de cada capacidade.

Os artefatos finais desta fase são os descritores de plano, de eventos e de estrutura de dados. Basicamente o **Descritor de Plano** (Anexo A, Item 12) contém um identificador, uma documentação, o tipo de evento manipulado, o contexto em que o plano ocorre, informações sobre o fracasso do plano, quais sub-tarefas são geradas pelo plano, entre outros. O **Descritor de Evento** (Anexo A, Item 13) é usado para especificar todos os eventos, deve identificar a finalidade do evento, bem com os dados utilizados pelo evento. As **Estruturas de Dados** (Anexo A, Item 14) são estruturas especializadas, que dependem da plataforma de implementação, ou do banco de dados utilizado. Os dados devem ser especificados utilizando

paradigmas ou diagramas apropriados (ex: diagrama de classe, tabela de dados, etc). O **Dicionário de Dados** (Anexo A, Item 15) é um artefato que deve ser elaborado desde o início do projeto e refinado em todas as fases do sistema.

Prometheus é suportado por duas ferramentas [Padgham 2002b]. *Jack Development Environment* (JDE), desenvolvida por Agent Oriented Software, permite modelar os diagramas de revisão existentes na metodologia. *Prometheus Design Tool* (PDT) permite verificar as consistências entre os diagramas, bem como gerar a documentação do projeto. Entretanto, nenhuma dessas ferramentas suporta a fase de especificação do sistema.

4.5 Metodologia Tropos

Tropos é uma metodologia de desenvolvimento de software orientado a agentes, criada por um grupo de pesquisadores de várias universidades do Brasil, do Canadá e da Itália [Bresciani 2002, Bresciani 2004, Castro 2002, Giunchiglia 2002, Mylopoulos 2002, Perini 2001, Sannicolo 2002]. A metodologia baseia-se em duas idéias principais: (i) usa a noção de agente e seus conceitos, tais como objetivos, planos, tarefas, etc., durante as fases do desenvolvimento do software; (ii) adota uma abordagem de refinamento de passos, utilizando um conjunto específico de operadores de transformação. Diferentemente das demais metodologias apresentadas, Tropos tem um foco maior na fase de requisitos iniciais, onde os conhecimentos e as necessidades dos clientes são identificados e analisados. A metodologia provê ferramentas e técnicas para construir modelos derivados do paradigma i* [Yu 1997] (que significa "intencionalmente distribuído"), um *framework* de modelagem que inclui os conceitos de atores (que podem ser agentes, posições ou papéis) e suas interdependências intencionais (incluindo dependência de objetivos, tarefas e recursos).

4.5.1 Fases da Metodologia

A metodologia consiste de cinco fases: (i) requisitos iniciais; (ii) requisitos finais; (iii) projeto arquitetural; (iv) projeto detalhado e (v) implementação.

4.5.1.1 Fase de Requisitos Iniciais

O principal objetivo desta fase é o entendimento do problema, estudando a configuração organizacional existente e identificando o domínio e as necessidades do cliente. Primeiramente, o ambiente é analisado e modelado, incluindo seus atores relevantes e suas dependências. Os clientes são modelados como atores e suas intenções, como objetivos. Em seguida, cada objetivo é analisado sob o ponto de vista de um ator específico. Tropos adota dois tipos de objetivos: (i) objetivos propriamente ditos estão relacionados aos requisitos funcionais do sistema e (ii) objetivos leves ou secundários (*softgoals*) são os objetivos relacionados aos requisitos não funcionais do sistema, cujas condições não estão claramente definidas e que estão relacionadas aos aspectos qualitativos do sistema. A atividade é descrita como um processo iterativo baseado em transformações (*top-down* ou *bottom-up*) sucessivas, onde o modelo é progressivamente

definido e refinado. Os artefatos produzidos nesta fase são: **Diagrama de Ator** (modelo de dependência estratégica no paradigma i*) e **Diagrama de Raciocínio** (modelo de raciocínio estratégico no paradigma i*).

- **Diagrama de Ator:** modela graficamente os atores e suas dependências. Os atores são representados por círculos, seus objetivos por formas ovais e seus objetivos secundários têm forma de nuvem. As ligações de relacionamentos de dependências entre os atores são representadas por linhas com setas conectadas ao objeto de dependência, o qual pode ser um objetivo, um tarefa ou um recurso. A Figura 4.24 mostra o Diagrama de Ator da fase inicial (análise de requisitos) de um sistema de busca via *web* de informação cultural (eCulture System). Os atores iniciais do sistema são: Cidadão, PAT (*Provincia Autonoma di Trento*) e Museu. O ator Cidadão está associado com um único objetivo relevante: "obter informação cultural", enquanto que o ator Visitante possui um objetivo-leve de "apreciar uma visita". PAT possui o objetivo de "aumentar o uso da internet", enquanto que Museu possui o objetivo de "fornecer serviço cultural". Finalmente, o diagrama inclui um objetivo-leve de dependência onde Cidadão depende do PAT para satisfazer o objetivo-leve de "taxas bem pagas".

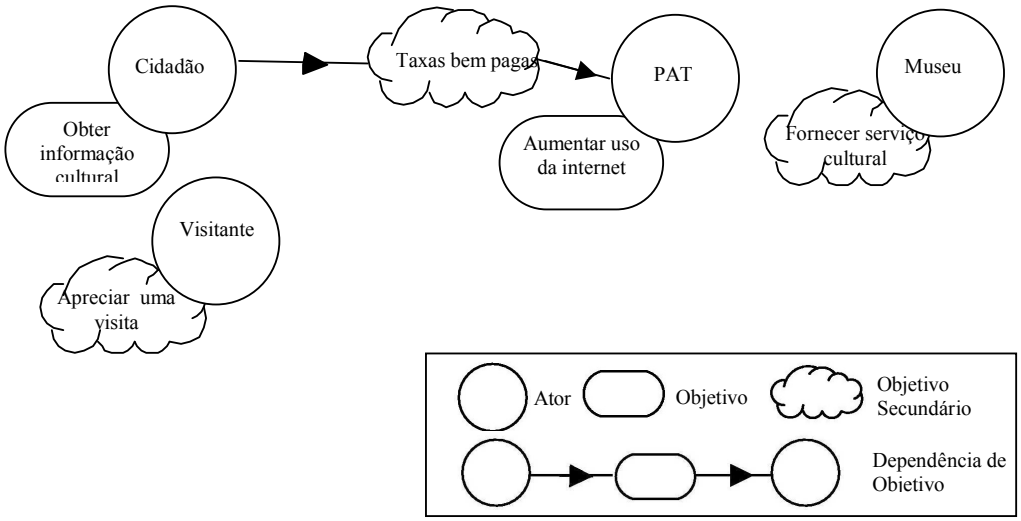


Figura 4.24: Diagrama de Ator: clientes do projeto eCultural [Bresciani 2004].

- **Diagrama de Raciocínio:** este diagrama modela os objetivos sob o ponto de vista de cada ator. O Diagrama de Raciocínio representa a visão de um ator como um grande círculo que contém grafos cujos nós são os objetivos (forma oval) e os objetivos-leves (forma de nuvem) e os arcos representam diferentes tipos de relacionamentos que podem ser identificados entre seus nós. Este diagrama determina por meio da análise "meios-fim" como os objetivos podem ser cumpridos. O sinal de "+" representa uma contribuição positiva entre objetivos. A Figura 4.25 mostra o Diagrama de Raciocínio do ator PAT. Por exemplo, o objetivo-leve "taxas bem pagas" obtém contribuições positivas do objetivo-leve "bons serviços" e "custo razoável".

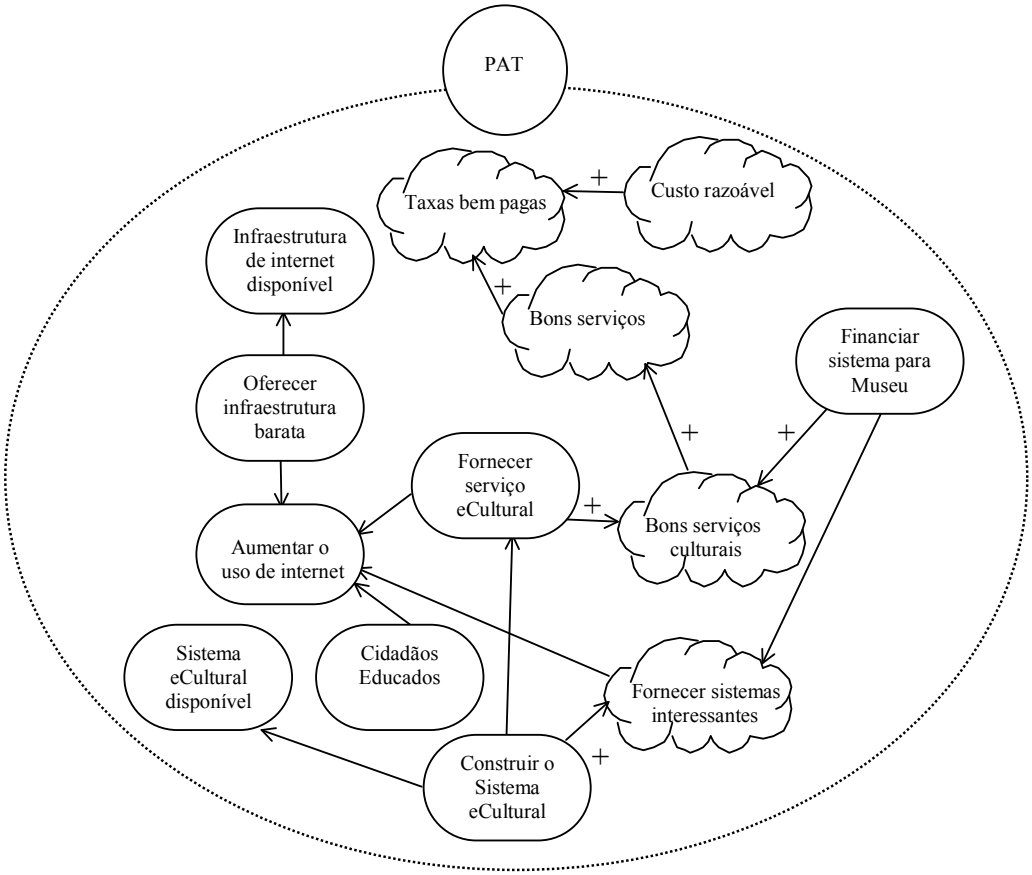


Figura 4.25: Diagrama de Raciocínio para o ator PAT [Bresciani 2004].

4.5.1.2 Fase de Requisitos Finais

Esta fase tem com objetivo especificar o sistema dentro do seu ambiente, incluindo suas funções e suas qualidades relevantes. O sistema é representado como um ou mais atores e suas dependências com outros atores no ambiente. Essas dependências definem os requisitos funcionais e não funcionais do sistema. Em outras palavras, o sistema começa a ter a forma de um ou mais atores que contribuem para a realização dos objetivos do sistema. A Figura 4.26 mostra a relação de dependência entre os atores Cidadão e Sistema eCultural. No exemplo, o objetivo "buscar informação" (um sub-objetivo do objetivo obter informação cultural) pode ser atingido em quatro planos diferentes: buscar por área (área temática), buscar por área geográfica, buscar por palavras chave e buscar por período de tempo. O sub-plano "encontrar origem da informação" depende do ator Museu para a descrição das informações fornecidas.

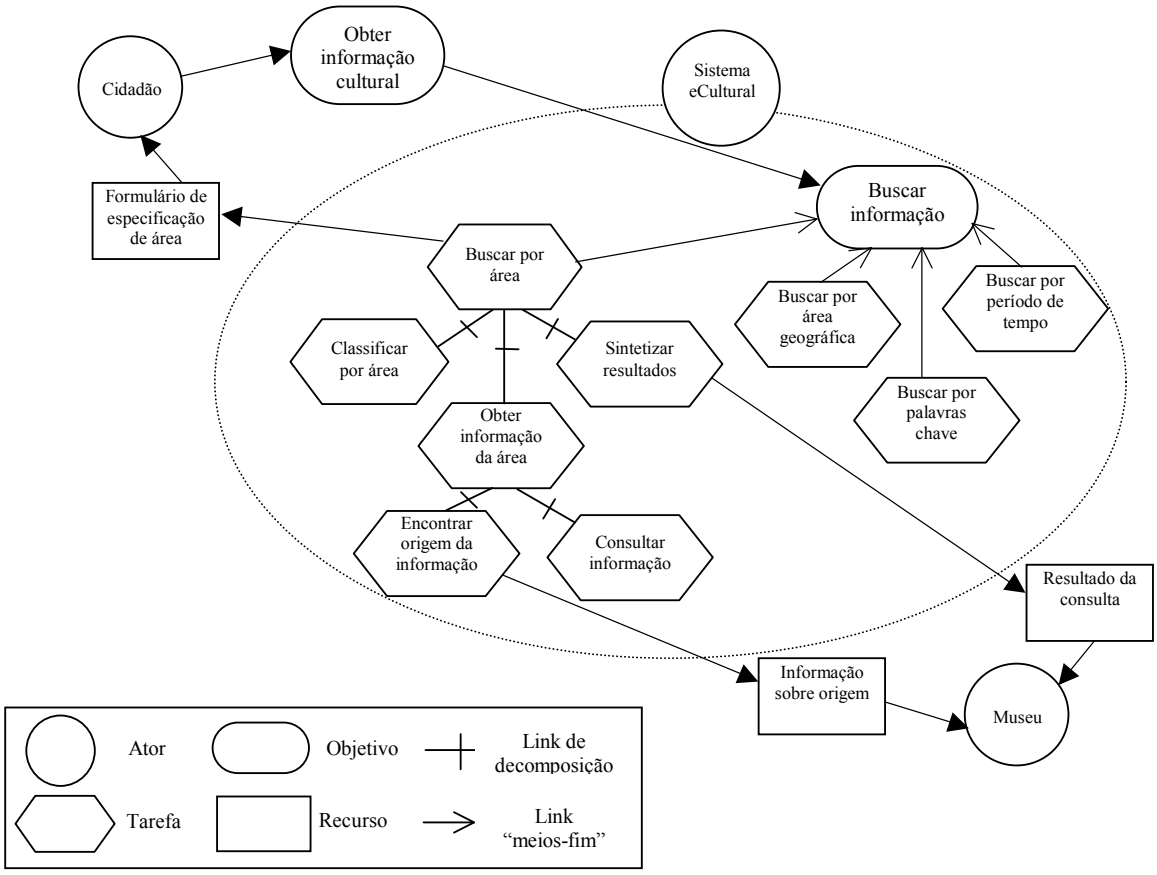


Figura 4.26: Diagrama de Raciocínio parcial para o objetivo "obter informação cultural" que o ator "Cidadãos" depende do ator "Sistema eCultural"[Bresciani 2004].

4.5.1.3 Fase de Projeto Arquitetural

O principal objetivo desta fase é definir a arquitetura global do sistema em termos de subsistemas interconectados através de dados e fluxos de controle. Os subsistemas são representados como atores e as interconexões de controle / dado são representadas como dependências de atores. Esta fase consiste basicamente de 3 passos:

1. O primeiro passo define a estrutura organizacional do sistema que é representada em termos de atores e suas dependências (similar aos modelos construídos nos passos anteriores, porém com mais detalhes). É necessário refinar o Diagrama de Ator introduzindo novos atores, devido ao surgimento de novos sub-objetivos do sistema ou para a realização de algum requisito funcional. Também é necessário decompor os atores e as dependências existentes em sub-atores e sub-dependências. A Figura 4.27 mostra a decomposição em sub-atores do Sistema eCultural e a delegação de alguns objetivos. No exemplo, o Sistema eCultural depende do sub-ator Responsável pela informação para fornecer a informação, do Responsável Educacional para fornecer serviços educacional. Adicionalmente, cada sub-ator pode ser decomposto em sub-atores responsáveis por atender um ou mais sub-objetivos.

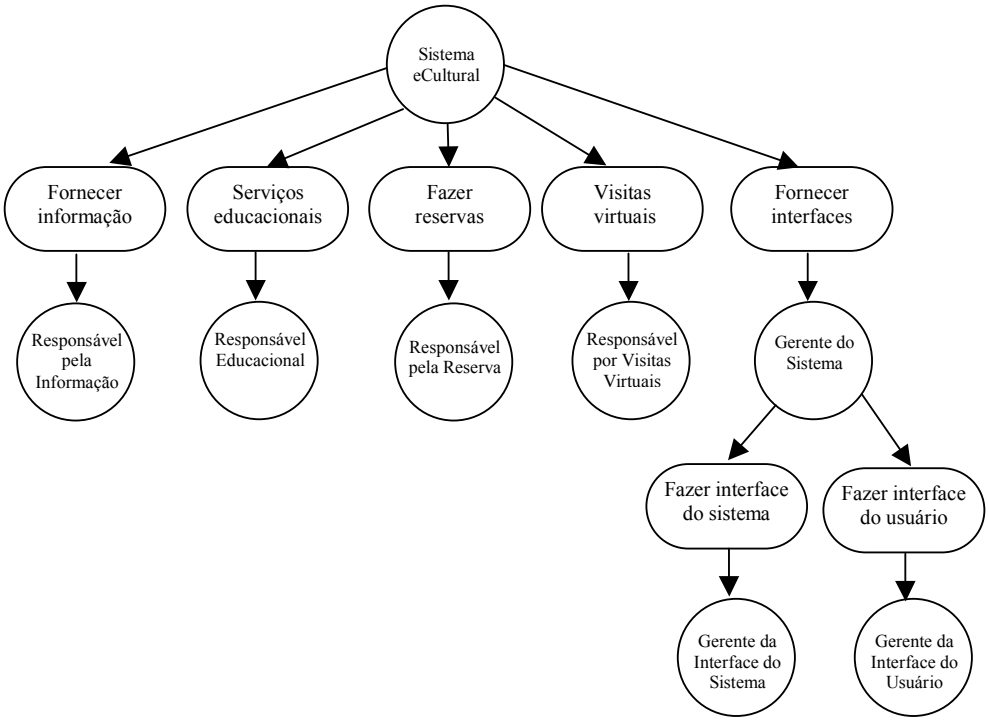


Figura 4.27: Decomposição de sub-atores para o Sistema eCultural [Bresciani 2004].

O resultado final desse passo é **Diagrama de Ator Estendido**, no qual são representados novos atores e suas dependências com outros atores. A Figura 4.28 mostra o Diagrama de Ator Estendido referente ao ator "Responsável pela Informação" e à tarefa "buscar por área". Foram incluídos novos atores, como por exemplo, "Gerente de Interface de Usuário" e "Gerente de Interface de Origem", que são responsáveis por relacionar o sistema aos atores externos "Cidadão" e "Museu".

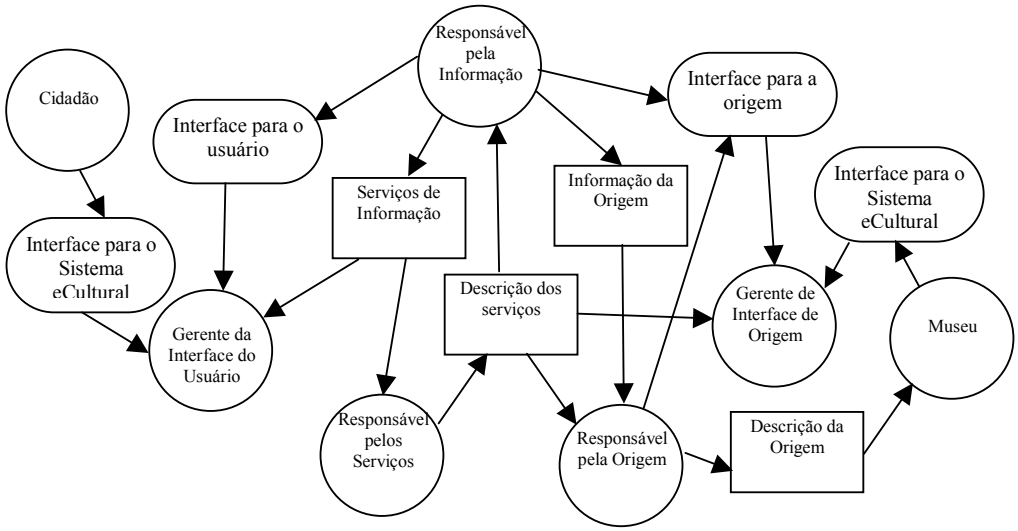


Figura 4.28: Diagrama de Ator estendido para a decomposição do ator "Responsável pela informação", mais especificamente para a tarefa "buscar por área" [Giorgini 2001].

2. O próximo passo consiste em identificar as capacidades necessárias para os atores realizarem seus objetivos e tarefas. As capacidades dos atores são definidas baseadas no relacionamento de dependência entre um ator e os demais atores do sistema. Cada dependência resulta em uma ou mais capacidades e para cada capacidade o ator tem um conjunto de tarefas que podem ser aplicado em diferentes situações. Uma tarefa descreve a seqüência de ações a executar e as condições sob o qual a tarefa pode ser aplicada. O artefato produzido é a **Tabela de Capacidades**, conforme modelo a seguir.

Tabela 4.1: Tabela de Capacidades dos Atores - adaptado [Bresciani 2004].

Nome do Ator	**Número**	**Capacidades**
Classificador da Área	1 2 3 4	Obter especificação da área Classificar área Fornecer informação da área Fornecer descrição do serviço
Pesquisador de Informação	5 6 7 8	Obter informação da área Encontrar origem da informação Fornecer informação da consulta Fornecer descrição de serviço
Gerenciar a Interface do Usuário	9 10 11 12	Obter especificação do usuário Fornecer especificação do usuário Obter resultados da consulta Apresentar os resultados da consulta ao usuário

3. E finalmente, é preciso definir um conjunto de agentes e atribuir para cada agente uma ou mais capacidades, ou seja, dadas as capacidades identificadas no passo anterior, elas serão atribuídas a agentes específicos. A análise do Diagrama do Ator refinado auxilia o projetista a decidir qual tipo de agente está associado a quais capacidades. Na Tabela 4.2 foram definidos os agentes que se relacionam com uma ou mais capacidades identificadas na Tabela 4.1.

Tabela 4.2: Tabela de Tipos de Agentes - adaptado [Bresciani 2004].

Agente	**Capacidades**
Manipulador Consulta	1, 3, 4, 5, 6, 7, 8
Classificador	2, 4
Buscador	6, 4
Interface de Usuário	9, 10, 11, 12, 4
Facilitador de Diretório	4

4.5.1.4 Fase de Projeto Detalhado

A fase de projeto detalhado tem como objetivo especificar o nível micro do agente, detalhando as capacidades e os planos, bem como os protocolos de comunicação e coordenação do agente. Tropos utiliza os seguintes diagramas:

- **Diagrama de Capacidades:** Utilizado para modelar a capacidade (ou o conjunto de capacidades equivalentes) do ponto de vista de um agente específico. É útil quando se deseja avaliar os aspectos dinâmicos de uma capacidade. Os Diagramas de Capacidades são iniciados por eventos externos e os estados de ação modelam as tarefas. A Figura 4.29 mostra um Diagrama de Capacidade referente à capacidade Apresentar os Resultados da Consulta relacionado ao agente Interface de Usuário.

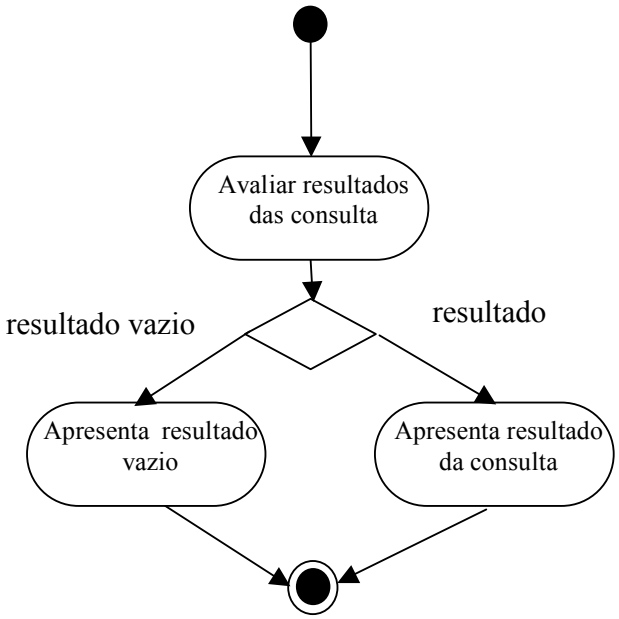


Figura 4.29: Diagrama da Capacidade "Resultado da Consulta" [Bresciani 2004].

- **Diagrama de Planos:** equivale ao diagrama de atividades da AUML. Utilizado para especificar o estado interno de um plano contido em uma capacidade.

- **Diagrama de Interação do Agente:** equivale ao diagrama de seqüência da AUML. Utilizado para modelar as interações dos agentes em termos de ato de comunicação, pois determina o momento da interação e a ordem do envio da mensagem, representando a seqüência cronológica da comunicação. Os agentes são colocados na linha horizontal (*top/down*) e os arcos de comunicação entre os agentes correspondem aos arcos de mensagens assíncronas. A Figura 4.30 mostra um Diagrama de Interação, utilizado na metodologia Tropos.

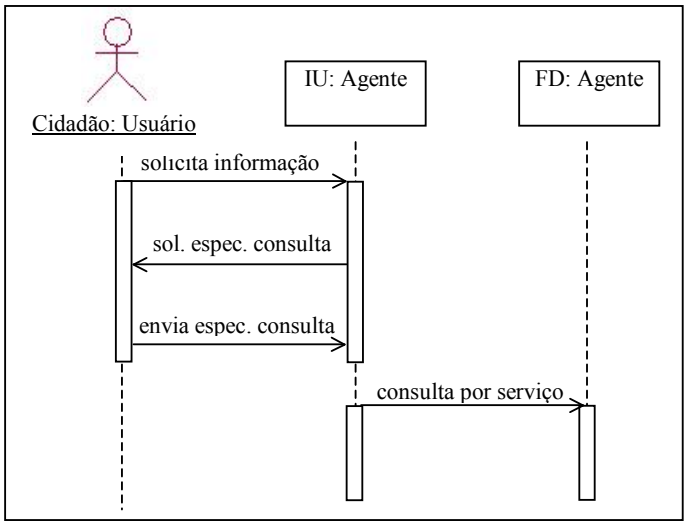


Figura 4.30: Diagrama de Interação. As caixas representam agentes (IU: Interface de Usuário; FD: Facilitador de Diretório) e as linhas modelam os atos comunicativos - adaptado [Bresciani 2004].

4.5.1.5 Fase de Implementação

As plataformas de agentes para suportar implementações de um Sistema Multi-Agentes especificados no contexto da metodologia Tropos incluem principalmente *Jack Intelligent Agents* [Busetta 1999] e JADE [Silva 2005]. Jack é um ambiente de desenvolvimento de agente construído em Java, onde os agentes são componentes de software autônomos que possuem objetivos (desejos) explícitos a atingir ou eventos para manipular. Na plataforma JADE, o comportamento representa uma tarefa que um agente pode realizar. Uma das mais importantes características da JADE é a habilidade de comunicação. As mensagens trocadas pelos agentes da JADE possuem formatos especificados pela linguagem ACL definidos pela FIPA.

4.6 Critérios para Avaliação das Metodologias Orientadas a Agentes

Diferentemente das metodologias orientadas a objetos, não existem muitos trabalhos que comparam e avaliam as metodologias orientadas a agentes, principalmente por se tratar de uma tecnologia recente; porém, alguns critérios estão sendo elaborados com o objetivo de contribuir com esse processo de avaliação.

O'Malley e DeLoach [O'Malley 2002] propõem um conjunto de critérios para auxiliar na decisão entre adotar metodologias orientadas a agentes ou metodologias orientadas a objetos, fazendo uma comparação entre a metodologia MaSE e a metodologia orientada a objeto desenvolvida por Booch. Esses critérios são divididos em dois grandes grupos: questões de gerenciamento e requisitos de projeto.

- **Gerenciamento:** engloba o custo para a aquisição da metodologia e ferramentas (inclui também o custo de treinamento de pessoal), práticas de negócio organizacional (qual o impacto que a adoção da metodologia terá nas práticas de negócio da organização? a metodologia permite seguir os padrões adotados pela organização?), disponibilidade de componentes reusáveis (a metodologia permite incorporar componentes pré-definidos ao sistema?) e maturidade da metodologia.

- **Requisitos de Projeto:** integração com outros sistemas (a metodologia permite integração com os sistemas desenvolvidos anteriormente?), distribuição (a metodologia suporta a modelagem de aspectos distribuídos do problema?), ambiente (a metodologia suporta o desenvolvimento de sistemas de software para ambientes com hardware e software diferentes?), estrutura dinâmica e escalabilidade (a metodologia é capaz de lidar com a adição ou exclusão de componentes de sistema sem prejuízo ao usuário?), agilidade e robustez (a metodologia é capaz de criar sistemas de software flexíveis que atendam as mudanças dinâmicas do ambiente?) e interação (a metodologia é capaz de lidar com a interação entre componentes do sistema, bem como com as entidades de fora do sistema como usuários e outros sistemas?).

Dam [Dam 2003] propõe um *framework* para avaliação de metodologias orientadas a agentes, composto por quatro critérios: conceitos, linguagem de modelagem, processo e critérios pragmáticos.

- **Conceitos:** os conceitos relacionados a agentes são divididos em características internas e características de cooperação. A metodologia deve suportar aspectos internos do agente, como autonomia (a metodologia modela mecanismos de tomada de decisão do agente?), atitudes mentais (a metodologia fornece técnicas de modelagem de objetivo, capturando os objetivos do sistema e do agente?), pró-atividade (a metodologia fornece modelos de planos e tarefas que mostram como os objetivos são realizados pelos agentes?), reatividade (a metodologia oferece mecanismos que representam as mudanças do ambiente?) e concorrência (a metodologia possui técnicas e modelos para capturar características concorrentes da conversação entre agentes?). As características de cooperação também devem ser abordadas pelas metodologias, como métodos de cooperação, modo de comunicação (qual o modo de comunicação suportado pela metodologia?), protocolos (a metodologia possui mecanismos para representar os protocolos de comunicação?) e linguagem de comunicação (que tipo de linguagem de comunicação é suportado pela metodologia?).

- **Linguagem de Modelagem:** os aspectos da linguagem de modelagem são relacionados aos critérios de usabilidade e critérios técnicos. Basicamente, os critérios de usabilidade avaliam se a metodologia é fácil de ser utilizada, se é fácil de ser compreendida (os símbolos e as sintaxes são bem definidos?). Por outro lado, os critérios técnicos consideram os aspectos de não-ambigüidade, consistência, rastreabilidade, refinamento e reusabilidade da metodologia (a metodologia fornece técnicas para verificar os modelos e garantir que todas as ambigüidades foram eliminadas? a metodologia possui roteiros e técnicas para checar as consistências

dentro e entre os modelos? a metodologia oferece mecanismos para o reuso ou derivação de componentes?).

- **Processo:** os aspectos relacionados ao processo de desenvolvimento de software devem ser levados em consideração para a avaliação da metodologia. Por exemplo, a metodologia cobre todas as fases de desenvolvimento do software? qual a abordagem da Engenharia de Software que a metodologia suporta (seqüencial, cascata, iterativo)?

- **Critérios pragmáticos:** esses critérios são baseados em questões de gerenciamento e questões técnicas (qual o custo para se adotar a metodologia? quais são os recursos disponíveis suportados pela metodologia? a metodologia suporta projeto de sistemas abertos? e distribuídos?).

Durante o processo de avaliação das metodologias MaSE, Prometheus e Tropos, realizado neste trabalho, utilizamos alguns desses critérios propostos por Dam, os quais serviram de instrumento para a seleção dos artefatos a serem utilizados na metodologia proposta. O resultado dessa avaliação é apresentado na Seção 6.2.

4.7 Resumo

Atualmente estão sendo propostas várias metodologias orientadas a agentes, sendo que cada uma delas possui vantagens, desvantagens e características específicas para suportar diferentes aspectos dos domínios das aplicações.

Neste capítulo foi feita uma revisão das metodologias orientadas a agentes mais discutidas na literatura, que são MaSE, Prometheus e Tropos, além da linguagem de modelagem AUML. O objetivo foi analisar a abordagem de cada uma dessas metodologias, seus principais pontos no processo de desenvolvimento de software e as ferramentas disponíveis.

Finalmente, foram apresentados alguns critérios utilizados para contribuir no processo de avaliação das metodologias orientadas a agentes. Percebe-se que é necessária uma maior colaboração entre os grupos de desenvolvimento de abordagens orientada a agentes para suprir as necessidades acadêmicas e industriais, fornecendo um processo robusto para o desenvolvimento de sistemas orientado a agentes, que possa ser utilizado em todo o seu ciclo de vida.

Capítulo 5

Proposta de um Novo Modelo de Representação de Interação

5.1 Introdução

Os Sistemas Multi-Agentes são inerentemente complexos devido ao comportamento não determinístico que esses sistemas podem exibir e ao fato de que os agentes podem interagir de maneira sofisticada e às vezes imprevisível. Uma interação define uma seqüência possível de mensagens que os agentes devem trocar para atingir seus objetivos. A posição que a interação ocupa nos Sistemas Multi-Agentes faz com que ela se torne sujeita às novas demandas, uma vez que o protocolo de interação deve ser bem implementado em cada agente e bem compreendido por todos os agentes pertencentes ao sistema. Assim, são necessários modelos que representem corretamente essas interações, as quais possam ser entendidas pelos engenheiros de software e projetistas, a fim de se evitar erros durante a implementação do sistema.

Atualmente, as interações entre os agentes são modeladas basicamente por meio de artefatos como os diagramas da AUML e suas adaptações. Entretanto esses diagramas são muitas vezes limitados para descrever toda a riqueza de comportamento que uma interação entre agentes pode desenvolver, uma vez que os agentes são autônomos, pró-ativos, reativos e buscam por mensagens no ambiente, características não totalmente contempladas nos diagramas da AUML. Dessa maneira, existe uma demanda por novos modelos de representação de interação que melhor representem essa complexidade de comportamentos.

Dentro dessa perspectiva, propomos a utilização da Rede de Processamento de Recursos (*Resource Processing Network* – ou RP-Net) como um novo modelo de representação de interação entre agentes, seguindo o Modelo de Interação do tipo Busca por Mensagens, introduzido na Seção 2.4. Esse novo modelo de representação de interação será utilizado como um artefato da Metodologia Unificada para o desenvolvimento de sistemas orientados a agentes.

Neste capítulo serão apresentados os conceitos básicos da RP-Net, uma ferramenta formal-computacional derivada das Redes de Petri. A Seção 5.2 aborda os tipos de representação dos modelos de interação. Na Seção 5.3 é detalhada a RP-Net e na Seção 5.4 é feito um resumo do capítulo.

5.2. Representação dos Modelos de Interação

Neste trabalho, estamos propondo que um agente deve seguir um modelo de interação do tipo "Busca por Mensagens" para que possa ser adequadamente chamado de um agente, conforme apresentado na Seção 2.4. Lembramos que, segundo este modelo, um agente deve buscar por mensagens que são depositadas em *buffers*. Dessa maneira, o agente pode coletar as informações

no ambiente por meio de sensores e atuar no ambiente através de atuadores, sem se preocupar com o processo que produz a informação desejada ou que irá consumi-la. Uma vez que adotamos essa restrição, verificamos que os artefatos para representar a interação dos agentes utilizados pelas metodologias estudadas não contemplam adequadamente essa restrição.

A proposta inicial de AUML é representar protocolos de interação de agentes que descrevam um padrão de comunicação análogo ao de objetos - ou seja, uma seqüência de mensagens (ou atos de comunicação) diretamente entre agentes, desconsiderando-se a existência de *buffers* intermediários desacoplando essas mensagens. Algumas limitações identificadas nos diagramas AUML são:

- **Mensagens síncronas e concorrentes:** embora seja possível representar mensagens assíncronas, a representação dos diagramas AUML é mais apropriada para mensagens síncronas, ideal para os sistemas orientados a objetos, não contemplando de maneira apropriada o conceito de mensagens assíncronas utilizadas pelos agentes. Da mesma forma, a utilização de *buffers* não é modelada adequadamente.
- **Dificuldade para modelar várias interações:** quando um agente envia mensagens diferentes para vários agentes, várias linhas de tempo são necessárias para representar essa interação. Dado um conjunto de *n* agentes, se existem *m* mensagens diferentes enviada por cada agente, o diagrama deverá mostrar *n* linhas do tempo com *m* mensagens em cada linha de tempo, isso tudo sem levar em consideração as condições de rejeição da mensagem ou *timeout*, tornando o diagrama de difícil leitura e compreensão. Dessa forma, a AUML não captura a natureza multilateral das interações entre agentes.
- **Diagramas estáticos para um comportamento dinâmico:** embora os diagramas da AUML tenham o objetivo de representar o comportamento dinâmico dos sistemas orientados a agentes, essa dinamicidade fica prejudicada pelo caráter estático dos diagramas. Dinâmicas mais complexas acabam gerando diagramas muito sobrecarregados, tornando mais difícil sua interpretação e portanto sua utilidade. Uma alternativa seria utilizar diagramas dinâmicos[4].

Para que um artefato modele adequadamente as interações entre os agentes é necessário possuir uma representação gráfica que elimine as dificuldades encontradas nos diagramas AUML. Com o objetivo de suprir essa necessidade, este trabalho propõe a utilização da RP-Net como uma melhoria na representação das interações entre os agentes.

5.3. RP-Net

A Rede de Processamento de Recursos (RP-Net), desenvolvida por Tatai [Tatai 2003] representa uma evolução da Rede de Objetos [Gudwin 1996], da Rede de Agentes [Guerreiro

[4] Um diagrama dinâmico é basicamente um diagrama que evolui no tempo. Possui uma parte estática, que permanece sempre a mesma, durante todo o tempo, e uma parte dinâmica, que se modifica à medida que o tempo transcorre. Diagramas dinâmicos são muito utilizados como sinóticos em simulações, por representarem fenômenos dependentes do tempo, tais como sistemas dinâmicos de maneira análoga à que ocorre em processos reais do mundo físico. Da mesma maneira, o uso de diagramas dinâmicos pode representar de maneira mais eficiente a riqueza de interações entre agentes.

2000, Gomes 2000] e da Rede Semiônica [Gudwin 2003]. Uma Rede de Processamento de Recursos (RP-Net) é uma ferramenta matemática formal que se destina a modelar sistemas que envolvem o processamento de recursos de qualquer natureza, transformando e/ou consumindo esses recursos de acordo com a dinâmica do sistema. Esse processamento de recursos pode ocorrer de maneira paralela e concorrente, podendo ser realizada por múltiplos agentes processadores simultaneamente [Tatai 2003].

A RP-Net é um diagrama dinâmico, com uma estrutura estática, constituída por um conjunto de lugares conectados por arcos, e uma estrutura dinâmica, constituída por um conjunto de recursos, ativos ou passivos, que ocupam os lugares da estrutura estática. Recursos ativos e passivos interagem de acordo com a conexão entre os lugares.

Cada recurso é único, identificado pelo seu nome e capaz de armazenar dados em sua estrutura interna. Um recurso capaz de manipular outros recursos é denominado de recurso ativo; caso contrário, é denominado de recurso passivo. Recursos passivos são entidades manipuladas, como por exemplo, recursos materiais (peças, matérias-primas, componentes) ou recursos de informação (mensagens, textos, dados, etc). Um recurso ativo possui as seguintes características:

- é capaz de executar tarefas que envolvam ou não outros recursos, ou seja, executa atividade de processamento de recursos (os recursos processados podem ser passivos ou ativos);
- é capaz de selecionar outros recursos de um dado conjunto, dos quais depende para executar suas tarefas;
- é capaz de destruir ou preservar esses recursos escolhidos, dependendo da tarefa;
- uma vez de posse desses recursos, realiza operações internas, de forma a gerar outros recursos;
- é capaz de gerar novos recursos, a partir da assimilação e transformação do conteúdo dos recursos previamente selecionados;
- realiza incessantemente esta atividade de escolha, assimilação e geração de novos recursos, de maneira autônoma e independente;
- pode possuir diferentes maneiras de escolher e processar os recursos em seu ciclo operacional.

Um exemplo de topologia de uma RP-Net pode ser vista na Figura 5.1.

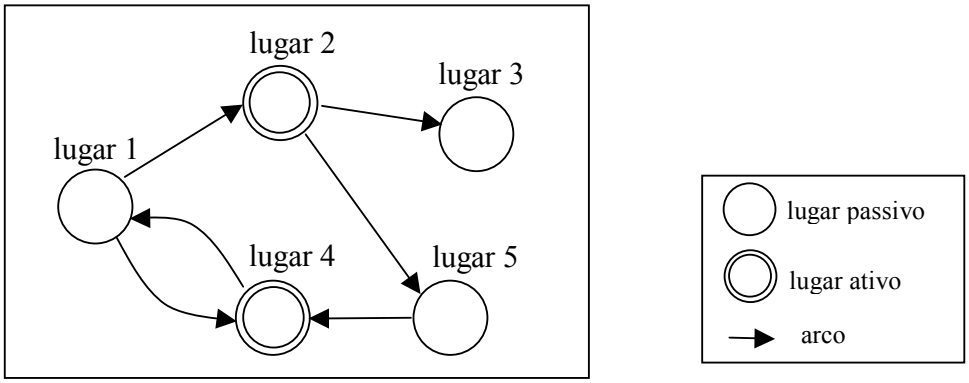


Figura 5.1: Topologia de uma RP-Net [Tatai 2003].

A Figura 5.2 mostra os lugares ativos e passivos, bem como exemplifica a posição que os recursos ativos e passivos ocupam dentro da RP-Net.

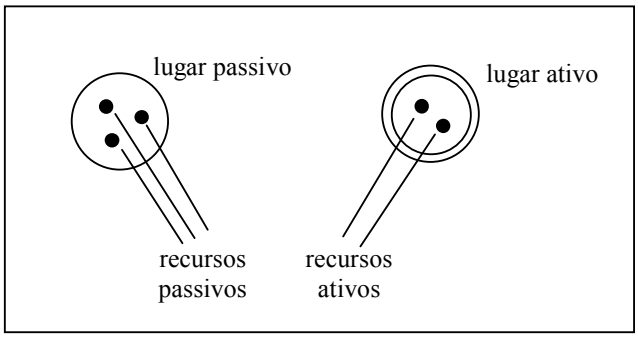


Figura 5.2: Lugares passivos e ativos, recursos ativos e passivos.

Um arco é definido, a grosso modo, como uma conexão entre uma porta de saída (ligada ao lugar de origem) e uma porta de entrada (ligada ao lugar de destino), conforme ilustrado na Figura 5.3.

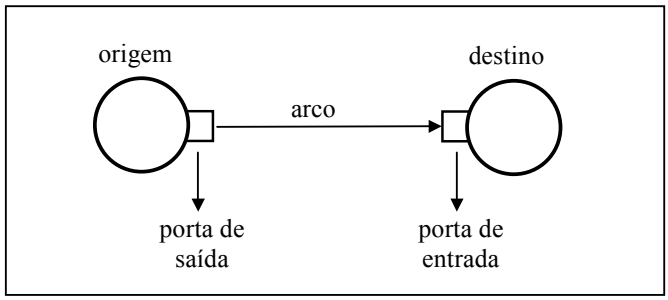


Figura 5.3: Portas de entrada e saída de uma RP-Net.

A Figura 5.4 mostra um exemplo de uma RP-Net com lugar ativo, lugares passivos, recurso ativo, recursos passivos, portas de entrada, portas de saída e arcos:

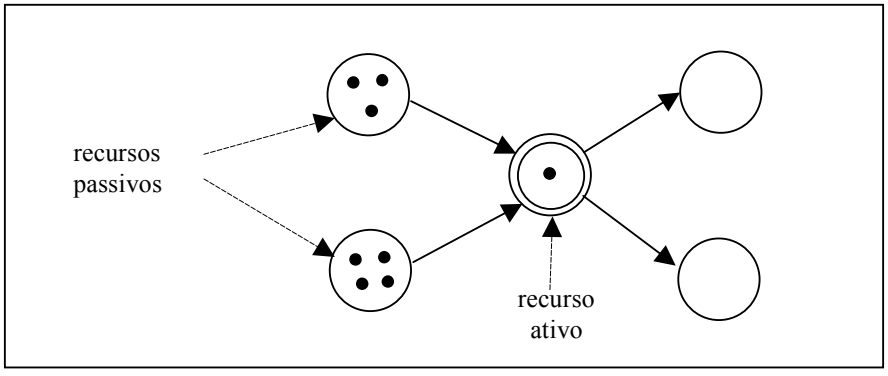


Figura 5.4: Exemplo de uma RP-Net.

A seguir são apresentados os conceitos formais da RP-Net elaborados por Tatai [Tatai 2003]:

Definição 5.1 - Lugar

Um lugar é uma entidade passiva p_j, $j \in N$, $j>0$ que agrega um ou mais recursos. O conjunto de todos os lugares é dado por $P = \{p_1,p_2, ...,p_n\}$.

Definição 5.2 - Portas de um lugar

Cada lugar possui um conjunto finito de portas de entrada e de portas de saída. Assim, $g_e(p,n)$, $n \in N$, $n> 0$ denota a n-ésima porta de entrada do lugar p. Da mesma forma, $g_s(p,n)$, $n \in N$, $n> 0$ denota a n-ésima porta de saída do lugar p.

Definição 5.3 - Arco

Um arco $a = (g_s(p_j,m),g_e(p_i,n))$ é um arco conectando a m-ésima porta de saída do lugar p_j à n-ésima porta de entrada do lugar p_i. O conjunto de todos os arcos é $A = \{a_1,a_2,...,a_q\}$.

Definição 5.4 - Recursos

O conjunto de recursos de uma rede é definido por:
$R = \{r_1,r_2, ...,r_q\}$ onde cada r_i, $i \in N$, $i> 0$ corresponde a um recurso.

Definição 5.5 - Marcação

Uma marcação M é definida como uma função $M:R \rightarrow P$, que associa recursos a lugares.

Definição 5.6 - RP-Net

Uma RP-Net é uma ênupla $RPN = \{P,A,M\}$ na qual:
$P = \{p_1,p_2, ...,p_m\}$ é o conjunto finito de lugares.
$A = \{a_1,a_2,...,a_n\}$ é o conjunto finito de arcos.
M é a marcação inicial da rede.

Definição 5.7 – Recurso Ativo

Um recurso $r \in R$ é chamado ativo quando é capaz de executar atividades de criação, transformação e destruição de outros recursos. A atuação dos recursos ativos em uma RP-Net é que confere à mesma seu comportamento dinâmico.

5.3.1 RP-Net e Engenharia de Software Orientada a Agentes

As noções de recurso ativo e passivo são abstrações bastante abrangentes em termos de modelagem. Considerando-se os conceitos da Engenharia de Software Orientada a Agentes, os agentes podem ser modelados como recursos ativos e as mensagens trocadas entre os agentes participantes de um Sistema Multi-Agentes podem ser modeladas como recursos passivos.

Assim, agentes e mensagens estão distribuídos em lugares da rede, onde cada lugar possui diferentes portas e os lugares estão conectados por arcos que ligam duas portas em diferentes lugares. Os agentes executam seus papéis capturando as mensagens por meio de sua interface de entrada, denominada de porta de entrada e gera outro conjunto de mensagens por meio de sua interface de saída, denominado de porta de saída, conforme pode ser visto na Figura 5.5. O agente, situado no lugar com linhas duplas, busca a mensagem a partir do lugar à sua esquerda e gera uma nova mensagem que é colocada no lugar à sua direita. Os lugares onde as mensagens são armazenadas são denominados de *buffers*. Os lugares onde os agentes são armazenados são denominados de agências.

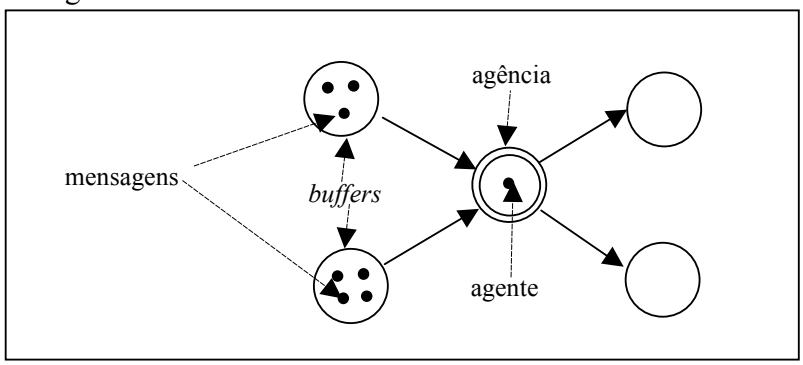


Figura 5.5: Exemplo da RP-Net com agentes e mensagens (adaptado de Gudwin [Gudwin 2004]).

Esse procedimento de busca por mensagens e geração de novas mensagens seria simples se existisse somente um agente e uma mensagem. Porém, quando se trata de um sistema Multi-Agentes com vários agentes sensoreando o ambiente em busca de mensagens e atuando no ambiente, gerando novas mensagens, esta tarefa começa a ficar bastante complexa, principalmente pelo fato de que os agentes podem competir entre si pelos mesmos recursos para atingir seus objetivos. A Figura 5.6(a) mostra um agente decidindo qual mensagem, situada em um mesmo lugar, deverá consumir. A Figura 5.6(b) mostra um agente decidindo qual mensagem, situadas em lugares diferentes, deverá consumir. A Figura 5.6(c) mostra dois agentes, situados em lugares diferentes, competindo pela mesma mensagem.

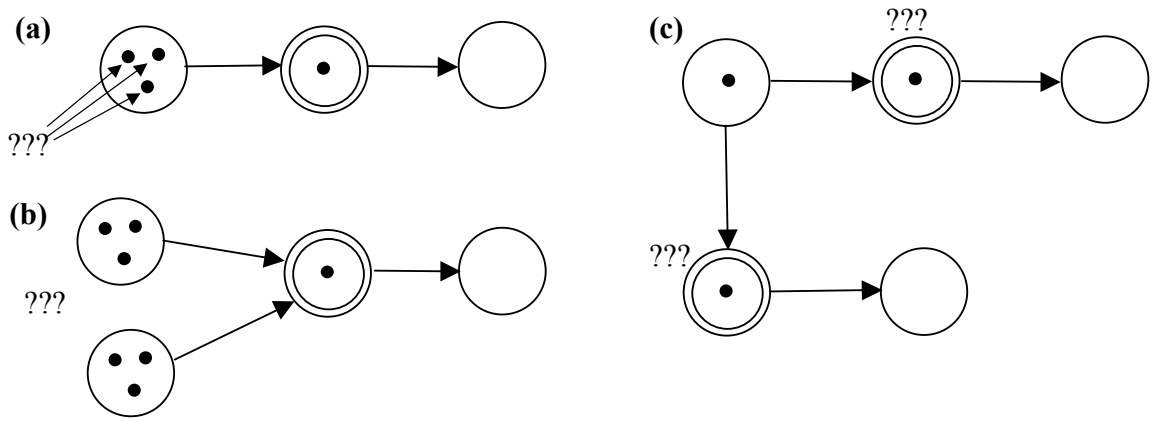


Figura 5.6: Exemplos de situações de conflito dos agentes.

Para modelar o comportamento dinâmico do sistema, cada agente necessariamente passa por etapas funcionais importantes: decisão e ação, as quais serão detalhadas a seguir.

5.3.1.1 Etapa de Decisão

Na etapa de decisão, é necessário escolher qual mensagem (ou conjunto de mensagens, quando houver) é a mais interessante para aquele agente em particular e o que deve acontecer com essa mensagem após seu processamento, se ela deve ser preservada ou não. Essa tarefa não é fácil, pois podem existir muitas mensagens disponíveis para aquele agente e também podem existir vários agentes interessados na mesma mensagem. Assim, a etapa de decisão deve ser executada de maneira coordenada, permitindo mensagens paralelas e concorrentes. A etapa de decisão é dividida em duas sub-fases: a fase de avaliação e a fase de atribuição.

A fase de avaliação inicia quando o agente se depara com várias mensagens disponíveis para interação e precisa decidir qual mensagem utilizar, e ao mesmo tempo decidir o que deverá acontecer com ela após a sua utilização: a mensagem pode ser modificada e enviada para um lugar diferente, pode retornar ao seu lugar original ou pode ser destruída após a interação. Todas as combinações possíveis de mensagens disponíveis devem ser avaliadas, sendo que cada combinação possível é chamada de escopo habilitante para uma dada "função de transformação" (f_trans), sendo basicamente uma lista de potenciais mensagens para interação. A fase de avaliação termina quando o agente avalia todos os escopos habilitantes possíveis e atribui para cada um, um valor de interesse (f_eval) e o modo de acesso pretendido. O modo de acesso diz respeito às intenções do agente com as mensagens, informando basicamente se é permitido o compartilhamento da mensagem com outros agentes e se o agente pretende destruir a mensagem após a interação. Assim, os possíveis modos de acesso permitidos pela RP-Net são: compartilhamento não-exclusivo sem consumo; compartilhamento não-exclusivo com consumo; compartilhamento exclusivo sem consumo e compartilhamento exclusivo com consumo (Figuras 5.7 a 5.10). Um único recurso pode ser utilizado para disparar duas atividades simultaneamente.

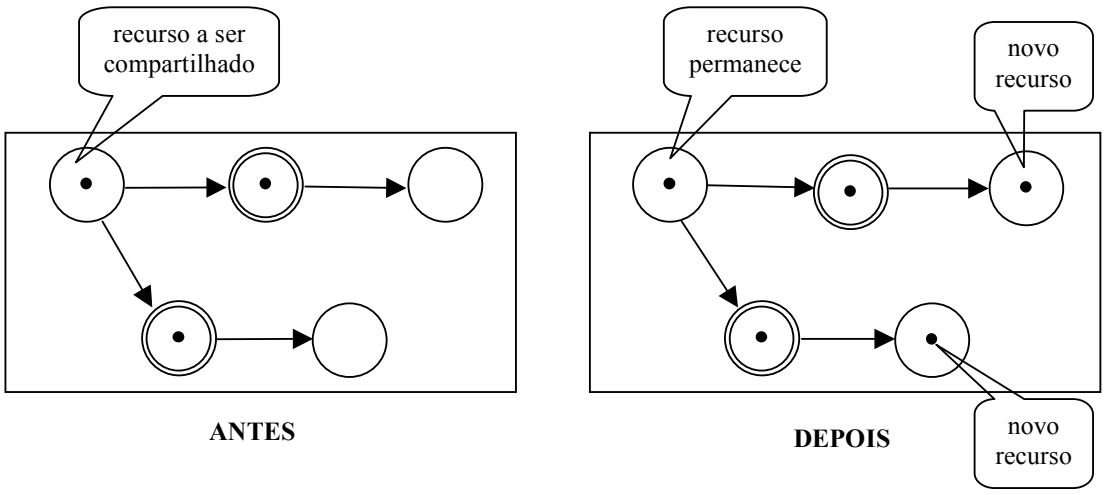


Figura 5.7: Exemplo do modo de acesso "compartilhamento não-exclusivo sem consumo".

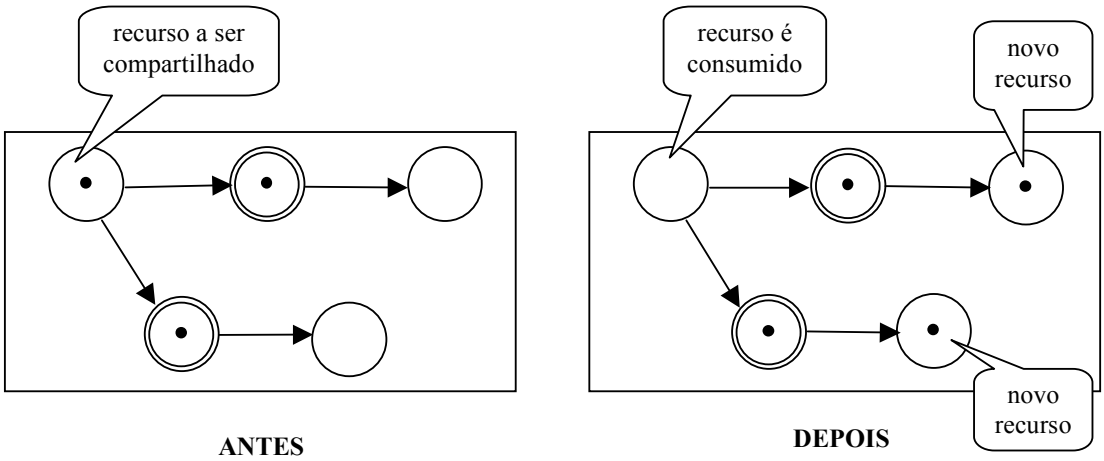


Figura 5.8: Exemplo do modo de acesso "compartilhamento não-exclusivo com consumo".

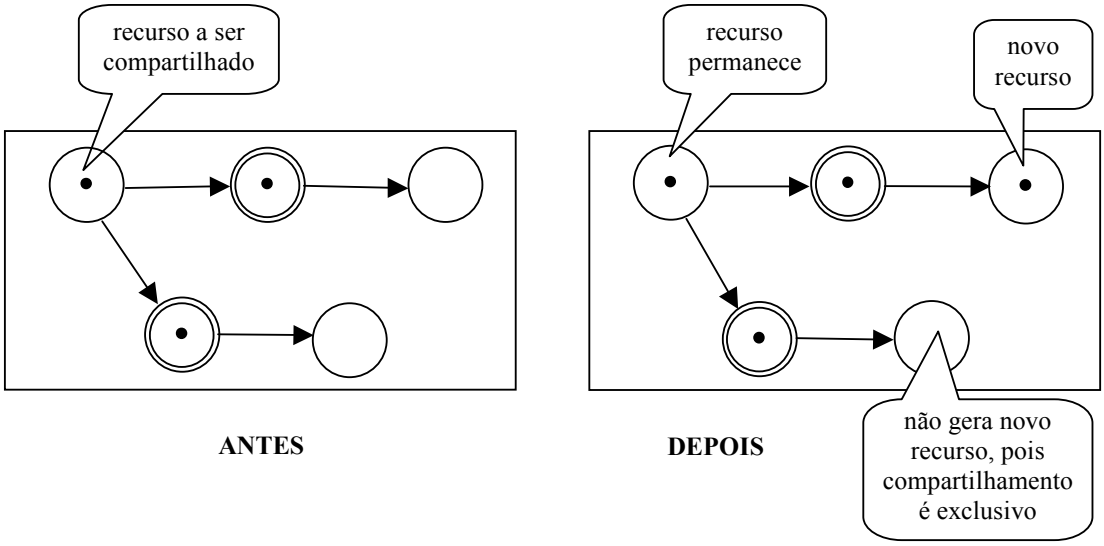


Figura 5.9: Exemplo do modo de acesso "compartilhamento exclusivo sem consumo".

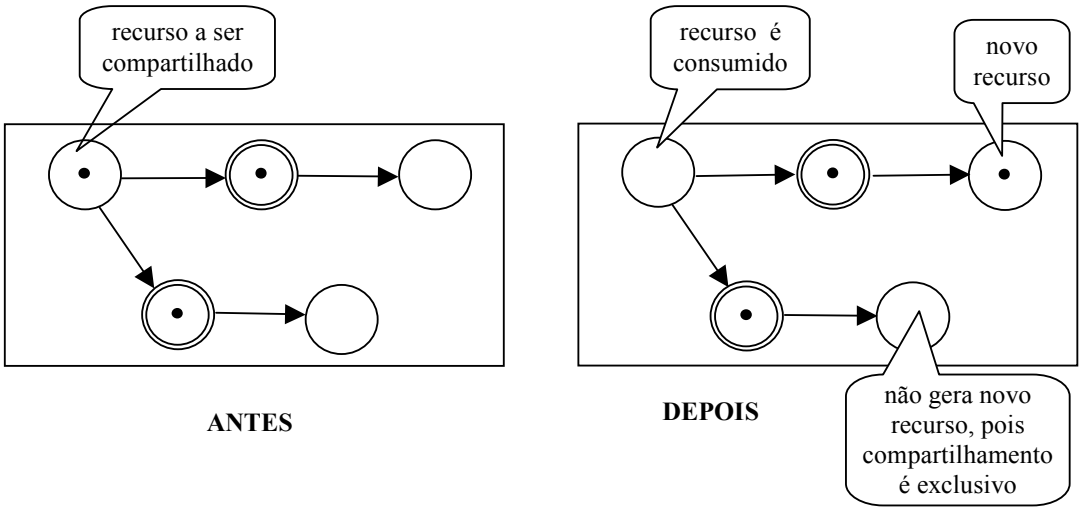


Figura 5.10: Exemplo do modo de acesso "compartilhamento exclusivo com consumo".

Já a fase de atribuição é responsável por resolver os possíveis conflitos com os planos de cada agente na rede. Com essa finalidade, foi desenvolvido por Guerrero *et al.* [Guerrero 1999] um algoritmo, denominado de BMSA (*Best Matching Search Algorithm*), que após coletar os graus de interesse de todos os agentes do sistema, atribui a mensagem ao agente que obteve a melhor pontuação, respeitando o modo de acesso pretendido pelo agente.

5.3.1.2 Etapa de Ação

Na etapa de ação, o agente efetivamente utiliza as mensagens, podendo criar uma nova mensagem, destruir ou modificar a mensagem selecionada. Essa etapa é dividida em duas sub-fases: a fase de assimilação e a fase de geração. Na fase de assimilação, o agente decide qual ação tomar, dependendo do modo de acesso definido na etapa de decisão. Dependendo do modo de acesso, o agente irá ler ou obter as mensagens através de suas portas de entrada. Após assimilar a informação necessária, o agente, dependendo do seu plano, pode deixar a mensagem no seu estado original, destruí-lo ou capturar a mensagem para gerar um processamento. Finalmente, na fase de geração, o agente poderá gerar novas mensagens ou modificar a informação assimilada, se for o caso. Essa informação é então processada, aplicando uma função de transformação que gera novas mensagens, as quais serão então enviadas para lugares apropriados na rede.

5.3.2 Implementação

Com o objetivo de testar a funcionalidade da RP-Net, o grupo de pesquisa em Semiótica Computacional do DCA/FEEC/UNICAMP implementou um software, chamado de RPNToolkit [Tatai 2003]. A RPNToolkit é uma ferramenta geral específica para o design e simulação das redes de processamento de recursos. Foi desenvolvida a partir do SNToolkit [Gudwin 2003], a qual foi remodelada e re-estruturada para incorporar os avanços introduzidos com as definições das redes de processamento de recursos. O SNToolkit é uma ferramenta para o desenvolvimento das Redes Semiônicas, bem como suas precursoras, a Rede de Agentes e a Rede de Objetos (ONToolkit) [Gomes 2000, Guerreiro 2000].

O comportamento da RP-Net é muito parecido com o de redes de Petri [Murata 1989]. Os sistemas de processamento de recursos poderiam perfeitamente ser modelados por Redes de Petri, mais especificamente Redes de Petri Estendidas, tais como Redes de Petri Coloridas ou Redes de Petri Orientadas a Objetos. Porém, as RP-Nets apresentam algumas vantagens sobre as Redes de Petri. Nas Redes de Petri, os recursos ativos seriam parcialmente modelados por transições e *tokens* (que guardariam informações a respeito do processamento de recursos). Nas RP-Nets optou-se por se discriminar explicitamente os autores das ações executadas, informação que não é explicitada nas Redes de Petri. Uma vez que o uso destas redes visa modelar a interação entre agentes, as RP-Nets mostram-se portanto mais convenientes nesse caso. Outra vantagem importante das RP-Nets é a possibilidade de se modelar processos de aprendizagem e adaptação, o que com dificuldade poderia ser modelado em Redes de Petri [Tatai 2003].

5.3.3 Aplicações

A RP-Net possui uma vasta gama possível de aplicações. Dentre outras, podemos citar sua utilização no desenvolvimento de oponentes inteligentes em jogos de computador [Tatai 2003], em redes semiônicas para modelagem de organizações empresariais, modelagem de sistemas flexíveis de manufatura, modelagem de sistemas híbridos de inteligência computacional e modelagem de sistemas de controle de robôs autônomos [Gudwin 2003, Gudwin 2004].

5.3.4 Flexibilizando a Modelagem

A RPNToolkit possui duas funções que possibilitam a flexibilidade da modelagem, denominadas de set_balance e set_max_firecount. Essas funções permitem que vários recursos passivos (as mensagens), situados em um mesmo lugar possam ser consumidos, no mesmo instante, pelo recurso ativo (o agente). Basicamente, permite-se flexibilizar a quantidade de valor que um recurso ativo poderá atribuir a um recurso necessário a uma tarefa (set_balance), de modo que durante a função de avaliação o recurso ativo consiga ganhar a posse dos recursos necessários a efetuar diversos disparos em uma mesma iteração. Set_max_firecount é utilizada para determinar o número máximo de vezes que o recurso ativo pode disparar. Isso pode ser útil quando se deseja garantir que haverá apenas um disparo. As Figuras 5.11 a 5.13 mostram exemplo onde existem vários recursos situados no mesmo lugar e que serão consumidos pelo recurso ativo

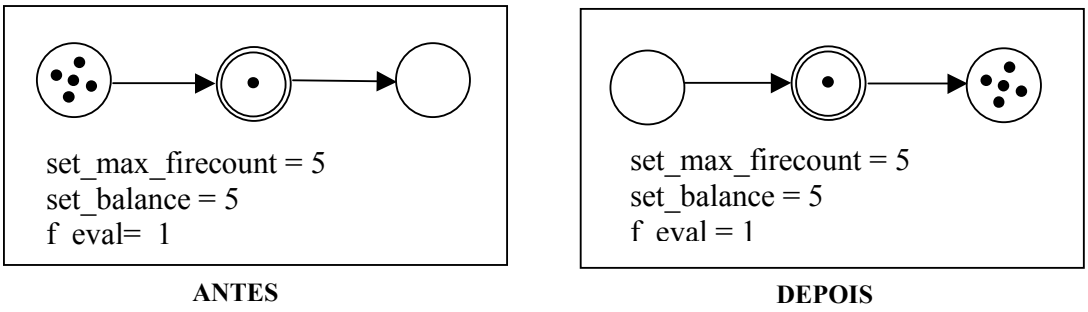


Figura 5.11: Exemplo de uso das técnicas set_balance e set_max_firecount.

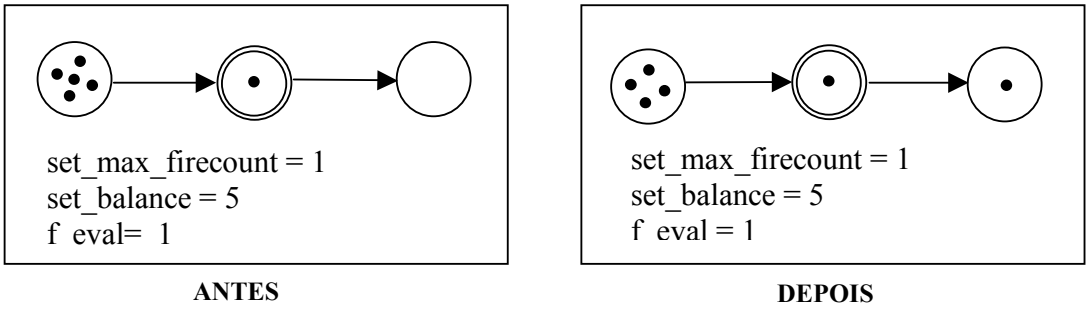


Figura 5.12: Exemplo de uso das técnicas set_balance e set_max_firecount.

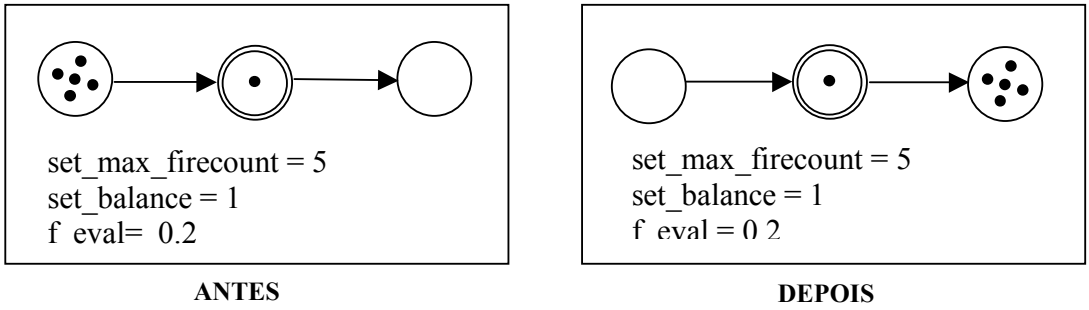


Figura 5.13: Exemplo de uso das técnicas set_balance e set_max_firecount.

A RP-Net também permite a flexibilização da representação da interação do agente, conforme o conceito proposto neste trabalho, onde o agente busca as informações no *buffer* por meio de seus sensores e armazena informações no *buffer* por meio de seus atuadores. As mensagens podem estar todas centralizadas em um *buffer* único ou estarem distribuídas em *buffers* descentralizados. A idéia de um repositório único equivale ao conceito de *blackboard*, onde os dados compartilhados são acessíveis a todos os agentes, conforme Figura 5.14. Diferentemente do conceito de *blackboard*, em um sistema multi-agentes, os agentes também podem buscar informações que estão distribuídas em vários *buffers* espalhados no ambiente, conforme Figura 5.15.

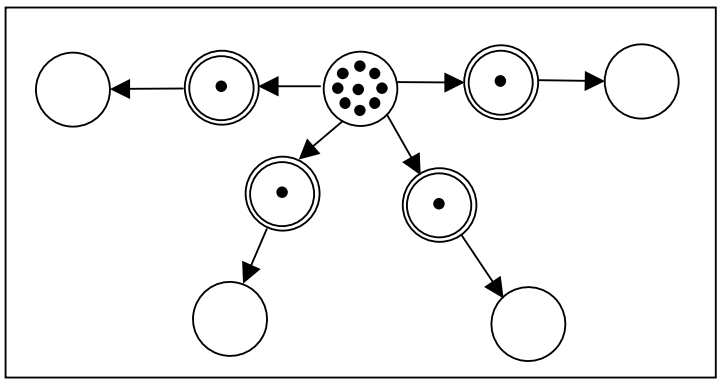


Figura 5.14: Exemplo de um sistema com buffer centralizado.

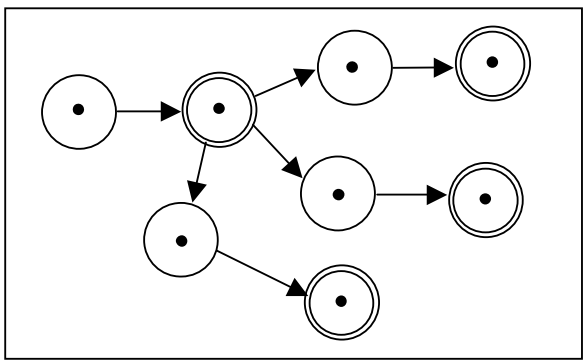


Figura 5.15: Exemplo de um sistema com vários buffers.

5.4 Resumo

Neste capítulo foram identificadas algumas limitações existentes nas metodologias MaSE, Prometheus e Tropos, no que tange a modelagem de interação entre os agentes. Foi proposta a utilização da RP-Net como um novo artefato para suprir essa necessidade da Engenharia de Software Orientada a Agentes. Para o entendimento da RP-Net, apresentamos suas definições formais e detalhamos seu funcionamento.

Finalmente, mostramos um breve histórico da RP-Net e da ferramenta para sua implantação, o RPNToolkit.

Capítulo 6

Metodologia Unificada para o Desenvolvimento de Sistemas Orientados a Agentes

6.1 Introdução

Neste capítulo apresentamos nossa proposta para uma metodologia de desenvolvimento de sistemas orientados a agentes, denominada de Metodologia Unificada. Essa metodologia foi desenvolvida a partir dos estudos realizados com as metodologias MaSE, Prometheus e Tropos e da linguagem de modelagem AUML, constituindo-se de uma unificação dos principais artefatos identificados. A finalidade da metodologia proposta é a definição de um conjunto de passos e artefatos a serem utilizados durante a construção de um sistema multi-agentes, abrangendo todas as fases do ciclo de vida do desenvolvimento do software.

Com o objetivo de modelar a interação entre os agentes de uma forma mais explícita, foi necessária a inclusão de um novo artefato, a RP-Net, apresentada no Capítulo 5. Esse artefato é uma ferramenta de modelagem de Sistemas Multi-Agentes utilizada na fase de projeto. A Seção 6.2 apresenta os critérios que levaram à seleção dos artefatos. Na Seção 6.3, é apresentada uma avaliação comparativa das metodologias apresentadas no Capítulo 4. Na Seção 6.5, a Metodologia Unificada é descrita detalhadamente, incluindo suas fases e artefatos. E, finalmente, na Seção 6.6 é apresentado o resumo do capítulo.

6.2 Critérios

Para a seleção de quais artefatos de cada metodologia seriam os mais adequados para compor a Metodologia Unificada, foi necessária a utilização de critérios sistemáticos de avaliação, além de uma análise de similaridade entre eles. Os critérios utilizados neste trabalho foram adaptados do trabalho proposto por Dam [Dam 2003], apresentados na Seção 4.6. Basicamente, eles estão divididos em três categorias: conceitos, linguagem de modelagem e processo. Os critérios relacionados aos conceitos verificam se a metodologia aborda as principais características dos agentes (autonomia, pró-atividade, reatividade, protocolos de conversação). Os critérios da categoria de linguagem de modelagem abordam a usabilidade da metodologia (clareza / entendimento e facilidade de uso) e critérios técnicos (rastreabilidade). Com relação aos critérios de processos, são levadas em consideração as fases de desenvolvimento de sistema cobertas pela metodologia. Esses critérios são detalhados a seguir:

- **Autonomia**: esse critério está relacionado com a definição de agência discutida na Seção 2.3. Mais especificamente, verifica se a metodologia suporta a característica de

autonomia do agente. Por exemplo, as funcionalidades e as tarefas encapsuladas no agente podem aumentar seu grau de autonomia.

- **Identificação do Agente**: esse critério verifica se a metodologia possui meios explícitos para a identificação dos agentes.
- **Pró-atividade:** esse critério avalia se a metodologia possui técnicas de modelagem de objetivos, capturando os objetivos do sistema e do agente e se a metodologia fornece modelos que detalham como o agente atinge seu objetivo.
- **Reatividade:** verifica se a metodologia fornece mecanismos para representar as mudanças do ambiente (como por exemplo, eventos/incidentes) e para especificar e representar as respostas do agente às mudanças de ambiente.
- **Protocolos:** esse critério examina se a metodologia fornece modelos para definir a especificação de protocolos que caracterize a interação entre os agentes.
- **Clareza e entendimento:** se os símbolos e a sintaxe da metodologia são bem definidos e se os modelos permitem vários tipos de abstrações.
- **Facilidade de uso:** se a notação utilizada pela metodologia é familiar ao usuário, se é fácil de desenhar ou escrever e se os diagramas produzidos pela metodologia são limpos (sem anotações desnecessárias).
- **Rastreabilidade:** se a metodologia permite o rastreamento entre seus modelos e se existe uma ligação clara entre as fases da metodologia.
- **Fases de desenvolvimento:** quais são as fases de desenvolvimento suportadas pela metodologia.

6.3 Avaliação comparativa das Metodologias

Com base nos critérios descritos acima, foi possível fazer uma avaliação comparativa entre as metodologias MaSE, Prometheus e Tropos, apresentadas no Capítulo 4. Embora tenha sido feito o estudo da AUML, ela não fez parte dessa avaliação. A AUML não é uma metodologia propriamente dita, mas os seus diagramas são muito utilizados em várias metodologias orientadas a agentes, principalmente nas que surgiram como uma especialização das abordagens orientadas a agentes.

Essa avaliação é uma avaliação preliminar elaborada a partir de somente um estudo de caso, utilizando-se de alguns critérios subjetivos baseados na experiência da autora desse trabalho. Considerando-se que as metodologias estão em constante desenvolvimento pode-se aqui estar fazendo injustiças com uma ou outra metodologia que com certeza estão sendo aprimoradas. Como estudo de caso utilizamos um Sistema Multi-Agente de Busca de Informações Científicas utilizando as três metodologias estudadas (Anexo B). A tabela 6.1 mostra o resultado obtido dessa avaliação. Foram assumidos três valores para a avaliação das metodologias.: o sinal "+" significa que a metodologia atende parcialmente ao critério, o sinal "++" significa que a metodologia atende a todos os requisitos do critério e o sinal "+++" significa que a metodologia supera os requisitos do critério. Por exemplo, o critério de identificação do agente não é abordado claramente pela metodologia MaSE sendo avaliada como "+". Esse critério é contemplado pela metodologia Tropos, portanto, foi avaliada como "++". Porém, a metodologia Prometheus supera os requisitos desse critério, recebendo a avaliação de "+++".

Com base nos resultados obtidos no estudo de caso, obtivemos subsídios técnicos necessários para a seleção dos artefatos que irão compor a Metodologia Unificada proposta.

Tabela 6.1: Resultado da avaliação das metodologias de acordo com os critérios. Os resultados podem ser + (atende parcialmente), ++ (atende os requisitos) e +++ (supera os requisitos).

Critérios	MaSE	Prometheus	Tropos
Autonomia	+++	+++	+++
Identificação do Agente	+	+++	++
Pró-atividade	+++	+++	+++
Reatividade	+	+++	+
Protocolos / Interação	+	+	+
Clareza / Entendimento	++	++	++
Facilidade de uso	++	++	+
Rastreabilidade	++	+++	++
Fases de desenvolvimento	++	+++	+++

A seguir serão detalhados os motivos, baseados nos critérios acima, que levaram à escolha de cada artefato existente em cada fase da Metodologia Unificada:

- **Diagramas de Ator e de Objetivos**: como a fase de levantamento dos requisitos iniciais do sistema é explicitamente detalhada somente na metodologia Tropos, através dos Diagramas de Ator e de Raciocínio, esses artefatos foram selecionados e adaptados para a metodologia proposta. Eles foram escolhidos de acordo com os critérios de pró-atividade e de fases de desenvolvimento da metodologia.
- **Diagrama Hierárquico de Objetivos**: esse diagrama, adaptado da metodologia MaSE, faz parte da identificação dos objetivos do sistema, satisfazendo o critério de pró-atividade.
- **Modelo de Funcionalidades**: as três metodologias suportam o conceito básico da orientação a agentes, a autonomia, representada pelas funcionalidades e tarefas encapsuladas no agente. Uma técnica comum utilizada por essas metodologias para tratar com a identificação do agente é começar identificando as entidades menores que agentes, ou seja, identificar as capacidades na Tropos, as funcionalidades no Prometheus e os papéis na MaSE. Neste trabalho, optou-se por fazer uma adaptação do Descritor de Funcionalidades da metodologia Prometheus, gerando o Modelo de Funcionalidades, pois abrange várias informações sobre as funcionalidades que o agente pode assumir.
- **Diagrama de Agrupamento de Funcionalidades**: Prometheus é a metodologia que apresenta uma técnica clara que atende ao critério de identificação do agente, pelo Diagrama de Ligação de Dados, o qual converte funcionalidade em agente. Por essa razão, na Metodologia Unificada fizemos uma adaptação desse diagrama gerando o Diagrama de Agrupamento de Funcionalidades.

- **Modelo de Descritor do Agente**: as metodologias estudadas apresentam técnicas para representar os objetivos do agente (critério de pró-atividade) e quais são as respostas do agente às mudanças do ambiente (critério de reatividade); porém, são melhor detalhados pelos Descritores do Agente, de Plano e de Evento da metodologia Prometheus. Neste trabalho, foi feita uma adaptação desses artefatos, gerando o Modelo de Descritor do Agente, contendo essas informações em um único artefato, de maneira mais resumida.
- **Diagrama de RP-NET**: com relação aos protocolos de conversação, todas as metodologias possuem mecanismos de troca de mensagens entre os agentes, sendo que a maioria utiliza os artefatos da AUML. MaSE, Prometheus e Tropos utilizam os diagramas de seqüência/interação para representar as interações, sendo que Prometheus e Tropos mostram as interações entre os agentes, enquanto as interações do diagrama de seqüência da MaSE são entre papéis. Porém, como nenhuma delas aborda o conceito proposto neste trabalho, onde o agente busca as mensagens do ambiente, está sendo utilizado o Diagrama RP-NET.
- **Digrama de Revisão do Sistema**: este diagrama foi selecionado da metodologia Prometheus para compor a metodologia proposta, pois representa uma importante ferramenta de rastreabilidade do sistema em desenvolvimento, uma vez que ele faz a consistência clara entre outros artefatos utilizados durante o processo de modelagem.
- **Descritor da Estrutura Interna do Agente**: atende aos critérios de autonomia, pró-atividade e reatividade, pois possui informações específicas de cada agente, representando o processamento interno do agente, como por exemplo: os planos do agente para realizar seus objetivos. Este é um modelo novo e foi inspirado no Modelo de Funcionalidades.
- **Diagrama de Estados**: este artefato é utilizado para modelar a estrutura interna dinâmica de cada agente, satisfazendo o critério de protocolo/interação.

Todas as metodologias estudadas possuem símbolos e sintaxes bem definidas, porém algumas notações não são de fácil entendimento. Tropos apresenta uma dificuldade na utilização, pois seus artefatos não são fáceis de serem desenhados. Além disso, quando se opta por unificar várias metodologias, torna-se necessário estabelecer padrões de formatação e notação, pois cada metodologia possui a sua própria notação. Esses motivos fizeram com que durante a elaboração desse trabalho, alguns artefatos tivessem que ser adaptados para comporem a Metodologia Unificada.

A tabela 6.2 relaciona os passos da Metodologia Unificada com os artefatos selecionados das metodologias MaSE, Prometheus e Tropos, segundo os critérios adotados. Por exemplo, o passo de identificar os objetivos não é abordado pelas metodologias MaSE e Prometheus, sendo abordado pelos Diagramas de Ator e de Raciocínio na metodologia Tropos. Esses artefatos foram selecionados para comporem a Metodologia Unificada, satisfazendo os critérios de pró-atividade e de fases de desenvolvimento, porém alteramos o nome de Diagrama de Raciocínio para Diagrama de Objetivos, que nos pareceu mais apropriado. A caixa com borda azul representa que o artefato foi selecionado da metodologia Tropos. A caixa com borda amarela representa que o artefato foi selecionado da metodologia MaSE. A caixa com borda verde representa que o artefato foi selecionado da metodologia Prometheus. A caixa com borda lilás representa que o artefato foi selecionado da linguagem de modelagem AUML. E finalmente, a caixa com borda vermelha representa que é um artefato novo gerado pela Metodologia Unificada.

Tabela 6.2: Resumo dos artefatos utilizados na Metodologia Unificada

Passos	**MaSE**	**Prometheus**	**Tropos**	**Metodologia Unificada**
Identificar os objetivos	Não possui	Não possui	Diagrama de Ator Diagrama de Raciocínio	Diagrama de Ator Diagrama de Objetivos (pró-atividade, fases)
Estruturar os objetivos	Diagrama Hierárquico de Objetivos	Não possui	Diagrama de Ator Diagrama de Raciocínio	Diagrama Hierárquico de Objetivos (pró-atividade)
Identificar as funcionalidades	Modelo de Papéis	Descritor de Funcionalidades	Tabela de Capacidades	Modelo de Funcionalidades (autonomia)
Identificar os tipos de agentes	Diagrama de Classe de Agente	Diagrama de Ligação de Dados	Tabela de Tipos de Agentes	Diagrama de Agrupamento de Funcionalidades (identificação do agente)
Refinar os agentes	Não possui	Descritor do Agente	Não possui	Modelo de Descritor do Agente (pró-atividade, reatividade)
Identificar as interações entre os agentes	Diagrama de Classe de Comunicação	Diagrama de Interação (de Seqüência – AUML)	Diagrama de Interação (de seqüência – AUML)	Diagrama RP-Net (interação)
Revisar o sistema	Não possui	Diagrama de Revisão do Sistema	Não possui	Diagrama de Revisão do Sistema (rastreabilidade)
Construir a estrutura interna de cada agente	Não possui	Diagr. de Revisão do Agente Diagr. de Revisão de Capacidade Descritor de Plano	Diagr. de Capacidades	Descritor da Estrutura Interna do Agente (autonomia, pró-atividade, reatividade)
Construir a estrutura dinâmica de cada agente	Não possui	Descritor de Evento	Diagrama de Planos	Diagrama de Estados (reatividade)

6.4 Metodologia Unificada

A Metodologia Unificada, aqui proposta para o desenvolvimento de sistemas orientados a agentes, foi elaborada a partir dos estudos feitos das metodologias MaSE, Prometheus e Tropos. O principal objetivo da metodologia é conduzir o engenheiro de software durante todo o ciclo de desenvolvimento do sistema, através de um conjunto de passos, atividades e artefatos utilizados na sua construção, independentemente de qualquer arquitetura de sistemas multi-agentes ou linguagem de programação. O conjunto de atividades produz os artefatos, os quais representam qualquer informação produzida (modelos ou documentos).

Como as metodologias da Engenharia de Software Orientada a Agentes utilizam nomenclaturas diferentes para os mesmos conceitos, torna-se necessário para o entendimento da metodologia proposta descrever quais os conceitos adotados:

- **Agente**: sistema de computador situado em um ambiente, sendo capaz de agir de maneira autônoma em seu ambiente para atingir seus objetivos [Wooldridge 1999a]. Os agentes devem seguir o Modelo de Interação de Busca por Mensagens proposto neste trabalho.
- **Ator**: representa as entidades externas ao sistema que possuem objetivos estratégicos.
- **Objetivo**: representa os interesses dos atores envolvidos no sistema, bem como as metas que o agente deve cumprir. Esses objetivos são divididos em objetivos e objetivos-leves (*softgoals*). Os objetivos leves são os requisitos que não estão claramente definidos e que estão relacionados aos requisitos de qualidade do sistema, como responsabilidade, flexibilidade, adaptabilidade e interações com o ambiente.
- **Funcionalidade**: o conceito de funcionalidade utilizado na Metodologia Unificada equivale ao conceito utilizado na metodologia Prometheus e ao conceito de papéis da MaSE. Um agente pode representar várias funcionalidades e vários agentes podem representar a mesma funcionalidade. Uma funcionalidade pode ser decomposta em planos, eventos e outras funcionalidades, gerando habilidades específicas ao agente.
- **Atividades**: conjunto de ações/planos que o agente deve executar para atingir um objetivo.

A seguir são detalhadas todas as fases da Metodologia Unificada.

6.4.1 Fases necessárias

A Metodologia Unificada consiste de três fases:

1. **Análise**: focaliza na identificação dos objetivos do sistema, incluindo suas funcionalidades básicas, suas entradas e saídas.
2. **Projeto Arquitetural**: utiliza as saídas da fase anterior para determinar quais agentes o sistema deve conter e como eles irão interagir.
3. **Projeto Detalhado**: focaliza no interior de cada agente e como ele irá realizar suas tarefas dentro do sistema.

A fase de análise é dividida em três passos: identificar os objetivos, estruturar os objetivos e identificar as funcionalidades. A fase de projeto arquitetural é composta por quatro passos: identificar os tipos de agentes, refinar os agentes, identificar as interações entre os agentes e revisar o sistema. A fase de projeto detalhado é dividida em dois passos: construir a estrutura interna estática do agente e construir a estrutura dinâmica de cada agente.

É importante destacar que ao se modelar um sistema multi-agentes, não se deve limitar a definir as funcionalidades computacionais como nas metodologias procedimentais. Deve-se identificar as responsabilidades do agente e quais as atividades que o agente deverá cumprir para realizar o objetivo. Caberá ao engenheiro de software identificar a necessidade de elaboração de cada atividade.

A Figura 6.1 ilustra todas as fases da Metodologia Unificada proposta neste trabalho.

6.4.1.1 Fase de Análise

Esta fase tem como principal objetivo o entendimento do problema, identificando as necessidades do cliente, os requisitos do sistema e especificando o sistema dentro do seu ambiente. Antes de começar a projetar o sistema é necessário discutir com os clientes, gerentes e demais *stakeholders* qual a finalidade do sistema a ser construído, identificando, modelando e analisando os principais objetivos funcionais e não-funcionais do sistema.

Assim, na Metodologia Unificada, primeiramente deve-se fazer o levantamento dos requisitos do sistema, incluindo seus principais atores (clientes), seus principais objetivos e as principais dependências entre os atores no ambiente. Em seguida, cada objetivo deve ser analisado sob o ponto de vista de um ator específico. Finalmente, são identificadas as funcionalidades básicas que irão atingir esses objetivos.

A fase de análise engloba 3 passos:

- Identificar os objetivos;
- Estruturar os objetivos;
- Identificar as funcionalidades.

a) Identificar os objetivos

Este passo busca entender o sistema e a razão de sua existência, identificando o domínio e as necessidades do cliente. Os diagramas utilizados nesse passo são o **Diagrama de Ator** e o **Diagrama de Objetivos** baseados respectivamente, no Diagrama de Ator e no Diagrama de Raciocínio da metodologia Tropos, por ser a única metodologia, entre as apresentadas neste trabalho, que trata explicitamente do levantamento dos objetivos do sistema.

O **Diagrama de Ator** modela graficamente os clientes como atores e suas intenções como objetivos, além de representar as suas dependências. Os atores são representados por círculos, os objetivos por retângulos de cantos arredondados e os objetivos-leves por retângulos com linhas duplas e cantos arredondados. Objetivos leves são os objetivos relacionados aos requisitos não funcionais dos sistemas, cujas condições não estão claramente definidas. A notação para objetivos-leves foi alterada da metodologia Tropos, que utiliza um símbolo em forma de nuvem, para facilitar a modelagem. Os relacionamentos de dependências são representados por linhas

com setas conectadas ao objetivo. A Figura 6.2 mostra um diagrama de ator com seus objetivos, objetivos-leves e os relacionamentos de dependência entre eles.

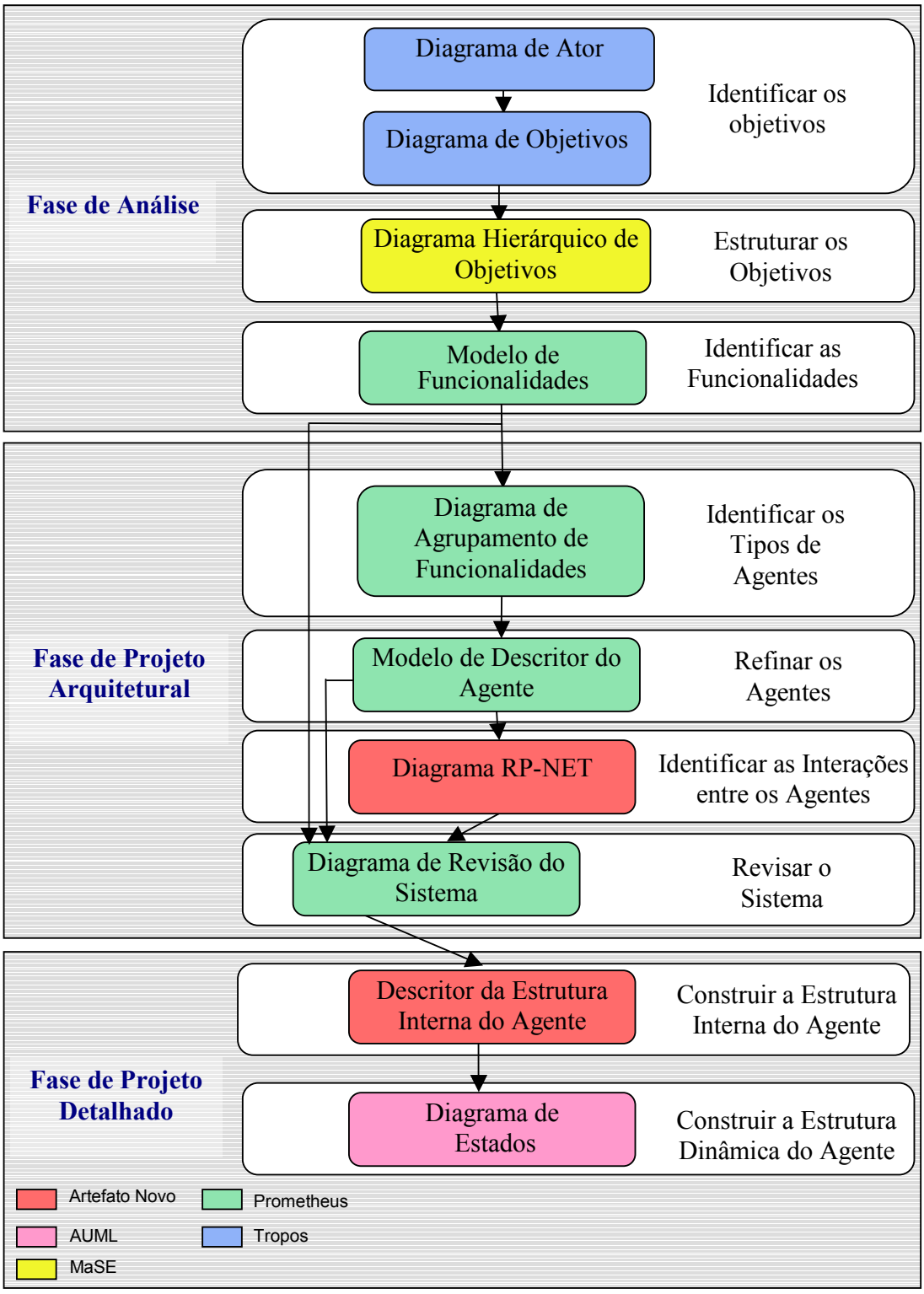


Figura 6.1: Visão geral da metodologia dividida em fases.

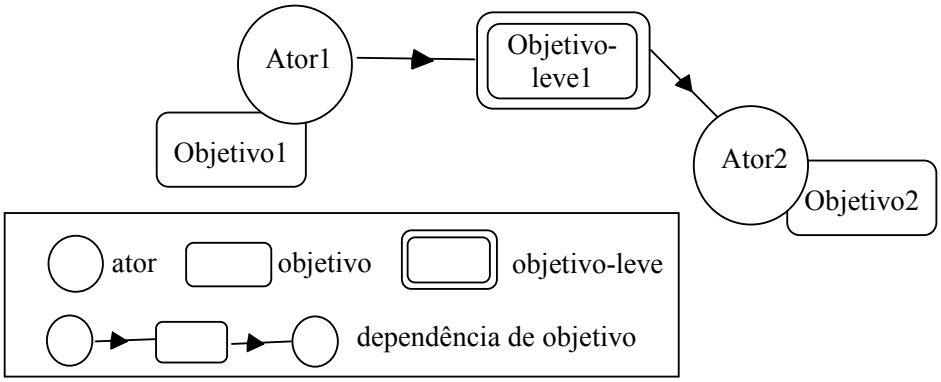


Figura 6.2: Diagrama de Ator da Metodologia Unificada.

No **Diagrama de Objetivos** o sistema a ser desenvolvido é modelado como um outro ator. Esse ator é relacionado aos atores sociais em termos de dependência. Deste diagrama é possível identificar quais os requisitos funcionais e não-funcionais (*softgoals* – objetivos leves) do sistema e como esses objetivos se relacionam. A notação é semelhante à do Diagrama de Ator, os objetivos são representados por retângulos de cantos arredondados, os objetivos-leves por retângulos com linhas duplas e cantos arredondados, o ator por um círculo e os relacionamentos de dependências são linhas com setas conectadas ao objetivo. O sinal de "+" representa uma contribuição positiva entre objetivos. A visão do ator é representada como um grande círculo contendo os objetivos, objetivos leves e seus relacionamentos.

Na Figura 6.3 são apresentados os objetivos, os objetivos leves e os relacionamentos identificados para o ator1. Por exemplo, o objetivo3 contribui para satisfazer o objetivo2 e o objetivo-leve3 e o objetivo-leve3 contribui para satisfazer o objetivo-leve1. Neste diagrama, a análise do objetivo-leve é realizada identificando os objetivos que contribuem positiva ou negativamente para satisfazê-lo. Na Figura 6.3, o objetivo-leve2 e o objetivo-leve3 contribuem positivamente para satisfazer o objetivo-leve1. O resultado final desta fase é um conjunto de dependências estratégicas entre atores, construídos de maneira incremental por meio da análise dos planos e objetivos de cada ator, até que todos os objetivos tenham sido analisados.

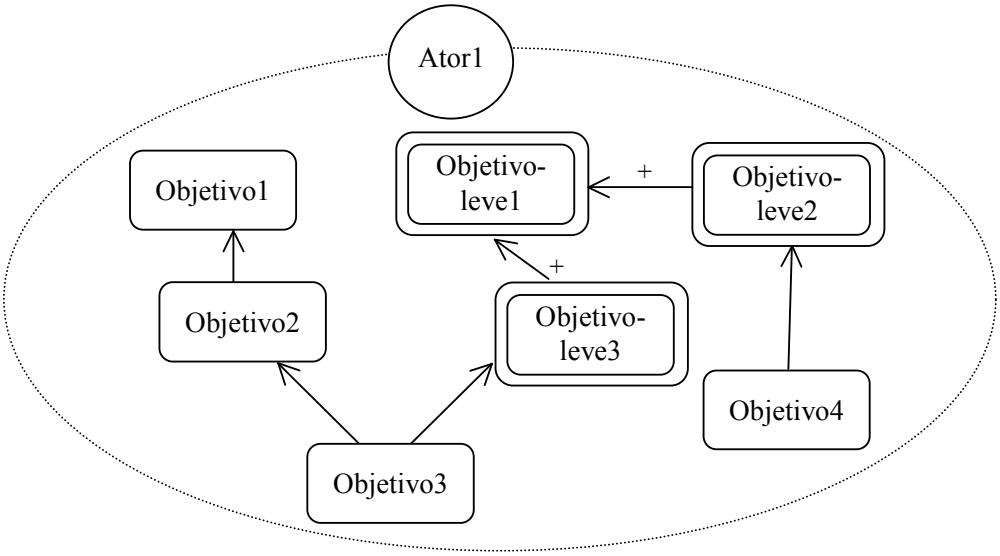


Figura 6.3: Diagrama de Objetivos da Metodologia Unificada.

b) Estruturar os objetivos

Após serem identificados os objetivos, eles devem ser refinados e organizados de maneira hierárquica, visando facilitar o entendimento do problema. O artefato utilizado pela Metodologia Unificada é o **Diagrama Hierárquico de Objetivos**, baseado no artefato de mesmo nome da metodologia MaSE. Uma melhoria nesse artefato é a inclusão de uma notação específica para modelar os objetivos-leves do sistema, os quais não eram representados na MaSE. O nome do sistema é representado por um quadrado, os objetivos por retângulos com cantos arredondados e os objetivos-leves por retângulos com linhas duplas e cantos arredondados. Cada nível da hierarquia contém os objetivos organizados pelo grau de importância dentro do sistema, ou seja, no primeiro nível está o nome do sistema e nos demais níveis estão os objetivos organizados hierarquicamente. O objetivo localizado no nível abaixo da estrutura (objetivo-filho) é necessário para satisfazer o objetivo pai. A Figura 6.4 mostra um exemplo de um Diagrama Hierárquico de Objetivos.

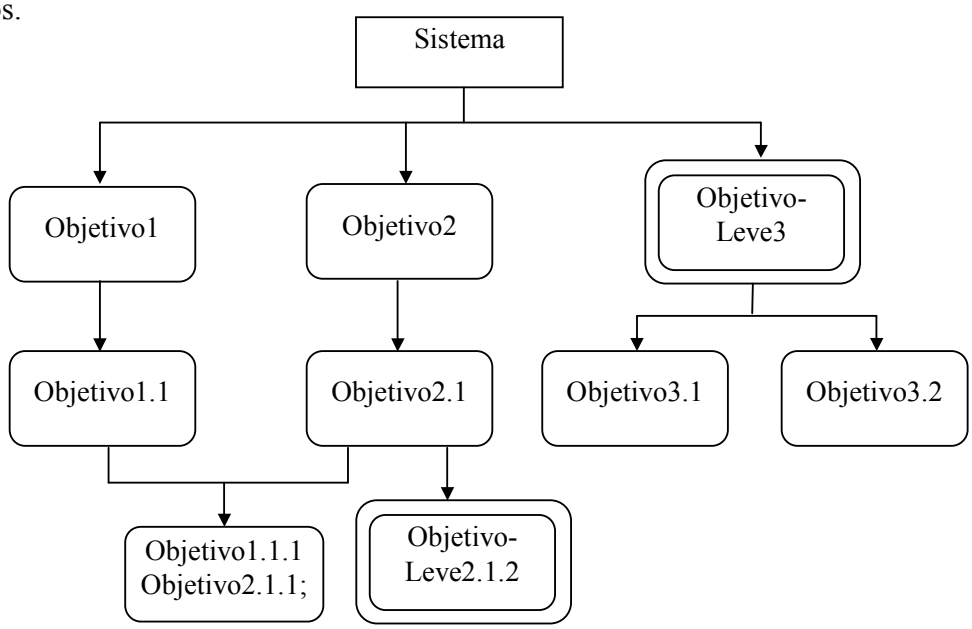


Figura 6.4: Diagrama Hierárquico de Objetivos da Metodologia Unificada.

c) Identificar as funcionalidades

Uma vez identificados e estruturados os objetivos, o próximo passo da metodologia proposta é identificar quais as funcionalidades do sistema. Uma funcionalidade representa os deveres a serem executados para que um objetivo seja atingido e os direitos a serem respeitados pelo agente.

Para auxiliar na identificação das funcionalidades, é necessário associar as funcionalidades aos objetivos, ou seja, deve-se identificar quais funcionalidades irão cumprir quais objetivos. Cada funcionalidade deve estar associada a um ou mais objetivos do sistema, enquanto cada objetivo deve estar associado a uma ou mais funcionalidades do sistema, ou seja, não existe uma relação um para um entre funcionalidade e objetivo.

Neste passo, a Metodologia Unificada utiliza o **Modelo de Funcionalidades**, adaptado da junção do Formulário de Objetivos e Funcionalidades com o Descritor de Funcionalidades da

metodologia Prometheus. Através deste artefato, são verificados os objetivos relacionados às funcionalidades, as percepções da funcionalidade, as ações que cada funcionalidade executa para atingir esses objetivos, as mensagens coletadas e enviadas pela funcionalidade, quais os *buffers* utilizados para armazenar os recursos necessários ao exercício de uma funcionalidade e quais informações cada funcionalidade necessita ou produz.

Primeiramente, deve-se atribuir um nome único que identifique facilmente a funcionalidade. Em seguida, deve-se associar um ou mais objetivos, identificados nos passos anteriores, à funcionalidade. Uma vez associados os objetivos relacionados à funcionalidade, é possível identificar as entradas possíveis para aquela funcionalidade, bem como as ações necessárias para que esta funcionalidade consiga cumprir suas metas.

Basicamente este modelo é formado por um retângulo, sendo que cada item é separado por uma linha. Os nomes dos itens devem ser em itálico e a separação entre o nome de cada item e sua especificação é feita por dois pontos. A Figura 6.5 mostra o Modelo de Funcionalidades incluindo todos os dados a serem preenchidos.

Nome: nome da funcionalidade
Descrição: pequena descrição da funcionalidade
Objetivos: quais os objetivos que a funcionalidade deve cumprir
Percepções: relação de todas as entradas (mensagens) necessárias para a funcionalidade
Ações: relação das ações executadas pela funcionalidade para atingir seus objetivos
Mensagens coletadas: mensagens coletadas pela funcionalidade
Mensagens enviadas: mensagens produzidas pela funcionalidade
Buffers de acesso: quais os *buffers* utilizados para armazenar os recursos necessários ao exercício de uma funcionalidade
Dados lidos: quais os dados lidos para esta funcionalidade atingir seus objetivos
Dados gerados: quais os dados produzidos por esta funcionalidade

Figura 6.5: Modelo de Funcionalidades.

A partir do levantamento feito no Modelo de Funcionalidades, deve-se verificar se todas as funcionalidades identificadas são realmente necessárias e se os objetivos estão relacionados com a funcionalidade adequada.

Esses três passos e seus respectivos artefatos permitem ao desenvolvedor entender os requisitos do sistema, identificando os objetivos e as funcionalidades do sistema. Essas informações servirão como entrada para a próxima fase, a de Projeto Arquitetural.

Todos os artefatos produzidos nesta fase devem ser revistos com o cliente até que todas as necessidades sejam atingidas.

6.4.1.2 Fase de Projeto Arquitetural

Nesta fase, é analisada a estrutura social estática e dinâmica dos agentes. A estrutura social estática consiste na determinação dos tipos de agentes existentes no sistema, enquanto a estrutura social dinâmica consiste na identificação das interações existentes entre os agentes.

A fase de projeto arquitetural engloba 4 passos:

- Identificar os tipos de agentes;
- Refinar os agentes;
- Identificar as interações entre os agentes;
- Revisar o sistema.

a) Identificar os tipos de agentes

Baseado nas funcionalidades definidas na fase anterior é possível dar início ao processo de identificação de tipos de agentes específicos. Esse processo consiste no agrupamento de funcionalidades relacionadas, envolvendo uma análise criteriosa das vantagens e das desvantagens geradas por esse agrupamento. Primeiramente, é necessário estabelecer qual o critério a ser utilizado para o agrupamento; exemplo: se existem funcionalidades similares, se utilizam os mesmos dados, se interagem freqüentemente, se atuam na mesma plataforma; sempre levando em consideração se a funcionalidade não sofrerá alteração em virtude do agrupamento. Outras questões importantes que devem ser tratadas durante o agrupamento das funcionalidades estão relacionadas com a segurança e a privacidade necessárias em cada funcionalidade, como por exemplo, se os dados associados a uma funcionalidade podem ser acessados por outra, ou seja, o agrupamento deve ser realizado de forma coerente.

O artefato proposto na Metodologia Unificada, responsável por modelar este agrupamento é o **Diagrama de Agrupamento de Funcionalidades**, baseado no Diagrama de Ligação de Dados da metodologia Prometheus. O nome do artefato foi alterado, uma vez que o agrupamento segue critérios que não necessariamente são de dados. Esse diagrama simplesmente agrupa as funcionalidades que estão relacionadas, representando uma técnica clara para a identificação de agentes.

As funcionalidades são representadas por retângulos chanfrados (octógono), o critério de ligação é representado por uma elipse, as ligações entre as funcionalidades e o critério de ligação são representadas por setas que indicam a direção da ligação, o agrupamento é representado por um quadrado e o nome do agrupamento é o nome do agente.

A Figura 6.6 mostra um Diagrama de Agrupamento de Funcionalidades agrupadas segundo o critério de utilização de banco de dados. No exemplo, estão agrupadas no Agente X as funcionalidades que acessam os bancos de dados 1 e 2 e no Agente Y, as funcionalidades que acessam os bancos de dados 3 e 4.

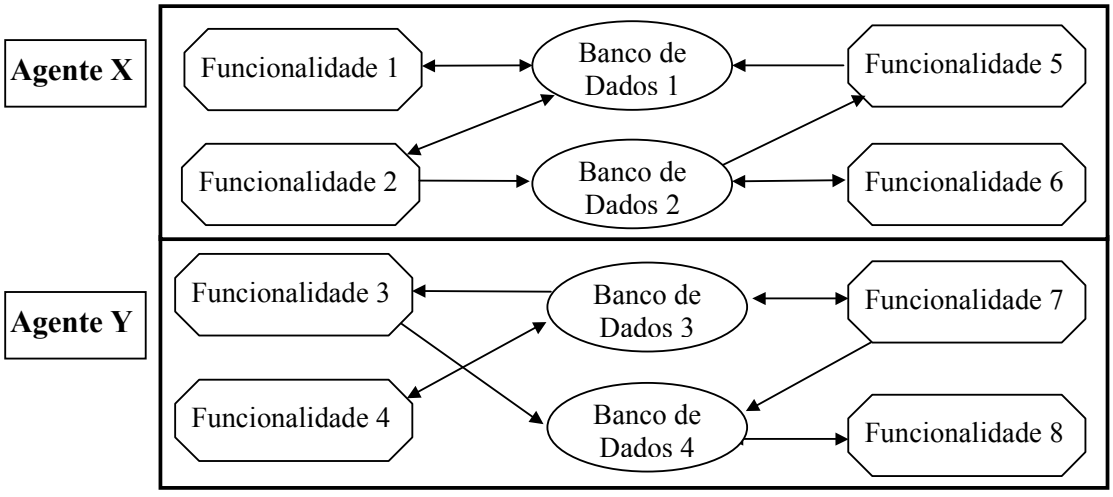


Figura 6.6: Diagrama de Agrupamento de Funcionalidades

b) Refinar os agentes

Definidos os tipos de agentes, o próximo passo é refinar os agentes, documentando os agentes identificados. Note que aqui não é analisada a arquitetura interna do agente. Somente são identificadas suas informações de alto nível e que servirão de base para a realização das próximas etapas. O artefato utilizado neste passo é o **Modelo de Descritor do Agente**. Este modelo é baseado no Modelo de Funcionalidades elaborado no passo "c" da fase de análise.

O Modelo de Descritor do Agente é formado pelo nome do agente, o qual deverá ser um identificador único do agente. O artefato contém um campo com uma breve descrição do agente, em linguagem natural. Em seguida, é necessário enumerar as funcionalidades atribuídas a este agente. Outro item importante neste artefato é documentar todos os objetivos que este agente deverá cumprir, já identificados a partir do modelo de funcionalidade e do diagrama de agrupamento de funcionalidade. Porém, outras informações devem ser tratadas durante a elaboração do Modelo de Descritor do Agente, como: Quantas instâncias de cada agente são necessárias para o sistema? Qual o ciclo de vida do agente? Quando ele deve ser inicializado? Quando ele deve ser destruído? Como o agente interage? Junto com o levantamento das interações devem ser identificadas as mensagens coletadas e produzidas pelo agente. Para facilitar o levantamento dessas informações o passo de construção da interação entre os agentes ou RP-NET (próximo passo) pode ser feito paralelamente a esse passo. Finalmente, no Modelo de Descritor do Agente deve conter as principais informações lidas e gravadas por este agente.

A notação utilizada para o Modelo de Descritor do Agente é um retângulo, os itens são separados por linhas, sendo que o nome de cada item deve ser escrito em itálico, seguido por dois pontos para separar da sua especificação. A Figura 6.7 mostra o Modelo de Descritor do Agente com todos os dados a serem preenchidos.

Após a elaboração deste modelo, ele deve ser comparado com o Modelo de Funcionalidades para verificar se as funcionalidades incorporadas a cada agente estão consistentes.

Nome: nome do agente
Descrição: pequena descrição em linguagem natural do agente
Funcionalidades: quais as funcionalidades contidas neste agente
Objetivos: quais os objetivos que o agente deve cumprir
Instâncias: quantidade de agentes deste tipo necessária para o sistema
Interações: nome dos *buffers* com os quais o agente se comunica
Mensagens coletadas: quais as mensagens coletadas por este agente
Mensagens produzidas: quais as mensagens produzidas por este agente
Inicialização do agente: quem ou o que é responsável por iniciar o agente no sistema
Destruição do agente: quem ou o que é responsável por eliminar o agente no sistema
Dados lidos: quais os dados lidos pelo agente
Dados gravados: quais os dados gravados pelo agente

Figura 6.7: Modelo de Descritor do Agente.

c) Identificar as interações entre os agentes

Depois de definidos os agentes existentes no sistema, o próximo passo é identificar detalhadamente quais as interações existentes entre eles. Esse passo pode ser feito concomitantemente ao passo anterior. Conforme apresentado no Capítulo 2, este trabalho propõe um novo modelo de interação do agente, que consiste no mecanismo de busca por mensagens pelos agentes. Adotando essa proposição, os artefatos utilizados pelas metodologias estudadas não abordam adequadamente a interação dos agentes. Portanto, o artefato utilizado pela Metodologia Unificada é o **Diagrama RP-Net**, que focaliza nesse mecanismo de interação. Maiores detalhes sobre a RP-Net podem ser encontrados no Capítulo 5. A Figura 6.8 mostra um exemplo onde a agência 1 e a agência 2 estão interagindo com o *buffer*, efetuando o mecanismo de busca por mensagens no ambiente.

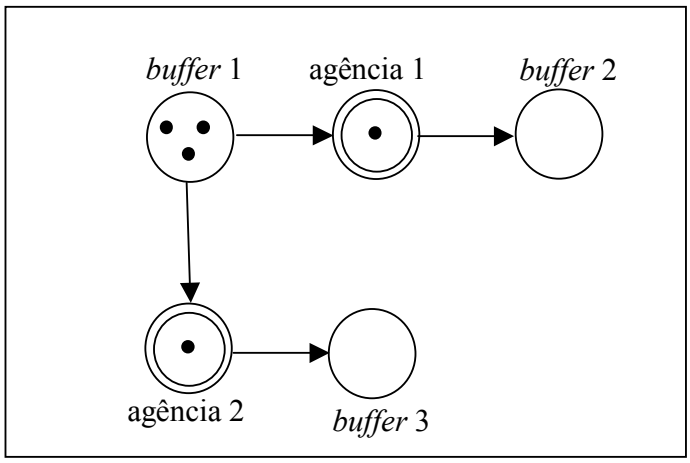


Figura 6.8: Diagrama RP-Net

d) Revisar o sistema

Uma vez identificados os agentes e suas interações, é necessário fazer a revisão do sistema, antes de dar início à fase de projeto detalhado. A Metodologia Unificada propõe a utilização do artefato **Diagrama de Revisão do Sistema**, baseado no Diagrama de Revisão do Sistema da metodologia Prometheus, porém adotando uma notação mais clara e de fácil entendimento para o projetista. A finalidade desse artefato é apresentar uma visão geral do sistema fazendo a consistência entre os dados obtidos neste diagrama com os dados obtidos a partir do Modelo de Funcionalidades, do Modelo Descritor do Agente e do Diagrama da RP-Net. O objetivo de se comparar com o Modelo de Funcionalidades é o de justamente verificar se os objetivos identificados para as funcionalidades estão sendo abordados pelo Modelo de Descritor do Agente. Ao se comparar o Diagrama de Revisão do Sistema com o Modelo de Descritor do Agente é possível checar se todos os agentes identificados anteriormente estão sendo tratados, resultando em uma verificação geral do sistema.

O Diagrama de Revisão do Sistema contém as seguintes informações: nomes dos agentes, acompanhados pelos nomes das funcionalidades, quais mensagens os agentes buscam no ambiente, quais mensagens os agentes produzem no ambiente e os recursos externos utilizados.

A visão do sistema é representada por um grande círculo tracejado. O agente executando uma funcionalidade específica é representado por um retângulo contendo o nome do agente seguido pelo nome da funcionalidade. O recurso externo é representado por um cilindro. A direção das setas distingue a leitura e gravação dos dados, ou seja, se a seta parte da caixa externa e entra até o agente representa uma leitura do banco de dados, se ela sai do agente, representa que o agente está gravando no banco de dados. As mensagens são armazenadas em *buffers* e os caminhos dessas mensagens são representados por setas de direcionamento. Um dos problemas identificados no Diagrama de Revisão do Sistema é a questão da escalabilidade, pois sistemas com muitos agentes podem tornar complexa a interpretação deste diagrama. Esse item será discutido no Capítulo 8.

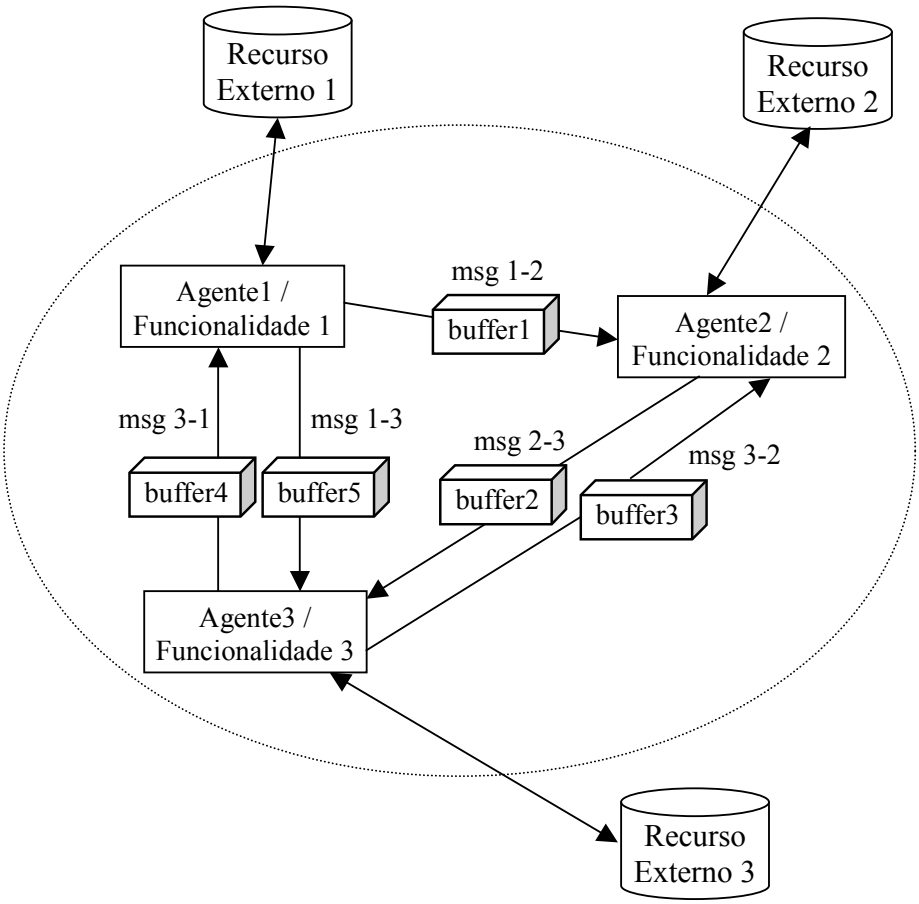


Figura 6.9: Diagrama de Revisão do Sistema

Nesta fase os agentes foram identificados e refinados, o Modelo de Descritor do Agente foi elaborado, as interações dos agentes foram modeladas e finalmente foi elaborado um modelo de revisão do sistema. Esses quatro passos permitem ao projetista ter uma visão tanto da estrutura estática quanto da estrutura dinâmica social dos agentes existentes no sistema. A partir do momento que todos esses dados são revistos pela equipe de engenheiros de software, projetistas e clientes, é possível passar para a fase de projeto detalhado do sistema.

6.4.1.3 Fase de Projeto Detalhado

A fase de projeto detalhado representa a última fase da Metodologia Unificada. Concentra-se na especificação no nível micro do agente, detalhando a arquitetura interna estática e dinâmica de cada agente, em termos de definir a estrutura do agente e como ele realiza suas tarefas.

A fase de projeto detalhado engloba 2 passos:

- Construir a estrutura interna de cada agente;
- Construir a estrutura dinâmica de cada agente.

a) Construir a estrutura interna de cada agente

Após a revisão do sistema multi-agentes ter sido realizada, o foco passa a ser o detalhamento de cada agente. Na fase anterior, o Modelo de Descritor do Agente fornece uma visão geral do agente, identificando suas funcionalidades, quantidade de instâncias de cada agente, quando o agente deve ser iniciado ou destruído. Na fase de Projeto Detalhado, são identificadas quais atividades cada agente deve executar para atingir cada objetivo, representando o processamento interno do agente. O artefato proposto pela Metodologia Unificada é o **Descritor da Estrutura Interna do Agente**, inspirado no Modelo de Funcionalidades também proposto na metodologia.

Primeiramente, deve-se identificar o agente que está sendo detalhado. Em seguida, deve-se relacionar todas as atividades vinculadas a este agente, acompanhado dos mecanismos de contingências relacionadas às atividades, especificando as possíveis causas de falhas da atividade e quais ações (ou reações) o agente deve executar para suprir essa falha.

A notação utilizada para o Descritor da Estrutura Interna do Agente é um retângulo, composto pelos itens separados por linhas. O nome de cada item deve ser escrito em itálico, seguido por dois pontos separando o nome do item e sua descrição. A Figura 6.10 mostra o Descritor da Estrutura Interna do Agente.

Nome: nome do agente
Atividades envolvidas: quais as atividades que o agente possui para atingir seu objetivo
Contingência: *no. atividade*: qual o número da atividade a que esta contingência está relacionada *Causa da falha*: quais as possíveis causas de falha que a atividade pode possuir *Reação*: quais as possíveis reações que o agente pode ter para que sua atividade seja realizada

Figura 6.10: Descritor da Estrutura Interna do Agente

b) Construir a estrutura dinâmica de cada agente

O último passo da Metodologia Unificada consiste em detalhar as atividades identificadas no passo anterior. O artefato utilizado pela Metodologia Unificada é o **Diagrama de Estados**,

baseado no Diagrama de Estados da AUML. O Diagrama de Estados modela a performance ou passos de uma atividade sob o ponto de vista de um agente. Primeiramente deve-se identificar qual o evento que gerou a atividade. A atividade de um agente pode ser gerada por dois tipos de eventos: externo (gerado por outro agente) e interno (gerado pelo próprio agente). Quando o evento é externo a notação utilizada é nome-do-evento(agente-envio, agente-recebimento, nome-da-atividade). Quando o evento é interno utiliza-se: nome-do-evento(nome-do-agente, nome-da-atividade). O estado inicial da atividade é representada por um círculo preenchido. O estado final é representado por um círculo contendo outro círculo preenchido dentro dele. A transição entre um estado e outro da atividade é representada por linhas e os estados das ações correspondem a um retângulo com cantos arredondados. A condição de guarda é representada por um losango. As duas barras horizontais juntas representam barras de sincronização entre fluxos paralelos ou concorrentes, conforme pode ser visualizado na Figura 6.11.

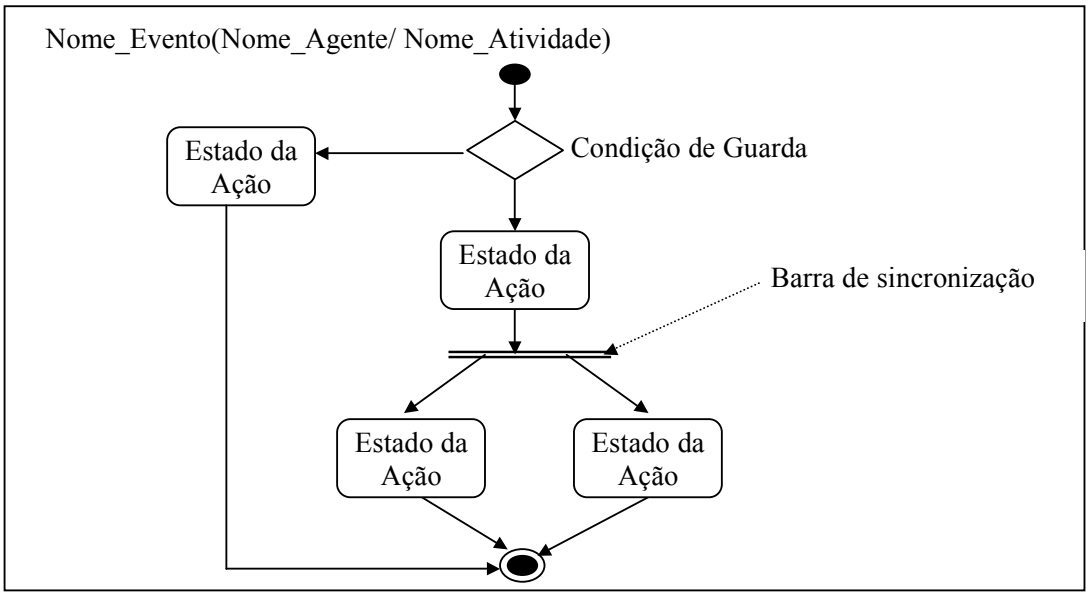


Figura 6.11: Diagrama de Estado

6.5 Resumo

Neste capítulo foram apresentados os critérios utilizados para a seleção dos artefatos que compõem a Metodologia Unificada.

Em seguida, foi apresentada uma avaliação comparativa das metodologias MaSE, Prometheus e Tropos. Essa avaliação foi realizada com base nos critérios e nas modelagens de um estudo de caso utilizando cada uma dessas metodologias.

Finalmente descrevemos detalhadamente a Metodologia Unificada proposta neste trabalho, incluindo suas fases e seus artefatos.

Capítulo 7

Estudo de Caso

7.1 Introdução

Neste capítulo é realizada uma modelagem para a validação da Metodologia Unificada, enfatizando suas fases, suas atividades e seus artefatos. O estudo de caso utilizado para validar a metodologia proposta refere-se a um Sistema Multi-Agentes de Busca de Informações Científicas a partir de um repositório comum.

Várias pesquisas estão sendo desenvolvidas com o objetivo de explorar os benefícios da utilização do paradigma de agentes para a busca de informações [Levy 1994, Maes 1994b, Finin 1998, Freitas 2002]. Com base nesses estudos e no crescente interesse pela utilização de sistemas multi-agentes nos sistemas de busca e recuperação de informação, decidiu-se escolher este tipo de sistema como um caso de estudo para a metodologia proposta.

A Seção 7.2 apresenta o estudo de caso abordado neste trabalho. Na Seção 7.3 o estudo de caso é modelado utilizando a Metodologia Unificada. Na Seção 7.4 é feita uma avaliação da Metodologia Unificada e finalmente na Seção 7.5 é apresentado um resumo do capítulo.

7.2 Descrição do Estudo de Caso

Atualmente, com a popularização e expansão da Internet, o usuário ao fazer uma busca de informação, recebe uma sobrecarga de informações não-pertinentes (ruídos), dificultando a sua pesquisa. Esta sobrecarga de informações e a necessidade de mecanismos eficientes para capturar a semântica do conteúdo das páginas da *web* despertaram grande interesse entre os pesquisadores da teoria de agentes, tornando o paradigma de sistemas multi-agentes uma abordagem eficiente para a busca e recuperação de informação, conforme apresentado na Seção 2.8.2.

Finin e Nicholas [Finin 1998] identificaram como as características de autonomia, cooperação e adaptação do agente estão relacionadas com as principais questões dos sistemas de busca e recuperação de informação, conforme apresentado na Tabela 7.1

Uma vez identificado o potencial do agente no desenvolvimento de Sistema de Busca de Informações, optou-se neste trabalho por adotar como estudo de caso um Sistema Multi-Agentes para colaboração entre grupos de pesquisa. Neste sistema, ilustrado na Figura 7.1, pesquisadores que desejam exercer atividade colaborativa, alimentam um diretório de colaboração em seus próprios computadores, onde armazenam documentos (publicações científicas, artigos,

referências bibliográficas, etc) desenvolvidos por seu grupo, ou cuja temática seja relevante para as atividades do grupo (por exemplo, artigos encontrados na Internet).

Tabela 7.1: Características de agentes e Recuperação de Informação - adaptado de Finin [Finin 1998].

Necessidades dos Sistemas de Busca e Recuperação de Informação	Autonomia	Cooperação	Adaptação
Realimentação Pertinente		√	√
Extração de Informação		√	
Recuperação Efetiva		√	√
Rota e Filtro	√	√	√
Eficiência e Flexibilidade	√	√	√
Recuperação de Informação Distribuída	√	√	

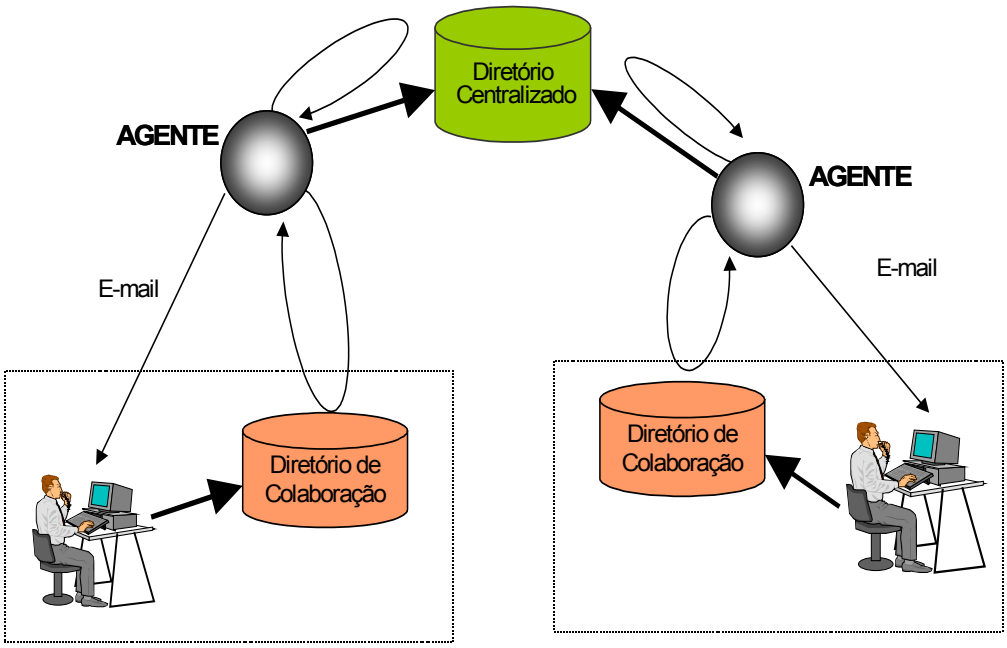


Figura 7.1: Exemplo do agente

Agentes locais vasculham esse repositório e elaboram um perfil dos interesses do pesquisador, que é armazenado como referência. Os agentes coletam esses documentos e os enviam a um diretório centralizado, para compartilhamento com outros pesquisadores. Baseado agora no perfil que montou sobre o pesquisador, eles passam a vasculhar o diretório central, em busca de documentos que possam ser do interesse do pesquisador, promovendo a colaboração. À medida que o usuário acrescenta novos documentos à sua área de trabalho, seu perfil é atualizado automaticamente.

Assim, o sistema multi-agentes abordado deverá realizar as seguintes atividades:

- buscar e recuperar informações (publicações científicas) relevantes existentes nas bases de dados dos membros do grupo;

- classificar as informações armazenadas no repositório comum, obedecendo a critérios pré-estabelecidos;
- filtrar informações de acordo com o perfil do colaborador;
- notificar o colaborador quando da existência de informações do seu interesse;
- analisar e atualizar o perfil de cada colaborador;
- preservar a segurança dos dados.

7.3 Aplicando a Metodologia Unificada

A seguir é apresentada a modelagem do Sistema Multi-Agentes para Busca de Informações Científicas, denominado de SISBIC, detalhando todas as fases, as atividades e os artefatos que compõem a Metodologia Unificada.

7.3.1 Fase de Análise

Na fase de análise são identificados e estruturados os objetivos do sistema, bem como as suas funcionalidades. A Tabela 7.2 mostra a relação entre as atividades e os artefatos de cada atividade propostos pela Metodologia Unificada.

Tabela 7.2: Relação de atividades e artefatos da fase de análise propostos pela Metodologia Unificada.

Atividades	Artefatos
Identificação dos objetivos	Diagrama de ator Diagrama de objetivos
Estruturar os objetivos	Diagrama hierárquico de objetivos
Identificar as Funcionalidades	Modelo de funcionalidades

a) Identificar os objetivos

Para a identificação dos objetivos, primeiramente devem ser identificados os principais atores do sistema. A lista dos atores relevantes para o Sistema Multi-Agentes para Busca de Informações Científicas, é composta por:

- DCA (Departamento de Engenharia de Computação e Automação Industrial): seu interesse é aumentar o compartilhamento de informações acadêmicas entre seus pesquisadores, denominados de colaboradores.
- Colaboradores: são os pesquisadores que desejam obter informações bibliográficas compartilhadas de fácil acesso.

A Figura 7.2 mostra esses atores e seus principais objetivos. Em particular, "colaborador" está associado a um único objetivo relevante: *obter informações atualizadas*, enquanto "DCA"

está associado ao objetivo: *aumentar o compartilhamento de informações*. Os objetivos leves são distinguidos dos objetivos pois estão relacionados aos aspectos qualitativos do sistema. O diagrama inclui um objetivo de dependência onde "colaborador" depende de "DCA" para atingir o objetivo leve: *informações atualizadas*.

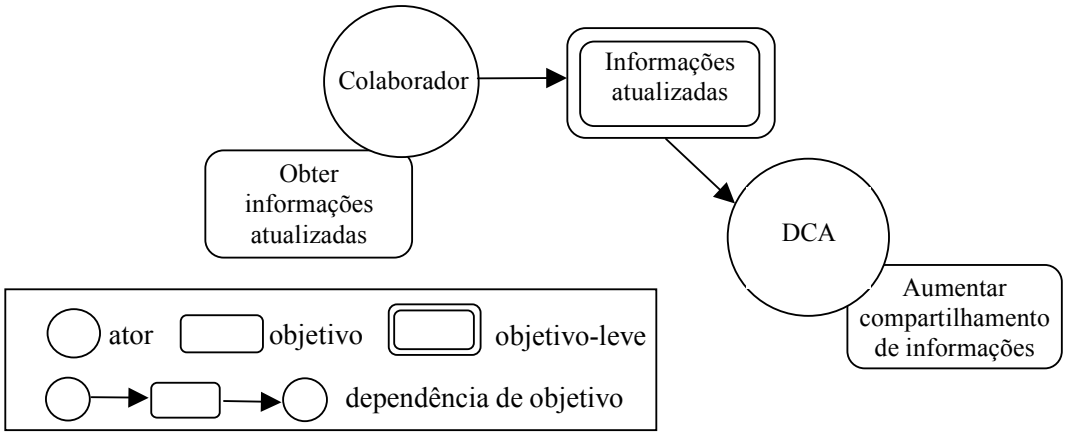


Figura 7.2: Os atores do projeto SISBIC

Após serem identificados os principais objetivos do sistema, o próximo passo é refinar o processo de análise para o levantamento dos demais objetivos, sob o ponto de vista de cada ator do sistema, conforme mostrado na Figura 7.3.

Para o ator "colaborador", o objetivo *obter informações atualizadas* é composto por *obter informações do repositório particular* e *obter informações do repositório compartilhado*. Além do objetivo de ligação entre os atores já identificados, o "colaborador" precisa que o "DCA" deixe o *sistema SISBIC disponível* e que este *sistema seja usável*.

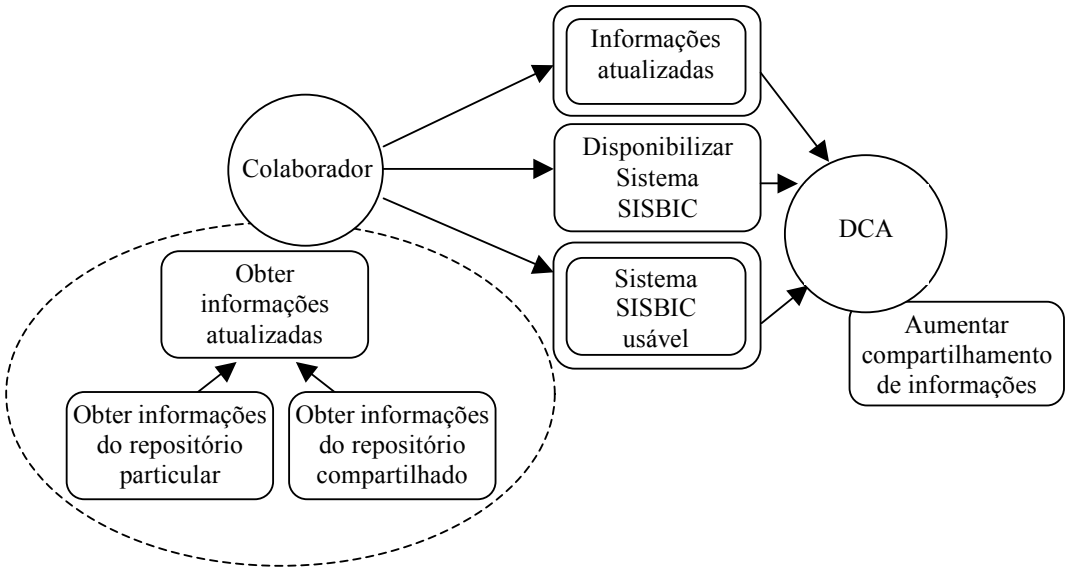


Figura 7.3: Diagrama de Ator da Metodologia Unificada.

O mesmo processo de refinamento deve ser feito para o ator "DCA". Na Figura 7.4, o objetivo *aumentar o compartilhamento de informação* está relacionado com o objetivo *fornecer um serviço de informações compartilhadas* e ao objetivo-leve *informações atualizadas*. O objetivo *construir o sistema SISBIC* está relacionado ao objetivo *fornecer serviço de informações compartilhadas*, ao objetivo *disponibilizar o sistema SISBIC* e ao objetivo-leve *sistema SISBIC usável*.

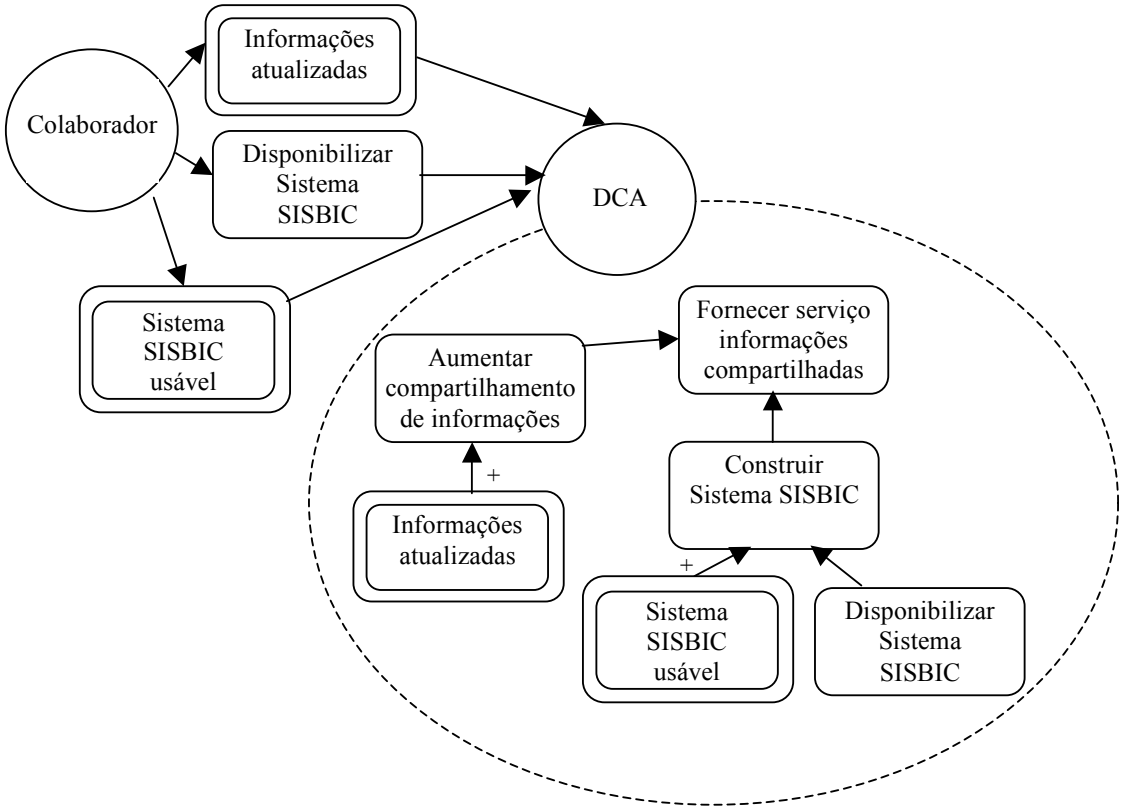


Figura 7.4: Diagrama de Ator da Metodologia Unificada.

No último passo da atividade de identificação de objetivos, o Sistema SISBIC é introduzido com um ator, modelado pelo Diagrama de Objetivos, conforme Figura 7.5. DCA depende do SISBIC para *fornecer serviço de informações compartilhadas*, que é o objetivo que contribui para o objetivo principal do DCA que é aumentar o *compartilhamento de informações*. O SISBIC deve contribuir para o objetivo-leve *sistema SISBIC usável* e *informações atualizadas*. Para atender ao objetivo de *fornecer informações compartilhadas*, foram identificados os seguintes objetivos: *pesquisar informações na área dos colaboradores*, *manipular as referências bibliográficas*, *controlar os acessos dos colaboradores* e *manter o colaborador informado sobre as atualizações no repositório central*, além do objetivo leve *sistema SISBIC usável*, o qual deve atender os seguintes objetivos leves: *sistema amigável e sistema disponível*.

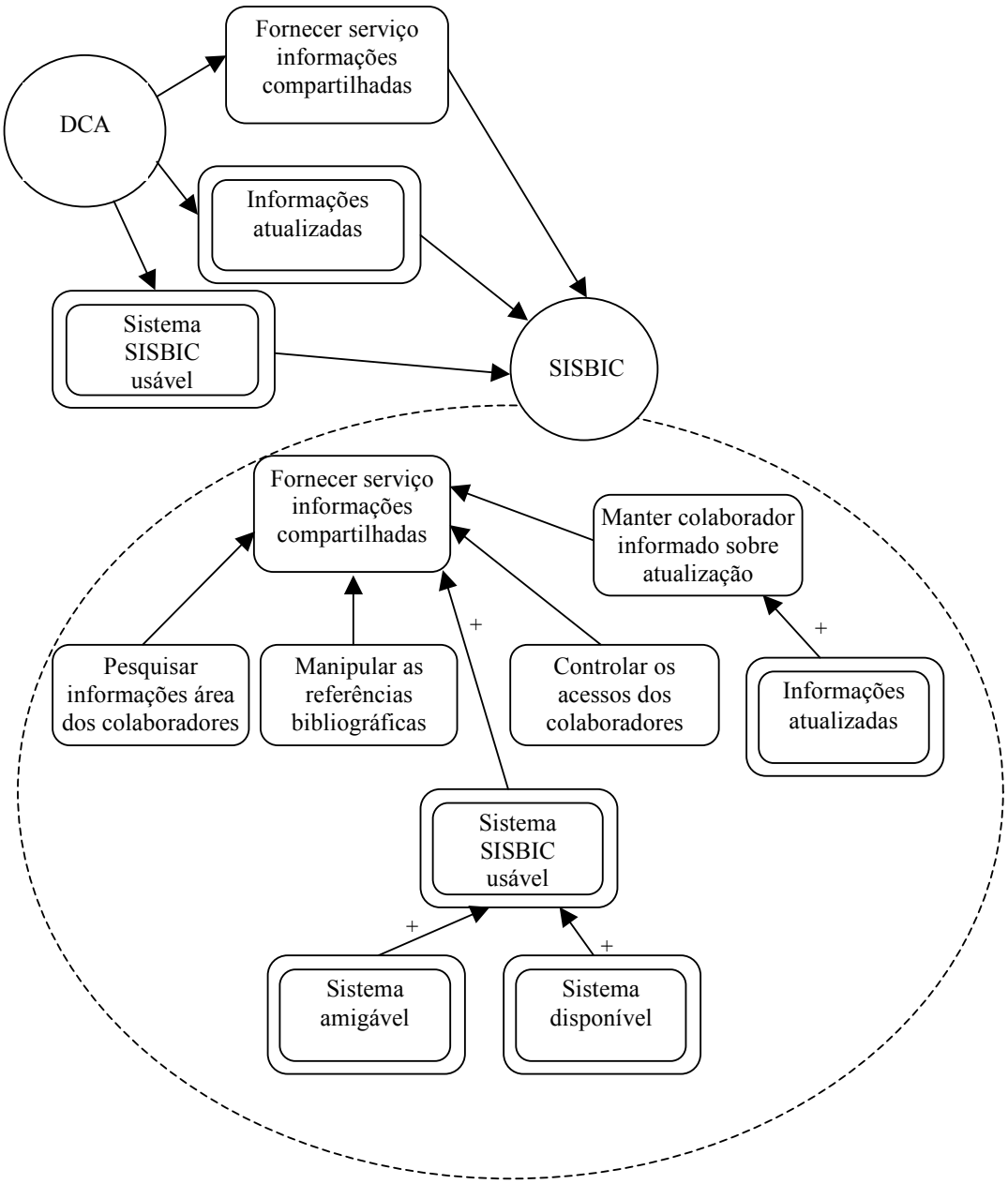


Figura 7.5: Diagrama de Objetivos do Sistema SISBIC.

b) Estruturar os objetivos

Após a construção dos diagramas de objetivos, onde os objetivos do SISBIC foram analisados e modelados de maneira incremental, o próximo passo proposto pela metodologia é organizar e refinar esses objetivos pelo Diagrama Hierárquico de Objetivos, conforme Figura 7.6.

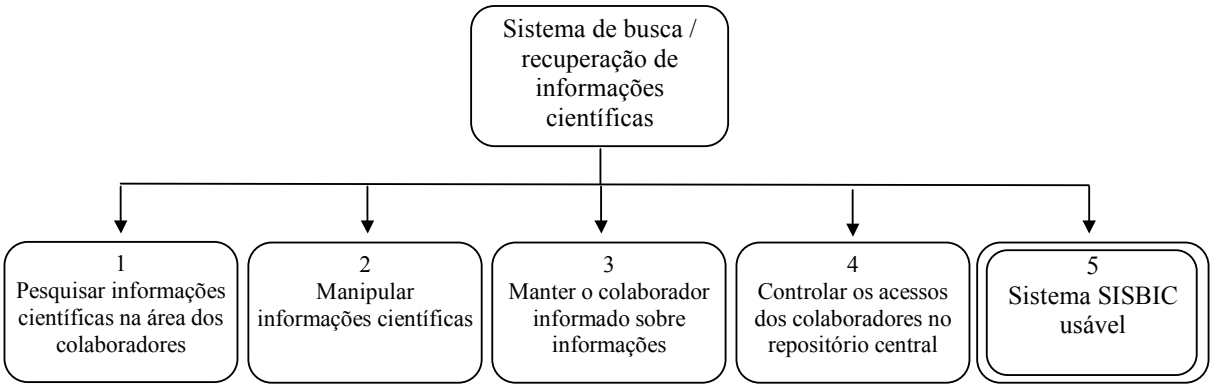


Figura 7.6: Diagrama Hierárquico de Objetivos da Metodologia Unificada.

Em seguida, cada um desses objetivos é refinado e organizado hierarquicamente. Para facilitar o entendimento, eles foram modelados em diagramas separados, conforme podem ser observados nas Figuras 7.7 a 7.10.

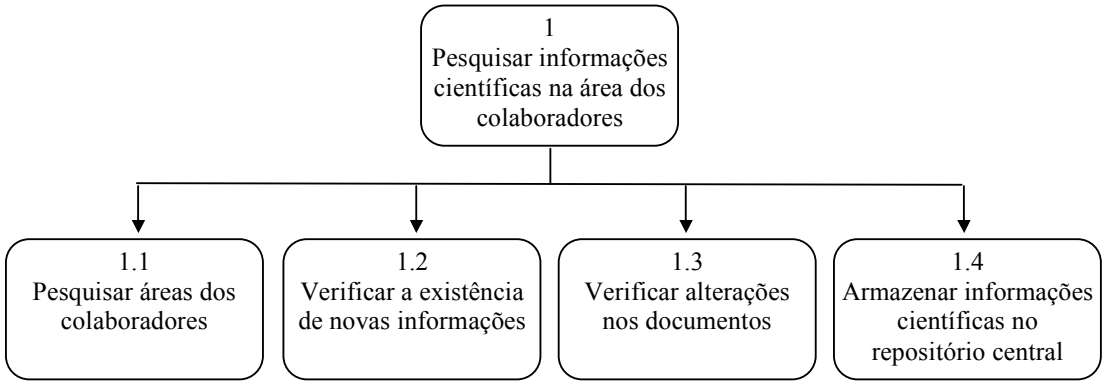


Figura 7.7: Diagrama Hierárquico de Objetivos, referente ao objetivo pesquisar informações científicas na área dos colaboradores.

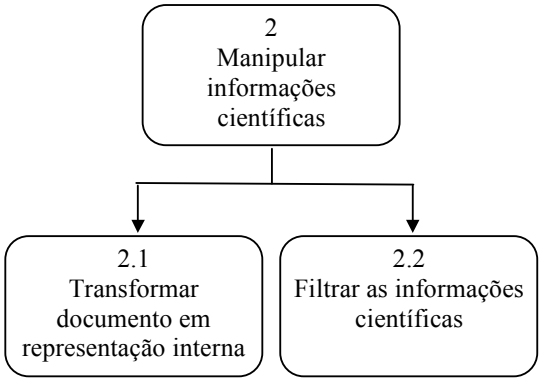


Figura 7.8: Diagrama Hierárquico de Objetivos, referente ao objetivo manipular as informações científicas.

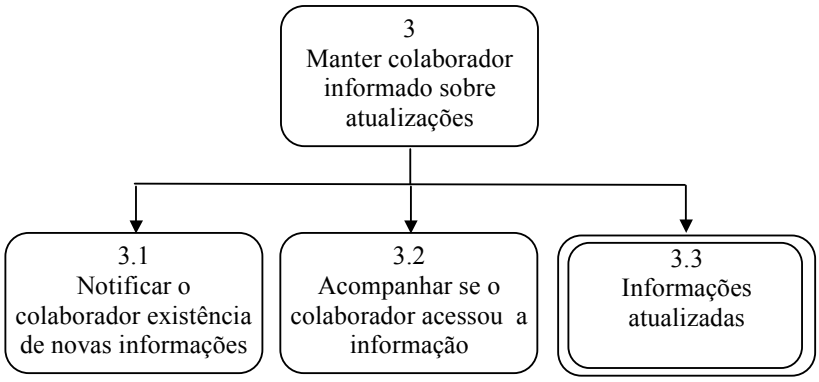


Figura 7.9: Diagrama Hierárquico de Objetivos, referente ao objetivo manter o colaborador informado sobre atualizações.

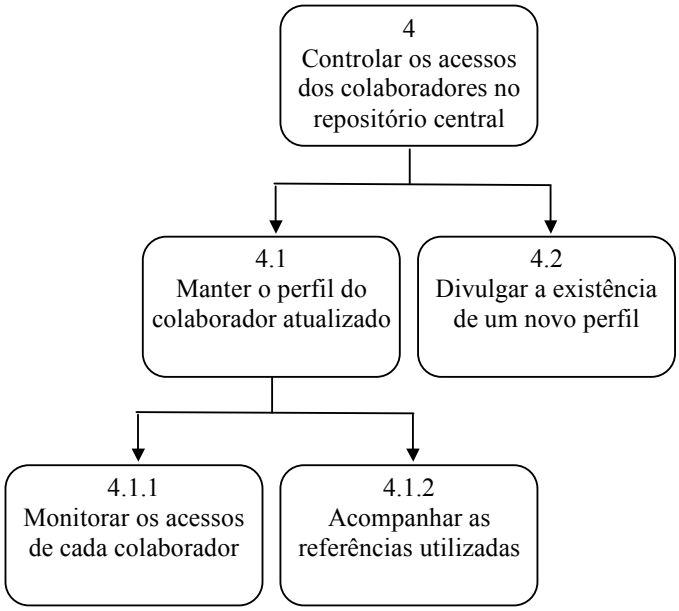


Figura 7.10: Diagrama Hierárquico de Objetivos, referente ao objetivo controlar os acessos dos colaboradores no repositório central.

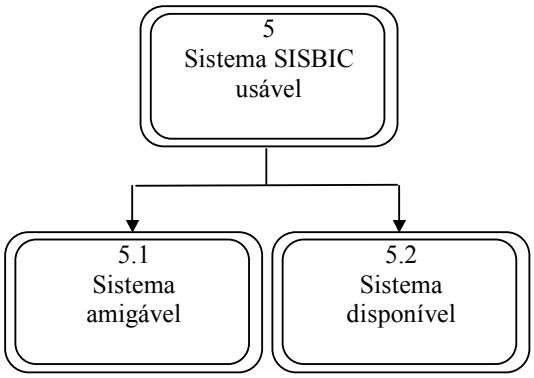


Figura 7.11: Diagrama Hierárquico de Objetivos, referente ao objetivo leve sistema SISBIC usável.

c) Identificar as funcionalidades

Uma vez identificados e refinados os objetivos do SISBIC, o próximo passo da Metodologia Unificada é identificar as funcionalidades do sistema. Primeiramente, é necessário associar cada funcionalidade a um ou mais objetivos do sistema, dependendo do caso. Em seguida, as informações sobre cada uma dessas funcionalidades são documentadas no **Modelo de Funcionalidades**. As Figuras 7.12 a 7.16 modelam as funcionalidades identificadas no SISBIC.

Nome: Descobridor
Descrição: Pesquisar informações científicas nas áreas dos colaboradores
Objetivos: Pesquisar áreas dos colaboradores Verificar a existência de novas informações científicas Verificar as alterações nos documentos armazenados no repositório central Armazenar as informações científicas, coletadas das áreas dos colaboradores, no repositório central
Percepções: Usuário grava novas informações científicas no seu repositório
Ações: Coletar as informações científicas no repositório de cada colaborador Armazenar as informações científicas coletadas no repositório central
Mensagens Coletadas: InformaPerfil_Descobridor
Mensagens Enviadas: InformaDescobridor_Classificador
Buffers de acesso: buffer_Monitor_Perfil_Descobridor, buffer_Descobridor_Classificador
Dados lidos: referencias_bibli_colaborador_db
Dados gerados: referencias_bibli_db

Figura 7.12: Modelo de Funcionalidades – Descobridor

Nome: Classificador
Descrição: Classifica as informações científicas coletadas a partir do repositório de cada colaborador.
Objetivos: Manipular as informações científicas armazenadas no repositório central Transformar o documento em representação interna, criando uma classificação por categoria (área, autor, palavra-chave)
Percepções: Existência de novas informações científicas no repositório central
Ações: Classificar as informações científicas do repositório central de acordo com os critérios pré-estabelecidos
Mensagens Coletadas: InformaDescobridor_Classificador
Mensagens Enviadas:

InformaClassificador_Filtrador
Buffers de acesso: buffer_Descobridor_Classificador, buffer_Classificador_Filtrador
Dados lidos: referencias_bibli_db
Dados gerados: referencias_bibli_db

Figura 7.13: Modelo de Funcionalidades – Classificador

Nome: Filtrador
Descrição: Filtra as informações científicas do repositório central de acordo com o perfil do colaborador
Objetivos: Manipular as informações científicas armazenadas no repositório central Filtrar as informações científicas de acordo com o perfil de cada colaborador
Percepções: Existência de novas referências no repositório central Existência de perfil novo ou atualizado
Ações: Ler as informações científicas do repositório central Filtrar as informações lidas de acordo com o perfil de do colaborador
Mensagens Coletadas: InformaClassificador_Filtrador InformaPerfil_Filtrador
Mensagens Enviadas: InformaFiltrador_Notificador
Buffers de acesso: buffer_Classificador_Filtrador, buffer_Monitor_Perfil_Filtrador, buffer_Filtrador_Notificador
Dados lidos: referencias_bibli_db
Dados gerados: referencias_bibli_perfil_db

Figura 7.14: Modelo de Funcionalidades – Filtrador

Nome: Notificador
Descrição: Notifica ao colaborador a existência de uma nova informação científica
Objetivos: Manter o colaborador informado sobre atualização Notificar ao colaborador a existência de novas referências Acompanhar se o colaborador acessou a informação Informações atualizadas
Percepções: Existência de nova informação científica armazenada no repositório central que atende ao perfil do colaborador
Ações: Informar ao colaborador que existe uma informação científica que atende seu perfil

Mensagens Coletadas: InformaFiltrador_Notificador
Mensagens Enviadas: InformaNotificador_Perfil InformaColaborador
Buffers de acesso: buffer_Filtrador_Notificador, buffer_Notificador_Monitor_Perfil, Colaborador
Dados lidos: referencias_bibli_perfil_db
Dados gerados: notificacao_db

Figura 7.15: Modelo de Funcionalidades – Notificador

Nome: Monitor_Perfil
Descrição: Avalia e monitora o perfil de cada colaborador
Objetivos: Controlar os acessos dos colaboradores no repositório central Manter o perfil do colaborador atualizado Monitorar os acessos de cada perfil Acompanhar as referências utilizadas por cada colaborador Divulgar a existência de um novo colaborador Sistema SISBIC usável Sistema amigável Sistema disponível
Percepções: Existência de um novo colaborador no sistema Alteração do perfil de um colaborador Notificação sobre nova informação enviada para o colaborador
Ações: Criar um perfil para o colaborador Atualizar o perfil de cada colaborador de acordo com os acessos Verificar os acessos feitos por cada colaborador
Mensagens Coletadas: LoginColaborador (mensagem) InformaNotificador_Perfil
Mensagens Enviadas: InformaPerfil_Descobridor InformaPerfil_Filtrador
Buffers de acesso: buffer_Monitor_Perfil_Descobridor, buffer_Notificador_Monitor_Perfil, buffer_Monitor_Perfil_Filtrador
Dados lidos: perfil_db, notificacao_db
Dados gerados: perfil_db

Figura 7.16: Modelo de Funcionalidades – Monitor_Perfil

7.3.2 Fase de Projeto Arquitetural

Nesta fase, o projetista irá construir uma arquitetura do sistema em desenvolvimento, obtendo a estrutura social estática e dinâmica dos agentes. As atividades e os artefatos gerados nesta fase são ilustrados na Tabela 7.3.

Tabela 7.3: Relação de atividades e artefatos da fase de projeto arquitetural proposta pela Metodologia Unificada.

Atividades	Artefatos
Identificar os tipos de agentes	Diagrama de Agrupamento de Funcionalidades
Refinar os agentes	Modelo de Descritor do Agente
Identificar as interações entre os agentes	Diagrama RP-Net
Revisar o sistema	Diagrama de Revisão do Sistema

a) Identificar os tipos de agentes

O processo de identificação dos tipos de agentes é elaborado a partir do Modelo de Funcionalidades construído no passo anterior. O artefato proposto pela Metodologia Unificada é o Diagrama de Agrupamento de Funcionalidades. A Figura 7.17 mostra este artefato referente ao estudo de caso do SISBIC, com funcionalidades agrupadas segundo o critério de utilização de banco de dados. Ou seja, as funcionalidades que acessam praticamente a mesma base de dados foram agrupadas formando os tipos de agentes necessários para o sistema. O agente A_Descobridor, cujo agrupamento é representado por uma linha simples, inclui a funcionalidade "Descobridor" e os bancos de dados "refer_bibl_colaborador" e "refer_bibl". O agente A_Classificador, representado por uma linha em negrito, inclui a funcionalidade "Classificador" e o banco de dados "refer_bibl". O agente A_Notificador, representado por linhas duplas, inclui a funcionalidade "Notificador" e os bancos de dados "refer_bibl_perfil" e "notificação". O agente A_Filtrador, representado por uma linha tracejada, inclui a funcionalidade "Filtrador" e os bancos de dados "refer_bibl" e "refer_bibl_perfil". O agente A_Monitor_Perfil, representado por uma linha em negrito, inclui a funcionalidade "Monitor_Perfil" e os bancos de dados "notificação" e "perfil".

b) Refinar os agentes

O objetivo desse passo é documentar, por meio do Descritor do Agente, as informações de alto nível dos agentes. De acordo com o Diagrama de Agrupamento de Funcionalidades, os agentes identificados para o SISBIC são: A_Descobridor, A_Notificador, A_Classificador, A_Filtrador e A_Monitor_Perfil. As Figuras de 7.18 a 7.22, mostram o Descritor de Agente referente a cada um desses agentes.

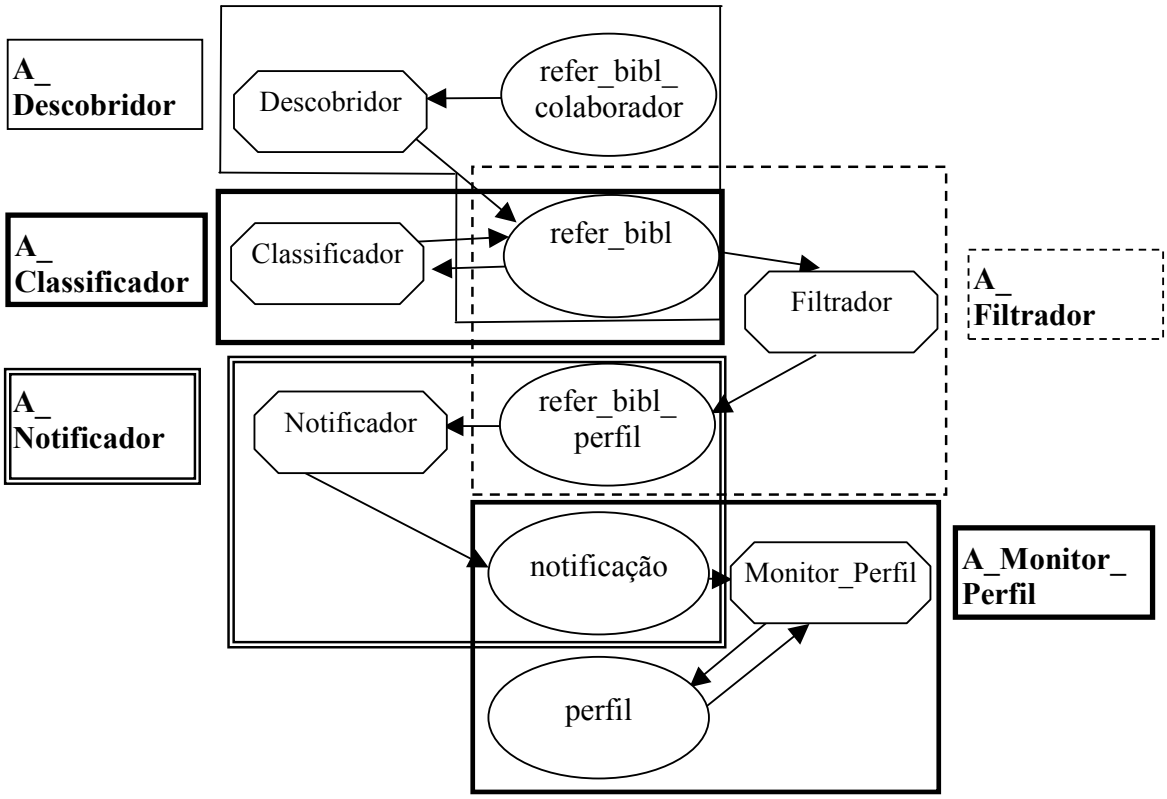


Figura 7.17: Diagrama de Agrupamento de Funcionalidades

Nome: A_Descobridor
Descrição: Busca referência bibliográfica da área de cada colaborador
Funcionalidades: Descobridor
Objetivos: Pesquisar áreas dos colaboradores Verificar a existência de novas informações científicas Verificar as alterações nos documentos armazenados no repositório central Armazenar as informações científicas, coletadas das áreas dos colaboradores, no repositório central
Instâncias: *n*, onde *n* é igual ao número de colaboradores.
Interações: buffer_Monitor_Perfil_Descobridor (buffer 6), buffer_Descobridor_Classificador (buffer 1)
Mensagens coletadas: InformaPerfil-Desc
Mensagens produzidas: InformaDescobridor-Class
Inicialização do agente: colaborador
Destruição do agente: colaborador
Dados lidos: referencias_bibli_colaborador_db
Dados gravados: referencias_bibli_db

Figura 7.18: Descritor do Agente A_Descobridor

Nome: A_Classificador
Descrição: Classifica as informações científicas coletadas a partir do repositório de cada colaborador
Funcionalidades: Classificador
Objetivos: Manipular as informações científicas armazenadas no repositório central Transformar o documento em representação interna, criando uma classificação por categoria (área, autor, palavra-chave)
Instâncias: 1
Interações: buffer_Descobridor_Classificador (buffer 1), buffer_Classificador_Notificador (buffer 2)
Mensagens coletadas: InformaDescobridor-Class
Mensagens produzidas: InformaClassificador-Filtr
Inicialização do agente: administrador do sistema
Destruição do agente: administrador do sistema
Dados lidos: referencias_bibli_db
Dados gravados: referencias_bibli_db

Figura 7.19: Descritor do Agente A_Classificador

Nome: A_Filtrador
Descrição: Filtra as referências do repositório central de acordo com o perfil do colaborador
Funcionalidades: Filtrador
Objetivos: Manipular as informações científicas armazenadas no repositório central Filtrar as informações científicas de acordo com o perfil de cada colaborador
Instâncias: *n*, onde *n* é o número de colaboradores
Interações: buffer_Classificador_Filtrador (buffer 2), buffer_Monitor_Perfil_Filtrador (buffer 5), buffer_Filtrador_Notificador (buffer 3)
Mensagens coletadas: InformaClassificador-Filtr, InformaPerfil-Filtr
Mensagens produzidas: InformaFiltrador-Notif
Inicialização do agente: colaborador
Destruição do agente: colaborador
Dados lidos: referencias_bibli_db
Dados gravados: referencias_bibl_perfil_db

Figura 7.20: Descritor do Agente A_Filtrador

Nome: A_Notificador
Descrição: Notifica ao colaborador a existência de uma nova informação científica
Funcionalidades: Notificador
Objetivos:

Manter o colaborador informado sobre atualização Notificar ao colaborador a existência de novas referências Acompanhar se o colaborador acessou a informação Informações atualizadas
Instâncias: *n*, onde *n* é o número de colaboradores
Interações: buffer_Filtrador_Notificador (buffer 3), buffer_Notificador_Monitor_Perfil (buffer 4), Colaborador
Mensagens coletadas: InformaFiltrador-Notif
Mensagens produzidas: InformaNotificador-Perf, InformaColaborador
Inicialização do agente: administrador do sistema
Destruição do agente: administrador do sistema
Dados lidos: referencias_bibli_perfil_db
Dados gravados: notificacao_db

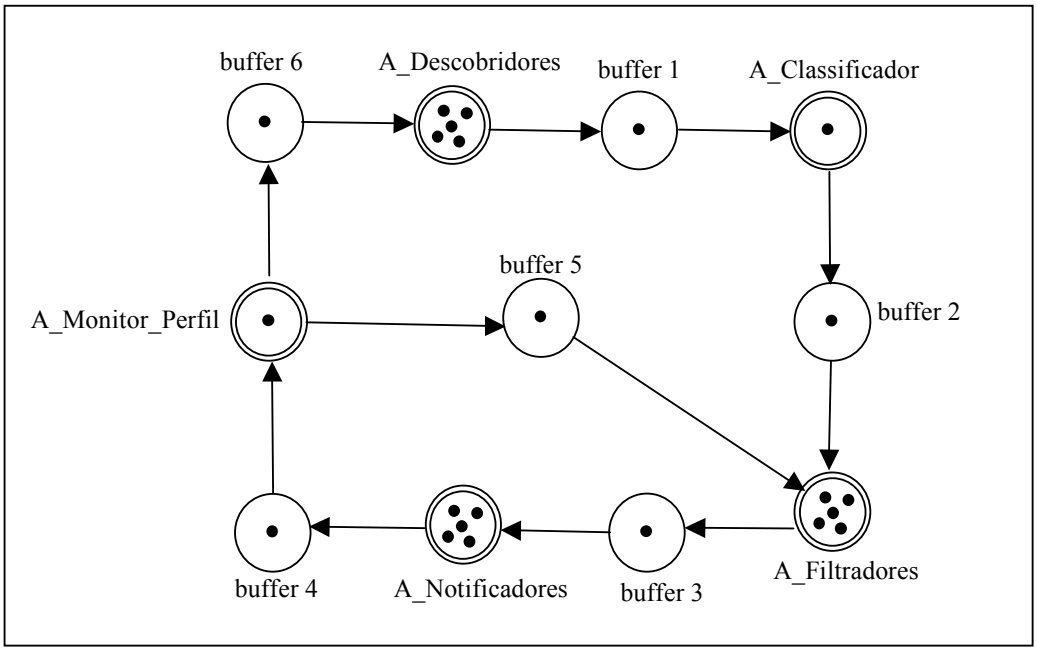
Figura 7.21: Descritor do Agente A_Notificador

Nome: A_Monitor_Perfil
Descrição: Avalia e monitora o perfil de cada colaborador
Funcionalidades: Monitor_Perfil
Objetivos: Controlar os acessos dos colaboradores no repositório central Manter o perfil do colaborador atualizado Monitorar os acessos de cada perfil Acompanhar as referências utilizadas por cada perfil Divulgar a existência de um novo perfil Sistema SISBIC usável Sistema amigável Sistema disponível
Instâncias: 1
Interações: buffer_Monitor_Perfil_Descobridor (buffer 6), buffer_Notificador_Monitor_Perfil (buffer 4), buffer_Monitor_Perfil_Filtrador (buffer 5)
Mensagens coletadas: InformaNotificador-Perf, LoginColaborador
Mensagens produzidas: InformaPerfil-Desc, InformaPerfil-Filtr
Inicialização do agente: colaborador
Destruição do agente: colaborador
Dados lidos: notificacao_db, perfil_db
Dados gravados: perfil_db

Figura 7.22: Descritor do Agente A_Monitor_Perfil

c) Identificar as interações entre os agentes

O produto gerado nesta fase é o Diagrama RP-NET, o qual representa graficamente os aspectos dinâmicos do sistema, enfatizando o mecanismo de busca por mensagens dos agentes. Este passo pode ser feito em paralelo com o passo anterior. No estudo de caso realizado com o SISBIC, foi obtido o Diagrama RP-Net apresentado na Figura 7.23.



Buffer	Descrição
buffer 1	Buffer_Descobridor_Classificador
buffer 2	Buffer_Classificador_Filtrador
buffer 3	Buffer_Filtrador_Notificador
buffer 4	Buffer_Notificador_Monitor_Perfil
buffer 5	Buffer_Monitor_Perfil_Filtrador
buffer 6	Buffer_Monitor_Perfil_Descobridor

Figura 7.23: Diagrama RP-NET do SISBIC

d) Revisar o Sistema

O último passo da fase de projeto arquitetural consiste na elaboração do Diagrama de Revisão do Sistema, onde é possível ter uma visão geral do sistema e fazer uma comparação entre o Modelo de Funcionalidades, o Modelo de Descritor do Agente e o Diagrama RP-Net, para garantir a consistência de todos os dados obtidos até esta etapa do processo de desenvolvimento do sistema. O Diagrama de Revisão do Sistema referente ao SISBIC é apresentado na Figura 7.24.

No Diagrama de Revisão do Sistema é possível se ter uma visão completa do sistema multi-agentes, incluindo seus agentes, mensagens e dados externos utilizados. Após a análise desse diagrama, pode ser necessário ter que refazer os passos anteriores para o refinamento dos diagramas e somente então poderá ser iniciada a próxima fase da metodologia.

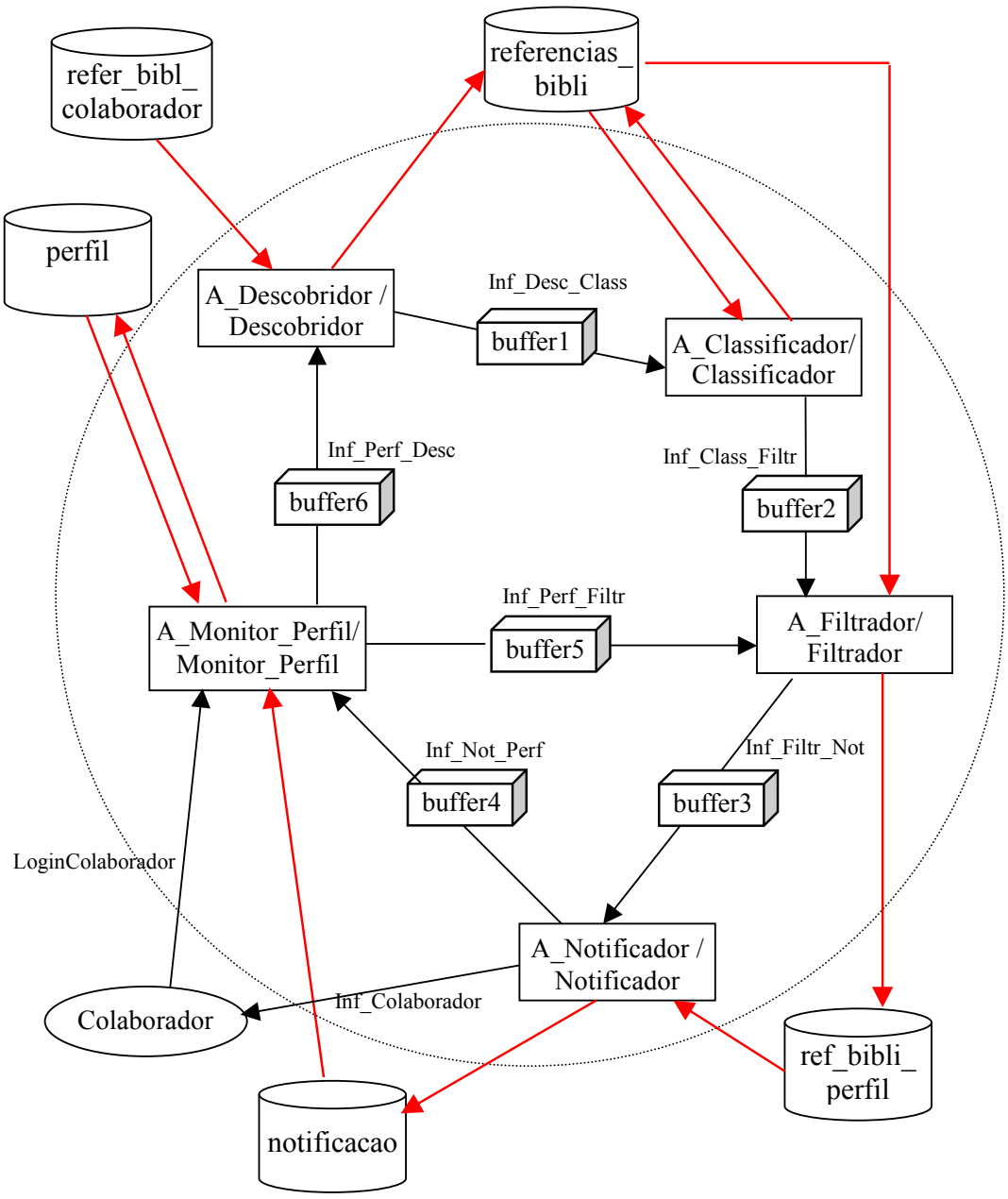


Figura 7.24: Diagrama de Revisão do Sistema SISBIC

7.3.3 Fase de Projeto Detalhado

A última fase da Metodologia Unificada consiste na especificação da arquitetura interna estática e dinâmica de cada agente, ou seja, na elaboração do nível micro do agente. As atividades e os artefatos gerados nesta fase são ilustrados na Tabela 7.4.

Tabela 7.4: Relação de atividades e artefatos da fase de projeto detalhado proposta pela Metodologia Unificada.

Atividades	Artefatos
Construir a estrutura interna de cada agente	Descritor da Estrutura Interna do Agente
Construir a estrutura dinâmica de cada agente	Diagrama de Estados

a) Construir a estrutura estática de cada agente

Essa etapa detalha cada agente, identificando as atividades necessárias para que seus objetivos sejam atingidos. As Figuras de 7.25 a 7.30 apresentam os Descritores da Estrutura Interna dos Agentes identificados no SISBIC.

Nome: A_Descobridor
Atividades envolvidas: 01 – Ler o repositório específico de cada colaborador 02 – Gravar no repositório central as informações científicas lidas do repositório específico de cada colaborador 03 – Verificar se a referência já não foi gravada no repositório central
Contingência: *no. atividade*: 01 *causa da falha*: colaborador não está logado / acessível *reação*: tentar novamente em intervalos de tempo
Contingência: *no. atividade*: 02 e 03 *causa da falha*: repositório central não está disponível (desligado / sobrecarregado) *reação*: informar colaborador / administrador sobre indisponibilidade do servidor e tentar novamente

Figura 7.25: Descritor da Estrutura Interna do Agente A_Descobridor

Nome: A_Classificador
Atividades envolvidas: 01 – Ler as novas referências bibliográficas armazenadas no repositório central 02 – Classificar as referências bibliográficas do repositório central por categoria (área, autor, palavra-chave)
Contingência: *no. atividade*: 01 *causa da falha*: repositório central não está disponível (não acessível ou sem permissão) *reação*: informar administrador do sistema
Contingência: *no. atividade*: 02 *causa da falha*: não se aplica *reação*: não se aplica

Figura 7.26: Descritor da Estrutura Interna do Agente A_Classificador

Nome: A_Filtrador
Atividades envolvidas: 01 – Filtrar as informações científicas do repositório central de acordo com o perfil do colaborador 02 – Verificar se o colaborador não possui a informação filtrada 03 – Disponibilizar as informações científicas filtradas
Contingência: *no. atividade*: 01 e 03 *causa da falha*: repositório central está vazio *reação*: tentar novamente em intervalos de tempo
Contingência: *no. atividade*: 02 *causa da falha*: colaborador perdeu a conexão com o sistema *reação*: cancelar a operação

Figura 7.27: Descritor da Estrutura Interna do Agente A_Filtrador

Nome: A_Notificador
Atividades envolvidas: 01 – Receber a mensagem sobre a existência de uma nova informação científica no repositório central que atende a um determinado perfil 02 – Notificar o colaborador a existência de uma nova informação científica 03 – Assegurar que o colaborador recebeu a notificação
Contingência: *no. atividade*: 02 e 03 *causa da falha*: colaborador não está logado no sistema *reação*: cancelar a operação ou tentar novamente quando o colaborador logar no sistema

Figura 7.28: Descritor da Estrutura Interna do Agente A_Notificador

Nome: A_Monitor_Perfil
Atividades envolvidas: 01 – Criar perfil inicial do colaborador 02 – Informar ao demais agentes do sistema a existência de um novo perfil 03 – Monitorar as informações científicas acessadas por cada colaborador 04 – Atualizar o perfil de cada colaborador
Contingência: *no. atividade*: 01, 03 e 04 *causa da falha*: colaborador perdeu a conexão com o sistema *reação*: cancelar a operação ou tentar novamente quando o colaborador logar no sistema
Contingência: *no. atividade*: 02 *causa da falha*: sistema inoperante *reação*: enviar aviso para o administrador do sistema

Figura 7.29: Descritor da Estrutura Interna do Agente A_Monitor_Perfil

b) Construir a estrutura dinâmica de cada agente

O objetivo desse passo é detalhar as atividades identificadas no Descritor da Estrutura Interna do Agente, identificando os aspectos dinâmicos do agente, como os estados das ações e a transição de um estado a outro. O artefato utilizado pela Metodologia Unificada é baseado no Diagrama de Estados da AUML. Na Figura 7.30 é apresentado o Diagrama de Estados, referente à atividade "Filtrar as informações científicas do repositório central de acordo com o perfil do colaborador" do sistema SISBIC.

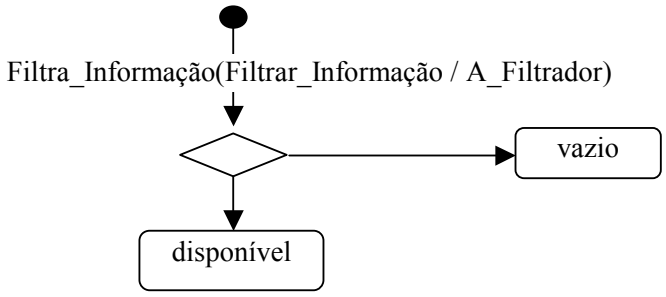


Figura 7.30: Diagrama de Estados da atividade "Filtrar as informações científicas do repositório central de acordo com o perfil do colaborador".

7.4 Avaliação da Metodologia Unificada

A Metodologia Unificada apresentada neste trabalho se mostrou adequada ao desenvolvimento de sistemas orientados a agentes, pelas seguintes razões:

- aborda as características inerentes dos agentes como autonomia, reatividade, pró-atividade e habilidade social;
- aborda a fase de análise, incluindo o levantamento de requisitos do sistema, a fase de projeto arquitetural e a fase de projeto detalhado;
- melhoria no levantamento dos requisitos dos sistemas e na estruturação dos objetivos encontrados, uma vez que a Metodologia Unificada auxilia o projetista a identificar os objetivos do sistema e a estruturar sua sintaxe;
- melhoria na notação gráfica utilizada nos artefatos das metodologias estudadas para comporem a Metodologia Unificada, simplificando o seu entendimento e sua utilização;
- o artefato utilizado para representar a interação do agente é mais apropriado ao mecanismo de busca de informações dos agentes do que os artefatos utilizados pelas metodologias estudadas, conforme discutido no Capítulo 5;
- fornece artefatos para documentação em todas as fases, pois além dos diagramas gerados pela Metodologia Unificada, também existem os Modelos de Descritores que permitem essa documentação;
- o Diagrama de Agrupamento de Funcionalidades representa um artefato específico para converter funcionalidades em agentes, incluindo critérios específicos para a identificação dos agentes necessários ao sistema;

- possibilita a rastreabilidade entre os artefatos propostos pela Metodologia Unificada, incluindo o Diagrama de Revisão do Sistema, o qual permite avaliar se os principais conceitos do sistema multi-agentes estão sendo modelados corretamente.

Assim, a Metodologia Unificada proposta neste trabalho, pretende contribuir com a Engenharia de Software Orientada a Agentes, facilitando a modelagem de sistemas multi-agentes, por meio de um conjunto de fases, atividades e artefatos.

7.5 Resumo

Neste capítulo foi apresentado o estudo de caso utilizado para a validação da Metodologia Unificada. Em seguida, este estudo de caso foi modelado, detalhando as fases, as atividades e os artefatos que compõem a metodologia proposta.

Após a modelagem, foi feita uma avaliação da Metodologia Unificada, ressaltando seus principais benefícios.

Capítulo 8

Conclusão e Trabalhos Futuros

8.1 Contribuições

As abordagens orientadas a agentes representam um paradigma emergente na Engenharia de Software, principalmente por existir uma grande demanda pela utilização de agentes nas aplicações industriais. Dessa maneira, é fundamental a existência de metodologias padronizadas, baseadas no paradigma de agentes, que auxiliem na construção sistemática de tais aplicações. Com o objetivo de atender essa necessidade, a principal contribuição deste trabalho é a proposta de uma metodologia unificada para o desenvolvimento de sistemas orientados a agentes.

De uma maneira mais detalhada, considera-se que as contribuições deste trabalho são as seguintes:

- Revisão do conceito de agente como um modelo de interação distinto.
- Proposição de um modelo de representação da interação dos agentes e conseqüentemente de um artefato da Engenharia de Software Orientada a Agentes para modelar essa interação.
- Análise de diferentes metodologias da Engenharia de Software Orientada a Agentes: MaSE, Prometheus e Tropos, além da linguagem de modelagem AUML, enfatizando as vantagens e desvantagens de cada uma delas.
- Proposta de uma metodologia unificada para o desenvolvimento de sistemas orientados a agentes, elaborada a partir da análise feita das metodologias MaSE, Prometheus e Tropos e da linguagem de modelagem AUML.
- Estudo de caso para validação da metodologia proposta e avaliação das metodologias estudadas.

Para a elaboração da metodologia unificada, primeiramente foi necessário entender o uso que se faz do conceito de agentes dentro da Engenharia de Software. Esse conceito é muitas vezes utilizado de maneira ambígua na literatura, ora sendo caracterizado como componente de software simples, ora sendo caracterizado como uma especialização de objetos. A contribuição deste trabalho neste sentido foi a proposição do conceito de agente como um modelo de interação distinto, onde o agente busca as informações do ambiente por meio de seus sensores e atua no seu ambiente por meio dos atuadores. Diferentemente das abordagens existentes, por meio do modelo de interação proposto, a representação da interação do agente não é realizada de maneira estática, na qual o agente fica esperando pela informação, mas é uma interação dinâmica onde o agente busca as informações distribuídas em *buffers* dentro do seu ambiente.

Após essa proposição do conceito de interação do agente, foi realizada uma análise de diferentes metodologias da Engenharia de Software Orientada a Agentes: MaSE, Prometheus e

Tropos, além da linguagem de modelagem AUML. A escolha de tais metodologias foi por possuírem documentações detalhadas e de fácil acesso, além de estarem bem difundidas na comunidade que estuda agentes. Utilizamos critérios sistemáticos para a avaliação dessas metodologias e conseqüente seleção dos artefatos para comporem a Metodologia Unificada proposta. Esses critérios englobam as características dos agentes (autonomia, pró-atividade, reatividade, protocolos de conversação), os aspectos da usabilidade das metodologias (clareza / entendimento e facilidade de uso), critérios técnicos (rastreabilidade) e as fases de desenvolvimento de sistema cobertas pela metodologia. Além desses critérios, modelamos um estudo de caso utilizando cada uma das metodologias estudadas, o que serviu de base para a elaboração da Metodologia Unificada.

As metodologias MaSE, Prometheus e Tropos suportam os conceitos básicos da orientação a agentes, como a autonomia. A identificação dos agentes começa com entidades menores, como por exemplo, os papéis na metodologia MaSE., sendo que na metodologia Prometheus, a identificação dos agentes é feita por meio do Diagrama de Ligação de Dados, o qual converte funcionalidades em agentes.

O levantamento dos objetivos iniciais do sistema é explicitamente identificado somente na metodologia Tropos, por meio dos Diagramas de Ator e de Raciocínio, sendo que as demais metodologias começam a modelagem a partir de objetivos previamente estabelecidos. MaSE possui o Diagrama Hierárquico de Objetivos, que organiza os objetivos do sistema de maneira hierárquica. Com relação aos objetivos dos agentes, todas apresentam técnicas para representá-los, porém Prometheus possui o Descritor do Agente que detalha como o agente atinge seu objetivo.

O critério de reatividade, que verifica as respostas do agente às mudanças do ambiente, é melhor detalhado na metodologia Prometheus, por meio dos Descritores de Plano e de Evento.

Com relação aos protocolos de conversação, todas as metodologias possuem mecanismos de troca de mensagens entre os agentes, sendo que a maioria utiliza os artefatos da AUML. MaSE, Prometheus e Tropos utilizam os diagramas de seqüência/interação para representar as interações, sendo que Prometheus e Tropos mostram as interações entre os agentes, enquanto as interações do diagrama de seqüência são entre papéis. Porém, nenhuma delas aborda o conceito proposto neste trabalho, onde o agente busca as mensagens do ambiente.

Essas metodologias apresentam notações de fácil entendimento e semântica clara, o que reduz a ambigüidade dos modelos. Existem similaridades em alguns artefatos das metodologias que representam a estrutura estática e dinâmica do agente, principalmente na fase de projeto.

Todas cobrem as fases de análise de requisitos e projeto arquitetural. Tropos é a única metodologia que aborda a fase de requisitos iniciais, MaSE e Prometheus abordam a fase de projeto detalhado. As fases de implementação, teste e manutenção não são definidas explicitamente pelas metodologias.

A AUML não dá suporte ao ciclo completo de desenvolvimento orientado a agente, sendo esse suporte maior à fase de projeto. Da mesma forma, não modela conhecimento compartilhado, nem o relacionamento social. Como já dissemos anteriormente, não é considerada uma metodologia e sim uma linguagem de modelagem. Os diagramas gerados pela AUML são utilizados em outras metodologias orientadas a agentes, principalmente nas que surgiram como uma especialização das abordagens orientadas a objetos.

Os maiores problemas existentes nas metodologias orientadas a agentes estão relacionados com a falta de documentação adequada (a maior parte do material está na forma de artigos ou tutoriais), de ferramentas adequadas que suportem todas as fases da metodologia e de medidas comparativas da qualidade dos sistemas orientados a agentes.

Com o objetivo de melhorar a notação e a facilidade de uso dos artefatos selecionados na Metodologia Unificada, identificamos a necessidade de adaptações de tais artefatos, como é o caso do Diagrama de Ator da metodologia Tropos. Além disso, foram inseridos na Metodologia Unificada o Diagrama RP-Net e o Descritor da Estrutura Interna do Agente, para satisfazerem algumas necessidades que não foram abordadas nas metodologias avaliadas. O Diagrama de Interação RP-Net substitui os diagramas de interação existentes nas metodologias estudadas, com o objetivo de suprir algumas limitações do tipo: dificuldade para modelar uma grande quantidade de interações, dificuldade para modelar mensagens assíncronas e concorrentes e não representam explicitamente o comportamento dinâmico dos sistemas orientados a agentes. O Descritor da Estrutura Interna do Agente detalha a estrutura interna estática de cada agente, identificando as atividades que o agente deve executar para atingir seus objetivos.

Uma vez identificados os benefícios e as deficiências das metodologias estudadas, obtivemos subsídios técnicos para elaborarmos a metodologia unificada para o desenvolvimento de sistemas orientados a agentes, o principal objetivo desse trabalho. A idéia de unificação das metodologias segue o mesmo princípio do *Rational Unified Process* (RUP), que surgiu a partir do trabalho de Grady Booch (com o Método Booch), de James Rumbaugh (com a Técnica de Modelagem de Objetos) e de Ivar Jacobson (com o Método da Engenharia de Software Orientada a Objetos). Assim, a Metodologia Unificada representa uma perspectiva diferente na conjunção de várias metodologias orientadas a agentes, surgindo como uma visão que uniformiza os artefatos de tais metodologias.

A metodologia unificada proposta está fundamentada nas propriedades e características inerentes a um sistema multi-agentes, como autonomia, pró-atividade, reatividade e habilidade social. Consiste de um conjunto de atividades e artefatos necessários para o desenvolvimento de sistemas orientados a agentes. Na Tabela 8.1, é apresentado um sumário contendo as fases, atividades e artefatos produzidos na Metodologia Unificada.

Tabela 8.1: Resumo da Metodologia Unificada.

Fases	**Atividades**	**Artefatos**
Fase de Análise	1. Identificar os objetivos 2. Estruturar os objetivos 3. Identificar as funcionalidades	1.1 Diagrama de Ator 1.2 Diagrama de Objetivos 2.1 Diagrama Hierárquico de Objetivos 3.1 Modelo de Funcionalidades
Fase de Projeto Arquitetural	1. Identificar os tipos de agentes 2. Refinar os agentes 3. Identificar as interações entre os agentes 4. Revisar o sistema	1.1 Diagrama de Agrupamento de Funcionalidades 2.1 Modelo de Descritor do Agente 3.1 Diagrama RP-Net 4.1 Diagrama de Revisão do Sistema
Fase de Projeto Detalhado	1. Construir a estrutura interna do agente 2. Construir a estrutura dinâmica do agente	1.1 Descritor da Estrutura Interna do Agente 2.1 Diagrama de Estados

Para validação da metodologia proposta, foi modelado um Sistema Multi-agentes de Busca de Informações Científicas. Deste estudo de caso pudemos concluir preliminarmente que a metodologia unificada se mostrou satisfatória pelas seguintes razões:

- representa uma melhoria no processo de desenvolvimento de sistema, aproveitando as vantagens e evitando as desvantagens encontradas nas metodologias estudadas;
- suporta as características básicas do agente de autonomia, pró-atividade, reatividade e habilidade social;
- representa o novo modelo de interação de agente proposto neste trabalho: por meio do Diagrama da RP-Net é possível representar a interação do tipo busca por mensagens, específica do agente;
- aborda as fases de desenvolvimento de software: de análise, projeto arquitetural e projeto detalhando. Inicia com a identificação dos objetivos do sistema e é mais abrangente do que as metodologias estudadas;
- possui clareza na identificação dos objetivos do sistema, por meio dos artefatos Diagrama de Ator e Diagrama de Objetos;
- possui clareza na definição dos agentes, pois são identificados de forma mais clara e coerente, pelo Diagrama de Agrupamento de Funcionalidades;
- proporciona certa rastreabilidade entre seus artefatos, em particular, o Diagrama de Revisão do Sistema representa um importante artefato de rastreabilidade, uma vez que apresenta os agentes, suas percepções, suas ações, suas mensagens e os dados utilizados;
- a representação gráfica adotada na metodologia contribui para a facilidade de uso e entendimento dos artefatos propostos;
- modela a estrutura interna estática e dinâmica do agente por meio do Descritor da Estrutura Interna do Agente e do Diagrama de Estados.

No entanto, baseando-se no estudo de caso realizado neste trabalho observamos algumas características criticáveis. O Diagrama de Revisão do Sistema pode ficar muito complexo quando são modelados sistemas grandes, dificultando a interpretação. Contudo existe a possibilidade de utilizar a RP-Net como uma ferramenta de Revisão do Sistema, o que diminuiria substancialmente o problema de escalabilidade encontrado no Diagrama de Revisão do Sistema. Essa possibilidade poderia ser investigada em um trabalho futuro. Outro ponto levantado pelo estudo de caso diz respeito à complexidade dos modelos, como o Diagrama de Ator e de Objetivos. Estes modelos podem se tornar bastante complicados, dependendo do domínio da aplicação tratada. Para gerenciar esta complexidade, mecanismos de estruturação precisam ser aplicados de forma a tornar os modelos mais simples e fáceis de entender. Aqui também consideramos pertinente uma investigação mais detalhada no futuro.

Sumarizando todos esses pontos concluímos que com a elaboração de uma Metodologia Unificada, que captura o que há de melhor em cada metodologia estudada, pode-se dizer que se obteve um salto qualitativo na Engenharia de Software Orientada a Agentes, mas além das diversas contribuições adicionadas por este trabalho, um longo caminho ainda está pela frente, sendo que um grande potencial para trabalhos futuros se abre no horizonte.

8.2 Trabalhos Futuros

Este trabalho certamente é um ponto de partida para uma longa linha de pesquisa relacionada à Engenharia de Software Orientada a Agentes. Além disso, com base no estudo de caso realizado temos verificado dificuldades e deficiências da proposta. A seguir, serão descritas algumas sugestões para trabalhos futuros:

- Estudos de Caso: neste trabalho, a metodologia proposta foi validada por meio de um pequeno estudo de caso. Porém, é necessário utilizar a metodologia em outras aplicações no desenvolvimento de sistema multi-agentes, em diferentes áreas de aplicações, para a validação efetiva da metodologia.
- Desenvolvimento de ferramenta: a elaboração de uma ferramenta que suporte todo o ciclo de vida da metodologia, facilitaria sua utilização e representaria um grande diferencial da metodologia proposta.
- Sociedade de Agentes: na metodologia proposta, não foi abordado explicitamente o conceito de sociedade de agentes, incluindo colaboração, cooperação e competição dos agentes e critérios para a formação da sociedade. Seria interessante a criação de um artefato que tratasse esse conceito, uma vez que fazem parte da característica de um sistema multi-agentes.
- Avaliação da metodologia: a avaliação das metodologias apresentadas neste trabalho utilizou critérios sistemáticos de avaliação e um estudo de caso. Uma sugestão seria fazer uma comparação dessas metodologias na comunidade acadêmica e industrial.
- Metamodelo: os metamodelos servem como um guia para construir modelos para o desenvolvimento de sistemas. A construção de um metamodelo específico para tratar os artefatos propostos neste trabalho contribuiria nos estudos da Engenharia de Software, principalmente pelo fato de que fortaleceria a distinção entre os artefatos específicos da orientação a agentes e da orientação a objetos.
- Metodologia Heterogênea: a metodologia proposta é adequada somente para o desenvolvimento de sistemas multi-agentes. Um estudo bastante interessante seria adequar a metodologia para uma abordagem heterogênea, incluindo agentes e objetos em um mesmo sistema.

8.3 Considerações Finais

A área de Engenharia de Software Orientada a Agentes está constantemente se aperfeiçoando, procurando metodologias consolidadas. Tentamos aqui propor uma metodologia unificada, reutilizando artefatos de outras metodologias, evitando assim esforços na criação de novos artefatos, mas avaliando os artefatos existentes nas metodologias estudadas. A Metodologia Unificada proposta surge como uma alternativa real a outras metodologias existentes para modelagem de sistemas multi-agentes, mas ainda demanda alguns aperfeiçoamentos, em particular, devido à simplicidade das fases e dos artefatos utilizados na metodologia. Nossa conclusão é a de que ainda há muito trabalho a ser feito em termos de desenvolvimento de sistema multi-agentes. Os esforços são modestos tendo em vista o potencial de aplicação de sistemas multi-agentes. Esperamos que este trabalho realmente esteja na direção das necessidades da Engenharia de Software Orientada a Agentes.

Referências Bibliográficas

[Agre 1989] Agre, P. E.: **The Dynamic Structure of Everyday Life.** PhD Thesis, Department of Electrical Engineering and Computer Science, MIT, 1989.

[Bates 1994] Bates, J.; Loyall, A. B.; Reilly, W. S.: **An Architecture for Action, Emotion and Social Behavior.** In Proceedings of the Fourth European Workshop on Modelling Autonomous Agents in Multi-Agent World, Lecture Notes in Computer Science, Vol. 830, pp. 55-68, London, 1994.

[Bates 1994] Bates, J.: **The Role of Emotion in Belierable Agents**. Communications of ACM, Vol. 37, n. 7, pp. 122-125, 1994.

[Bellifemine 2003] Bellifemine, F.; Caire, G.; Poggi, A.; Rimassa, G.: **JADE: A White Paper.** Telecom Italia EXP Magazine, Vol. 3, n. 3, 2003.

[Bergenti 2000] Bergenti, F.; Poggi, A.: **Exploiting UML in the Design of Multi-Agent Systems.** In Engineering Societies in the Agent World, First International Workshop, ESAW 2000, Lecture Notes in Computer Science, Vol. 1972, pp. 106-113, Germany, 2000.

[Bresciani 2002] Bresciani, P.; Perini, A.; Giorgini, P.; Giunchiglia, F.; Mylopoulos, J.: **Modeling Early Requirements in Tropos: A Transformation Based Approach.** Second International Workshop on Agent-Oriented Software Engineering, Lecture Notes in Computer Science, Vol. 2222, pp. 151-168, Canada, 2002.

[Bresciani 2004] Bresciani, P.; Giorgini, P.; Giunchiglia, F.; Mylopoulos, J.; Perini, A.: **Tropos: An Agent-Oriented Software Development Methodology.** In Journal of Autonomous Agents and Multi-Agent Systems, . Klumer Academic Publishers, 2004.

[Bratman 1988] Bratman, M. E.; Israel, D. J.; Pollack, M. E.: **Plans and Resource-Bounded Practical Reasoning.** Computational Intelligence – Vol. 4, pp. 349-355, 1988.

[Brazier 1997] Brazier, F. M. T.; Dunin-Keplicz, B. M.; Jennings, N. R.; Treur, J.: **DESIRE: Modelling Multi-Agent in a Compositional Formal Framework.** International Journal of Cooperative Information Systems, Vol. 6, n. 1, pp. 67-94, 1997.

[Brooks 1986] Brooks, R.: **A Robust Layered Control System for a Mobile Robot.** IEEE Journal of Robotics and Automation – Vol. 2, n. 1, pp. 14-23, 1986.

[Burrafato 2002] Burrafato, P.; Cossentino, M.: **Designing a Multi-Agent Solution for Bookstore with PASSI Methodology.** Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems, Toronto, 2002.

[Busetta 1999] Busetta, P.; Ronnquist, R.; Hodgson, A.; Lucas A.: **Jack Intelligent Agents - Components for Intelligent Agents in Java.** AOS Technical Report TR9901, 1999.

[Bush 2001] Bush, G.; Cranefield, S.; Purvis, M.: **The Styx agent methodology.** The Information Science Discussion Paper Series 2001/2002, University of Otago, New Zealand, 2001.

[Caire 2001] Caire, G.; Leal, F.; Chainho, P.; Evans, R.; Jorge, F. G.; Pavon, G. J.; Kearney, P.; Stark, J.; Massonet, P.: **MESSAGE: Methodology for Engineering Systems of Software AGEnts.** Technical Report P907, European Institute for Research and Strategic Studies in Telecommunications, EURESCOM, 2001.

[Castro 2002] Castro, J.; Kolp, M.; Mylopoulos, J.: **Towards Requirements-Driven Information Systems Engineering: The Tropos Project.** Information Systems, Vol. 27, n. 6, pp. 365-389, Elsevier, Amsterdam, The Netherlands, 2002.

[Chang 1997] Chang, W.-T; Ha, S.; Lee, E. A.: **Heterogeneous Simulation – Mixing Discrete-Event Models with Dataflow.** Journal of VLSI Signal Processing Systems for Signal Image and Video Techno, Vol. 15, n. 1, pp. 127-144, 1997.

[Chung 2000] Chung, L.; Nixon, B. A.; Yu, E.; Mylopoulos, J.: **Non-functional Requirements in Software Engineering.** The Kluwer International Series in Software Engineering, Vol. 5, 2000.

[Ciancarini 2001] Ciancarini, P.; Wooldridge, M. (editors).: **Agent-Oriented Software Engineering.** First International Workshop on Agent-Oriented Software Engineering, Lecture Notes in Computer Science, Vol. 1957, 2001.

[Collinot 1996] Collinot, A.; Drogoul, A.; Benhamou, P.: **Agent Oriented Design of a Soccer Robot Team.** In Proceedings of the Second International Conference on Multi-Agent Systems, pp. 41-47, Japan, 1996.

[Cliff 1999] Cliff, D.; Grand, S.: **The Creatures Global Digital Ecosystem.** Artificial Life, Vol. 5, n. 1, pp. 77-93, 1999.

[Dam 2003] Dam, K. H.: **Evaluating and Comparing Agent-Oriented Software Engineering Methodologies.** Master Thesis of Applied Science in Information Technology, RMIT University, Australia, 2003.

[Debenham 2002] Debenham, J. K.; Henderson-Sellers, B.: **Full Lifecycle Methodologies for Agent-Oriented Systems – The Extended OPEN Process Framework.** In Proceedings of Agent-Oriented Information Systems, pp. 87-101, Toronto, 2002.

[DeLoach 2001] DeLoach, S. A.: **Analysis and Design using MaSE and AgentTool.** In Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference, Miami University, Oxford, 2001.

[DeLoach 2003] DeLoach, S.A.; Matson, E. T.; Li, Y.: **Exploiting Agent Oriented Software Engineering in the Design of a Cooperative Robotics Search and Rescue System.** The International Journal of Pattern Recognition and Artificial Intelligence, Vol. 17, n. 5, pp. 817-835, August 2003.

[Eugster 2003] Eugster, P. T.; Felber, P. A.; Guerraoui, R.; Kemarrec, A-M.: **The Many Faces of Publish/Subscribe.** ACM Computing Surveys, Vol. 35, n. 2, pp. 114-131, 2003.

[Etzioni 1994] Etzioni, O.; Weld, D.: **A Softbot-Based Interface to the Internet.** Communications of ACM, Vol. 37, n. 7, pp. 72-76, 1994.

[Fiadeiro 1991] Fiadero, J.; Maibaum, T.S.E.: **Describing, Structuring and Implementing Objects.** In Proceedings of the REX School/Workshop on Foundations of Object-Oriented Languages, Lecture Notes in Computer Science, Vol. 489, pp; 274-310, New York, 1991.

[Figueiredo 2003] Figueiredo, O. A.: **Implementação de Espaços de Tuplas do Tipo JavaSpace.** Dissertação de Mestrado, ICMC/USP, São Carlos, 2003.

[Finin 1998] Finin, T.; Nicholas, C.: **Software Agents for Information Retrieval.** In Proceedings of the Third ACM Conference on Digital Libraries, Pittsburgh, 1998.

[Franklin 1996] Franklin, S.; Graesser, A.: **Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents.** In Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Lecture Notes in Artificial Intelligence, Vol. 1193, pp. 21-36, 1996.

[Freitas 2002] Freitas, F. L. G.: **Sistemas Multiagentes Cognitivos para Recuperação, Classificação e Extração Integradas de Informação da Web.** Tese de Doutorado, UFSC, Florianópolis, 2002.

[Gelernter 1985] Gelernter, D.: **Generative Communication in Linda.** In ACM Transactions on Programming Languages and Systems, Vol. 7, n. 1, pp.80-112, 1985.

[Giorgini 2001] Giorgini, P.; Perini, A.; Mylopoulos J.; Giunchiglia, F.; Bresciani, P.: **Agent-Oriented Software Development: A Case Study.** Proceedings of the Thirteenth International Conference on Software Engineering & Knowledge Engineering, Argentina, 2001.

[Giorgini 2004] Giorgini, P.; Müller, J. P.; Odell J. (editors).: **Agent-Oriented Software Engineering IV.** Fourth International Workshop on Agent-Oriented Software Engineering, Lecture Notes in Computer Science, Vol. 2935, 2004.

[Giunchiglia 2002] Giunchiglia, F.; Mylopoulos, J.; Perini, A.: **The Tropos Software Development Methodology: Processes, Models and Diagrams.** In Proceedings of the First International Joint Conference on Autonomous Agents & Multiagent Systems, Italy, 2002.

[Giunchiglia 2003] Giunchiglia, F.; Odell, J.; Weiss, G. (editors).: **Agent-Oriented Software Engineering III.** Third International Workshop on Agent-Oriented Software Engineering, Lecture Notes in Computer Science, Vol. 2585, 2003.

[Gmytrasiewicz 2001] Gmytrasiewicz, P. J.; Lisetti, C. L.: **Emotions and Personality in Agent Design and Modeling**. In Proceedings of 8th International Workshop Series – Agent Theories, Architectures and Languages, Lecture Notes in Computer Science, Vol. 2333, 2001.

[Gomes 2000] Gomes, A. S. R.: **Contribuições ao Estudo da Rede de Agentes.** Dissertação de Mestrado, DCA/FEEC/UNICAMP, Campinas, 2000.

[Gomes 2003] Gomes, E. R.; Silveira, R. A.; Vicari, R. M.: **Utilização de agentes FIPA em ambientes para Ensino a Distância.** XXIX Conferência Latino Americana de Informática, La Paz, 2003.

[Gudwin 1996] Gudwin, R. R.: **Contribuições ao Estudo Matemático de Sistemas Inteligentes.** Tese de Doutorado, DCA/FEEC/UNICAMP, Campinas, 1996.

[Gudwin 2001] Gudwin, R. R.: **Introdução à Teoria de Agentes.** Notas de aula, disponível em http://www.dca.fee.unicamp.br/~gudwin/courses/IA009/. Acesso em: 13/05/2005.

[Gudwin 2003] Gudwin, R. R.: **Semiônica: Uma proposta de Contribuição à Semiótica Computacional.** Tese de Livre Docência, DCA/FEEC/UNICAMP, Campinas, 2003.

[Gudwin 2004] Gudwin, R. R.: **Semionics: A Proposal for the Semiotic Modelling of Organizations.** The 6th International Workshop on Organisations Semiotics - Virtual, Distributed and Flexible Organisations, Kluwer Academic Publishers, pp. 15-34, 2004.

[Guerrero 1999] Guerrero, J. A. S; Gomes, A. S. R; Gudwin, R. R.: **A Computational Tool to Model Intelligent Systems.** Anais do 4º. Simpósio Brasileiro de Automação Inteligente, pp. 227-232, São Paulo, 1999.

[Hewitt 1977] Hewitt, C.: **Viewing Control Structures as Patterns of Passing Messages.** Journal of Artificial Intelligence, Vol. 8, n. 3, pp. 323-364, 1977.

[Hoyle 1997] Hoyle, M.; Lueg, C.: **Open Sesame: A look at Personal Assistants.** In Proceedings of the Second International Conference on Practical Applications of Intelligent Agents & Multi-Agent Technology, pp. 51-60, 1997.

[Hubner 2000] Hubner, J. F.; Sichman, J. S.: **SACI: Uma Ferramenta para Implementação e Monitoração da Comunicação entre Agentes.** International Joint Conference, 7th Ibero-American Conference on AI, pp. 47-56, Instituto de Ciências Matemáticas e de Computação, ICMC/USP, 2000.

[IEEE 1990] IEEE Std 610.12 - *IEEE Standard Glossary of Software Engineering Terminology*, 1990.

[Iglesias 1996] Iglesias, C. A.; Garijo, M.; Gonzalez, J. C.; Velasco, J. R.: **A Methodological Proposal for Multiagent Systems Development Extending CommonKADS.** In Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop, Canada, 1996.

[Iglesias 1999] Iglesias, C. A.; Garijo, M.; Gonzalez, J. C.: **A Survey of Agent-Oriented Methodologies.** In Proceedings of the 5th International Workshop on Agent Theories, Architectures, and Languages, Lecture Notes in Artificial Intelligence, Vol. 1555, pp. 317-330, Paris, 1999.

[Jennings 2000a] Jennings, N. R.: **On Agent-Based Software Engineering.** Artificial Intelligence, Vol. 117, n. 2, pp .277-296, 2000.

[Jennings 2000b] Jennings, N. R.; Faratin, P.; Norman, T. J.; O'Brien, P.; Odgers, B.; Alty, J. L.: **Implementing a Business Process Management System using ADEPT: A Real-World Case Study.** International Journal of Applied Artificial Intelligence, Vol. 14, n. 5, pp. 421-465, 2000.

[Jennings 2001] Jennings, N. R.; Wooldridge, M.: **Agent-Oriented Software Engineering.** Handbook of Agent Technology (ed. J. Bradshaw), AAAI/MIT Press, 2001.

[Kendall 1995] Kendall, E. A.; Malkoun, M. T.; Jiang, C. H.: **A Methodology for Developing Agent Based Systems.** First Australian Workshop on Distributed Artificial Intelligence, pp. 85-99, 1995.

[Kinny 1996] Kinny, D.; Georgeff, M. P.: **Modeling and Design of Multi-Agent Systems.** Intelligent Agents III: Third International Workshop on Agent Theories, Architectures, and Languages, 1996.

[Kozierok 1993] Kozierok, R.; Maes, P.: **A Learning Interface Agent for Scheduling Meetings.** In Proceedings of the ACM-SIGCHI International Workshop on Intelligent User Interfaces, pp. 81-88, New York, 1993.

[Kuwabara 1995] Kuwabara, K.; Ishida, T.; Osato, N.: **AgentTalk: Coordination Protocol Description for Multiagent Systems.** In Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, 1995.

[Lacey 2000] Lacey, T. H; DeLoach, S. A.: **Verification of Agent Behavioral Models**. In Proceedings of the International Conference on Artificial Intelligence, CSREA Press, pp. 557-564, Las Vegas, 2000.

[Laugier 1992] Hassoun, M.; Demazeau, Y.; Laugier, C.: **Motion Control for a Car-like Robot: Potential Field and Multi-Agent Approaches.** In Proceedings of the International Workshop on Intelligent Robots and Systems, 1992.

[Levy 1994] Levy, A. Y.; Sagiv, Y.; Srivastava, D.: **Towards Efficient Information Gathering Agents.** In Working Notes of the 1994 AAAI Spring Symposium on Software Agents, pp. 64-70, 1994.

[Lind 2000] Lind, J.: **MASSIVE: Software Engineering for Multiagent Systems.** PhD Thesis, University of Saarland, 2000.

[Maes 1994a] Maes, P.: **Social Interface Agents: Acquiring Competence by Learning from Users and other Agents.** In Working Notes of the AAAI Spring Symposium on Software Agents, pp. 71-78, 1994.

[Maes 1994b] Maes, P.: **Agents that Reduce Work and Information Overload.** Communications of the ACM, Vol. 37, n. 7, pp. 31-40, 1994.

[Muhl 2002] Muhl, G.; Fiege, L.; Buchmann, A. P.: **Filter Similarities in Content-Based Publish/Subscribe Systems**. International Conference on Architecture of Computing Systems, Lecture Notes in Computer Science, Vol. 2299, pp. 224-238, Germany, 2002.

[Murata 1989] Murata, T.: **Petri nets: Properties, Analysis and Applications.** In Proceedings of the IEEE, Vol. 77, n. 4, pp. 541-580, 1989.

[Murch 1999] Murch, R.; Johnson, T.: **Intelligent Software Agents.** Prentice Hall – New Jersey, 1999.

[Mylopoulos 2002] Mylopoulos, J.; Kolp, M.; Giorgini, P.: **Agent-Oriented Software Development.** In Proceedings of the Second Hellenic Conference on Artificial Intelligence: Methods and Applications of Artificial Intelligence, Lecture Notes in Computer Science, Vol. 2308, pp.3-17, 2002.

[Nwana 1996a] Nwana, H. S.: **Software Agents: An Overview.** Knowledge Engineering Review, Vol. 11, n. 3, pp. 1-40, 1996.

[Nwana 1996b] Nwana, H. S.; Ndumu, D. T.: **An Introduction to Agent Technology.** British Telecommunications Technology Journal, Vol. 14, n. 4, pp.55-67, 1996.

[Nwana 1999] Nwana, H. S.; Ndumu, D. T.; Lee, L. C.; Collis, J. C.: **ZEUS: A Toolkit for Building Distributed Multi-Agent Systems.** In Proceedings of the Third International Conference on Autonomous Agents, ACM Press, 1999.

[O'Hare 1996] O'Hare, G.; Jennings, N. R.: **Foundations of Distributed Artificial Intelligence.** New York, 1996.

[O'Malley 2002] O'Malley, S. A.; DeLoach, S. A.: **Determining When to Use an Agent-Oriented Software Engineering Methodology.** In Proceedings of the Second International Workshop on Agent-Oriented Software Engineering, Lectures Notes in Computer Science, Vol. 2222, pp. 188-205, 2002.

[Odell 2000] Odell, J.; Parunak, H. V. D.; Bauer, B.: **Extending UML for Agents.** In Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence, pp.3-17, 2000.

[Odell 2002] Odell, J.: **Objects and Agents Compared.** Journal of Object Technology, Vol. 1, n. 1, pp.41-53, 2002.

[Odell 2005] Odell, J.; Giorgini, P.; Müller, J. P. (editors): **Agent-Oriented Software Engineering V.** Fifth International Workshop on Agent-Oriented Software Engineering, Lecture Notes in Computer Science, Vol. 3382, 2005.

[Padgham 2002a] Padgham, L; Winikoff, M.: **Prometheus: A Methodology for Developing Intelligent Agents.** In Proceedings of the Third International Workshop on Agent-Oriented Software Engineering, at Autonomous Agents and Multi-Agent Systems (AAMAS), Italy, 2002.

[Padgham 2002b] Padgham, L; Winikoff, M.: **Prometheus: A Pragmatic Methodology for Engineering Intelligent Agents.** In Proceedings of the Workshop on Agent-Oriented Methodologies at Object-Oriented Programming, Systems, Languages and Applications (OOPSLA), Seattle, 2002.

[Padgham 2002c] Padgham, L.: **Design of Multi Agent Systems.** Tutorial at Net Object Days (NODe 2002), School of Computer Science and Information Technology, RMIT University, 2002.

[Padgham 2005] Padgham, L; Winikoff, M.: **Prometheus: A Practical Agent-Oriented Methodology.** Chapter 5 in Agent-Methodologies, edited by B. Henderson-Sellers and P.Giorgini, Idea Group, 2005.

[Peirce 1935]. Peirce, C. S.: **The Collected Papers of Charles Sanders Peirce**. Hartshorne, C. e Weiss, P. (Ed.). Cambridge, USA: Harvard University Press, 1931-1935, Vols. I-VI. (citado como CP seguido de volume e páragrafo).

[Perini 2001] Perini, A.; Bresciani, P.; Giorgini, P.; Giunchiglia, F.; Mylopoulos, J.: **Towards an Agent Oriented Approach to Software Engineering.** In Proceedings of the Workshop Dagli Oggetti agli Agenti: Tendenze Evolutive dei Sistemi Software, Italy, 2001.

[Pressman 2002] Pressman, R. S.: Engenharia de Software. 5ª Edição, McGraw-Hill, Rio de Janeiro, 2002.

[Rao 1995] Rao, A. S; Georgeff, M. P.: **BDI-Agents: From Theory to Practice.** In Proceedings of the First International Conference on Multiagent Systems, pp. 312-319, San Francisco, 1995.

[Rao 1996] Rao, A. S.: **AgentSpeak(L): BDI Agents Speak out in a Logical Computable Language.** In Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, 1996.

[Rational 2001] **IBM Rational Unified Process: Best Practices for Software Development Teams.** Rational Software White Paper, TP026B, Rev 11/01, ftp://ftp.software.ibm.com/software/rational/web/whitepapers/2003/rup_bestpractices.pdf. Acesso em: 13/05/2005.

[Robinson 2000] Robinson, D. J.: **A Component Based Approach to Agent Specification.** MS Thesis, School of Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base Ohio, USA, 2000.

[Russel 1995] Russel, S. J.; Norvig, P.: **Artificial Intelligence: A Modern Approach.** Artificial Intelligence. Prentice Hall, 1995.

[Sarmento 2004] Sarmento, L. M.: **An Emotion-Based Agent Architecture**. Dissertação de Mestrado, Faculdade de Ciências da Universidade do Porto.

[Sannicolo 2002] Sannicolo, F.; Perini, A.; Giunchiglia, F.: **The Tropos Modeling Language. A User Guide.** Technical Report DIT-02-61, Informatica e Telecomunicazioni, Università degli Studi di Trento, 2002.

[Sichman 1992] Sichman, J. S.; Demazeau, Y.; Boissier, O.: **When Can Knowledge-Based Systems Be Called Agents?** In Proceedings of the Ninth Brazilian Symposium on Artificial Intelligence, pp.172-185, Rio de Janeiro, 1992.

[Silva 2003] Silva, C. T. L.: **Detalhando o Projeto Arquitetural no Desenvolvimento de Software Orientado a Agentes: O Caso Tropos.** Dissertação de Mestrado, Universidade Federal de Pernambuco, 2003.

[Silva 2005] Silva, C. T. L.; Castro, J. F. B.; Tedesco, P. C. A. R.; Silva, I. G. L.: **Describing Agent-Oriented Design Patterns in Tropos.** XIX Simpósio Brasileiro de Engenharia de Software, Uberlândia, Brasil, 2005.

[Sloman 2001] Sloman, A.: **Beyond Shallow Models of Emotions**. Cognitive Processing, Vol. 2, n. 1, pp. 177-198, 2001.

[Sommerville 2003] Sommerville, I.: **Engenharia de Software.** 6ª Edição, Addison Wesley, São Paulo, 2003.

[Tveit 2001] Tveit, A.: **A Survey of Agent-Oriented Software Engineering.** In Proceedings of the First NTNU Computer Science Graduate Student Conference, Norwegian University of Science and Technology, 2001.

[Tatai 2003] Tatai, V. K.: **Técnicas de Sistemas Inteligentes Aplicadas ao Desenvolvimento de Jogos de Computador.** Dissertação de Mestrado, DCA/FEEC/UNICAMP, Campinas, 2003.

[Wagner 2000] Wagner, G.: **Agent-Object-Relationship Modeling.** In Proceedings of the Second International Symposium from Agent Theory to Agent Implementation together with EMCRS, 2000.

[Wagner 2003] Wagner, G.: **A UML Profile for External AOR Models.** In Proceedings of the Third International Workshop on Agent-Oriented Software Engineering, Lecture Notes ins Computer Science, Vol. 2585, 2003.

[Weiss 2002] Weib, G.: **Agent Orientation in Software Engineering.** The Knowledge Engineering Review, Vol. 16, n. 4, 2001. pp. 349-373, 2002.

[Wood 2001] Wood, M. F.; DeLoach, S. A.: **An Overview of the Multiagent Systems Engineering Methodology.** In Proceedings of the First International Workshop on Agent-Oriented Software Engineering, Lecture Notes in Computer Science, Vol. 1957, 2001.

[Wooldridge 1995] Wooldridge, M. J.; Jennings, N. R.: **Intelligent Agents: Theory and Practice.** The Knowledge Engineering Review, Vol. 10, n. 2, pp.115-152, 1995.

[Wooldridge 1999a] Wooldridge, M. J.: **Intelligent Agents.** In G. Weiss, editor: Multiagent Systems, The MIT Press, 1999.

[Wooldridge 1999b] Wooldridge, M. J.; Jennings, N. R.: **Software Engineering with Agents: pitfalls and pratfalls.** In IEEE Internet Computing, Vol. 3, n. 3, pp. 20-27, 1999.

[Wooldridge 2000] Wooldridge, M. J.; Jennings, N. R.; Kinny, D.: **The GAIA Methodology for Agent-Oriented Analysis and Design.** In Journal of Autonomous Agents and Multi-Agent Systems, Vol. 3, n. 3, pp. 285-312, 2000.

[Wooldridge 2002] Wooldridge, M. J.; Weiss, G.; Ciancarini, P. (editors): **Agent-Oriented Software Engineering II.** Second International Workshop on Agent-Oriented Software Engineering, Lecture Notes in Computer Science, Vol. 2222, 2002.

[Yourdon 1990] Yourdon, E.: **Análise Estruturada Moderna.** Campus, Rio de Janeiro, 1990.

[Yu 1997] Yu, E. S. K.: **Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering.** In Third IEEE International Symposium on Requirements Engineering (RE'97), 1997.

[Zambonelli 2001] Zambonelli, F.; Jennings, N. R.; Wooldridge, M.: **Organizational Abstractions for the Analysis and Design of Multi-Agent Systems.** In Proceedings of the First International Workshop on Agent-Oriented Software Engineering, Lecture Notes in Computer Science, Vol. 1957, 2001.

[Zambonelli 2003] Zambonelli, F.; Jennings, N. R.; Wooldridge, M.: **Developing Multiagent Systems: the GAIA Methodology.** ACM Transactions on Software Engineering and Methodology, Vol. 12, n. 3, 2003.

ANEXO A

Artefatos da Metodologia Prometheus

A seguir são apresentados detalhadamente os artefatos que constituem a metodologia Prometheus.

1. Objetivos e Funcionalidades.
Identificar os objetivos do sistema e as funcionalidades necessárias.

Objetivos:
……
……
……
……
Funcionalidades: Objetivo Associado #
……
……
……
……

2. Descritor de Funcionalidade.
Descrição de projeto de alto nível para cada funcionalidade.

Nome:
……
……
Descrição (2-3 sentenças):
……
……
……
Percepções:
……
……
Ações:
……
……
……
Mensagens Enviadas:

……
……

Dados usados:
……
……
Dados produzidos:
……
……
Interações:
……
……

3. Descritor de Cenários.

Cenário # _____
Nome:
……
Descrição:
……
……
Contexto:
……
……
Passos do Cenário (Percepção / Ação / Mensagem / Atividade com as funcionalidades associadas e os dados)

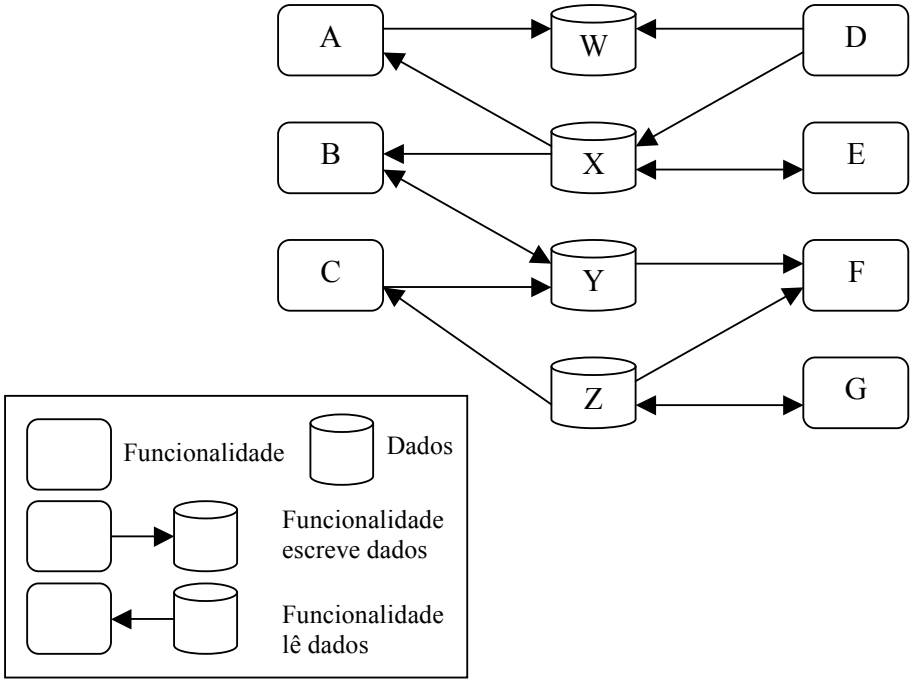
Nº Passo	Descrição Passo	Funcionalidade(s)	Dados lidos	Dados escritos
.....	.....	.....	.....	.....
.....	.....	.....	.....	.....
.....	.....	.....	.....	.....

Variações dos principais casos:

.....	.....
.....	.....

4. Diagrama de Ligação de Dados

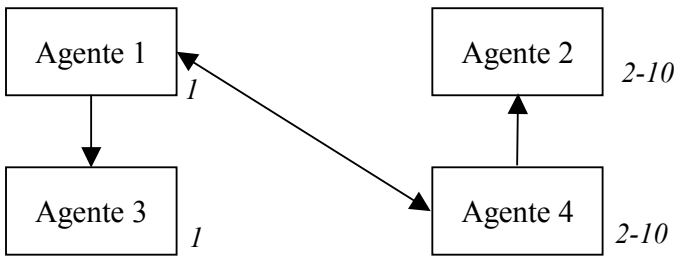
Utilizar os descritores de funcionalidade para auxiliar na identificação de possíveis agrupamentos de agentes e conseqüentemente na elaboração deste diagrama.



5. Diagrama de relacionamento do agente

Detalhar:

a. tipos (classes) de agentes
b. número esperado de cada tipo de agente
c. a interação entre agentes de diferentes tipos



→ Comunicação
2-10 : Número de agentes deste tipo

6. Descritor de Percepção/ Ação/ Mensagem

Descritor de Percepção

Nome:
.....
Descrição:
.....
.....
Origem: (sensor e outra origem)
.....
Estrutura: (qual a forma que o dado chegam)
.....
Eventos importantes da percepção:
.....
Dados auxiliares: (outros dados da percepção necessários para gerar eventos, ex: eventos prévios)
.....

Descritor de Ação
Nome:
.....
Descrição:
.....
Parâmetros: (influenciam a performance da ação, ex: velocidade para mover a ação)
.....
Tempo: (instantâneo / duração)
.....
Detecção falha: (quando pode falhar e como você saberia)
.....
Mudanças parciais:
.....
Falhas:
.....

Descritor de Mensagem
Nome:
.....
Descrição:
.....
.....
Informação transportada:
.....
.....
Protocolo / conversação pertence a:
.....
.....

Agente origem:
....……....……
....……....……
Agente destino:
....……....……
....……....……
Finalidade: (disparar resposta, atualizar informação, etc)
....……....……
....……....……

7. Descritor de Tipo de Agente

Descrição:
....……....……
....……....……
Objetivos:
....……....……
....……....……
Número de Instâncias:
....……....……
Capacidades necessárias:
....……....……
....……....……
....……....……
Ciclo de vida do agente:
....……....……
....……....……
Inicialização:
....……....……
Término:
....……....……
Percepção:
....……....……
....……....……
Ação:
....……....……
....……....……
Mensagens Enviadas:
....……....……
....……....……
....……....……
Mensagens Recebidas:
....……....……
....……....……
....……....……

Interações:

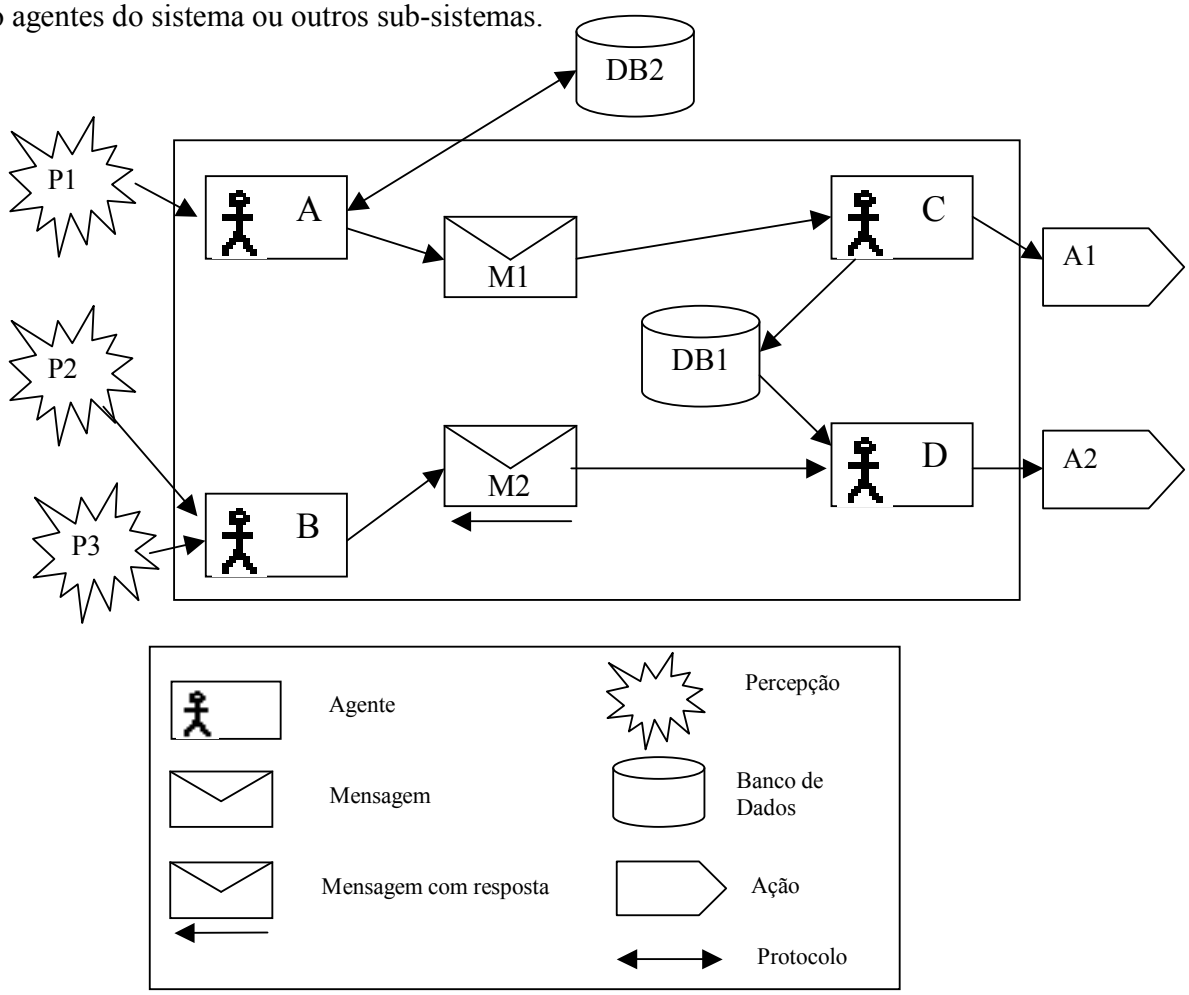
.....

.....

Informação Lida	Informação Escrita
.....	.....
.....	.....
.....	.....

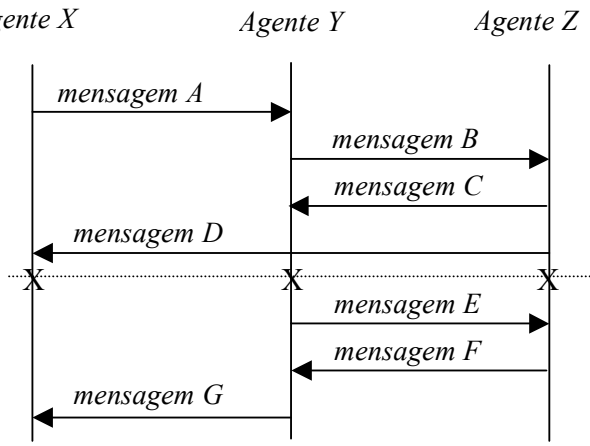
8. Diagrama de Revisão do Sistema

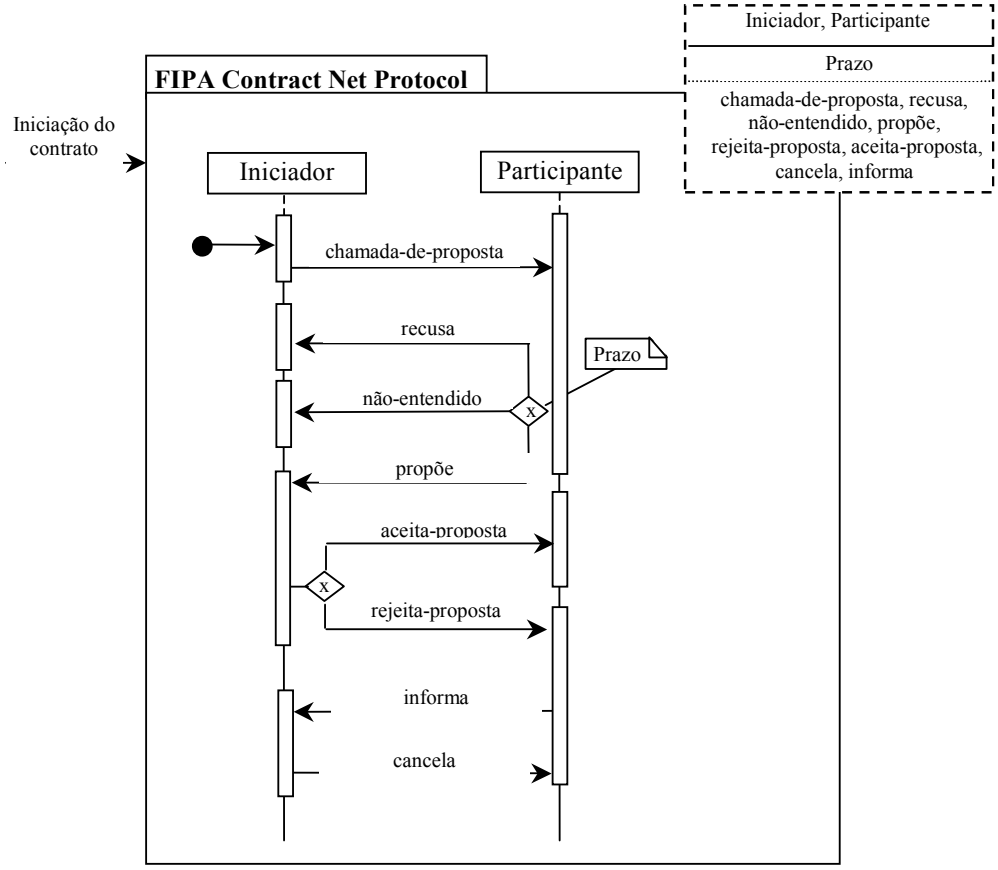
O diagrama de revisão do sistema mostra os agentes, as percepções a que os agentes respondem, as mensagens (ou protocolos/conversação) entre agentes, ações de agentes e uso de arquivos de dados compartilhados ou externos. Ele também pode mostrar as interfaces com as partes que não são agentes do sistema ou outros sub-sistemas.



9. Diagramas de Interação

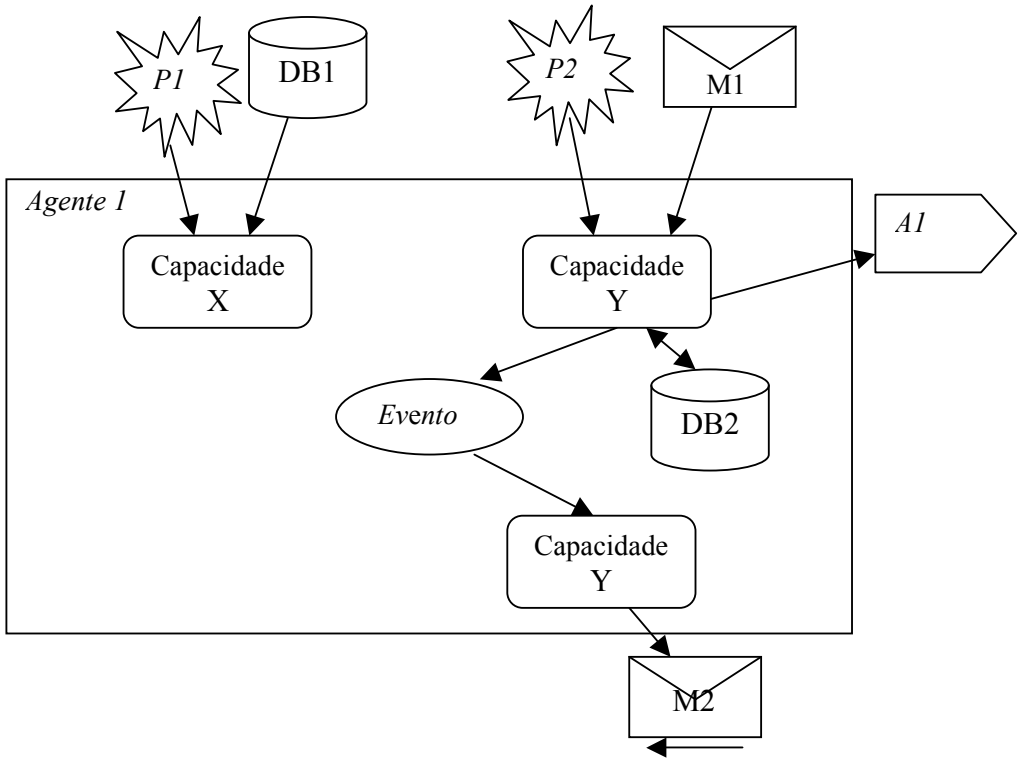
Para desenvolver os diagramas de interação são utilizados os cenários e as mensagens sobre quais funcionalidades pertencem a quais agentes. São identificados os agentes e como elas fluem entre os agentes. Abaixo estão os modelos do diagrama de interação e do protocolo de interação, respectivamente. O protocolo de interação utilizado é o FIPA Contract Net Protocol.

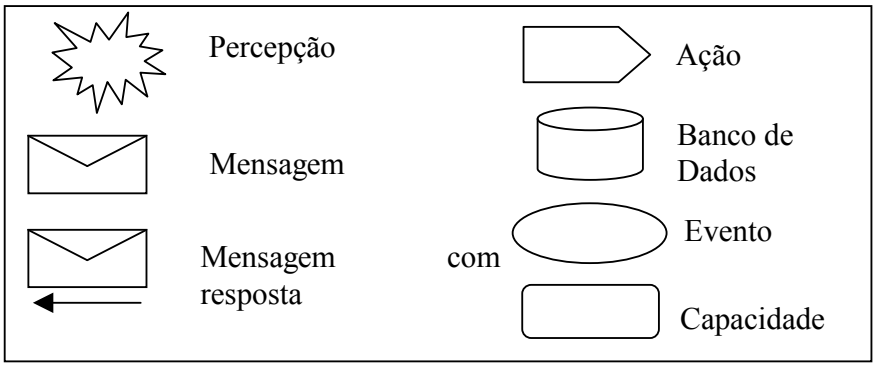




10. Diagramas de Revisão do Agente

Cada agente possui uma ou várias capacidades. As capacidades são blocos construtivos que podem ser reusados pelos agentes. Para aumentar a funcionalidade da capacidade, esta pode ser incorporada a outra capacidade. Para cada agente mostrado o diagrama de revisão do agente indica as capacidades do agente e o evento gerado e manipulado de cada capacidade. Ele também deveria mostrar dados importantes daquele agente e quais capacidades usam cada origem de dados.



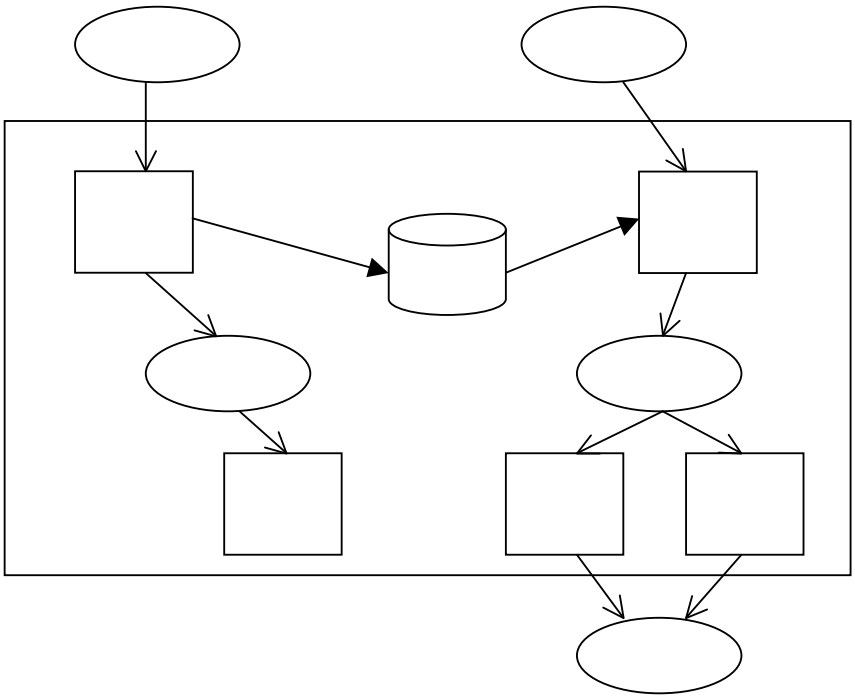


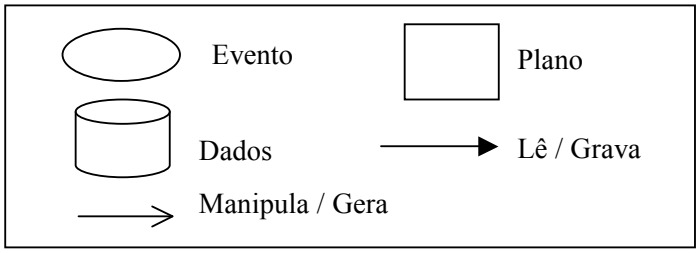
11. Diagrama de Revisão da Capacidade (Dado-Evento-Plano DEP)

Solução de projeto detalhado. Ilustra os relacionamentos em cada capacidade.

Documentação:

.....
.....
.....
.....
.....





12. Descritor de Tipo de Plano
Descrição de projeto detalhado do tipo de plano.

Documentação:
.....
.....
Tipo de evento manipulado:
.....
Contexto:
.....
Fracasso: (quando/como o plano deve fracassar)
.....
.....
Método de Falha: (o que é preciso fazer se o plano falhar, etc)
.....
.....
Sub-tarefas geradas:
.....
.....
Eventos da mensagem enviada:
.....
.....
Novos objetivos gerados:
.....
.....
Procedimentos:
.....
.....

Informação lida	Informação escrita
……	……
……	……

13. Descritor de Tipo de Evento

Descrição de projeto detalhado do tipo de evento.

Nome:
.....
Finalidade:
.....
.....
Argumentos:
.....
.....

Cobertura de Segurança: Se existe sempre um plano para manipular este evento.
.....
.....
Sobreposição: Se existem várias situações que podem ter múltiplas respostas para este evento. Se sim, explicar.
.....
.....

14. Crenças e Estruturas de Dados

Descrição de projeto detalhado da estrutura de crenças.
Deve ser uma estrutura especializada, dependendo da plataforma de implementação ou uma estrutura de dados de linguagem de programação, ou um banco de dados. Os dados devem ser projetados e especificados usando paradigmas e diagramas apropriados; por exemplo, diagramas de classes ou tabelas de banco de dados.
.....
.....
.....

15. Dicionário de Projetos

Dicionário de projetos para _____
projeto/aplicação / módulo
Lista de nomes de cada tipo.
Agentes:
.....
.....
Capacidades:
.....
.....
Planos:
.....
.....
Ações:
.....
.....
Eventos:
.....
.....
Mensagens:
.....
.....

Percepções:

....……....……

....……....……

Campos de Dados:

....……....……

....……....……

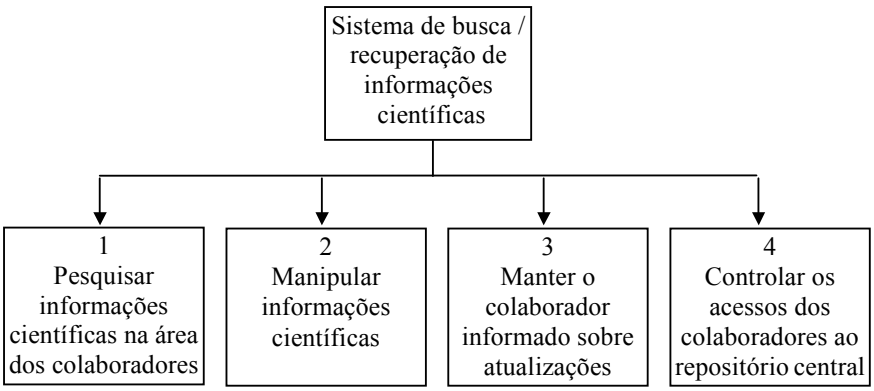
ANEXO B

ESTUDO DE CASO

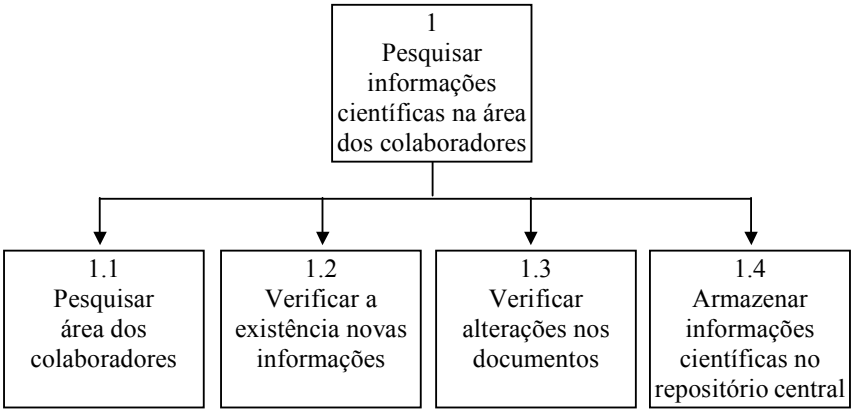
Metodologia MaSE

Fase de Análise

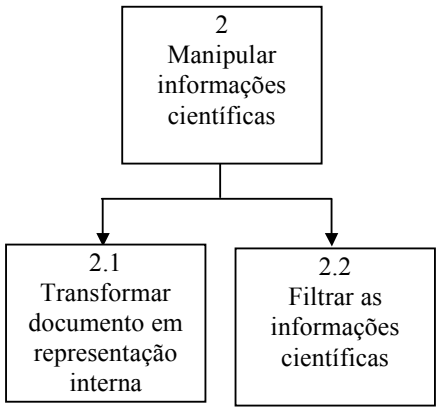
1. Diagrama Hierárquico de Objetivos da Metodologia Unificada.



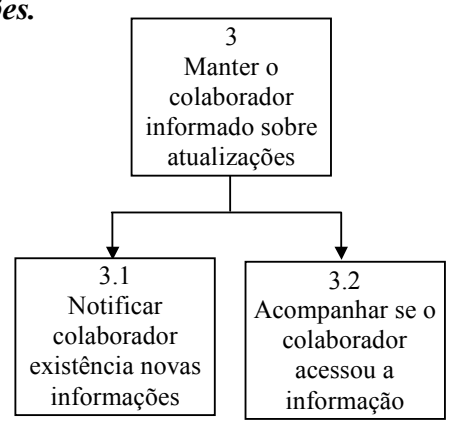
2. Diagrama Hierárquico de Objetivos, referente ao objetivo *pesquisar informações científicas na área dos colaboradores*.



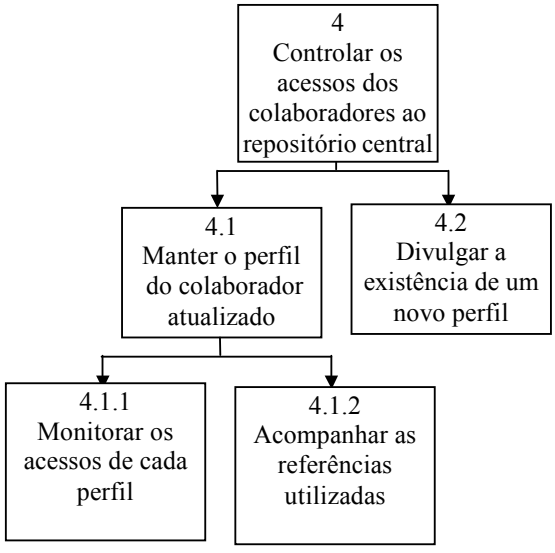
3. Diagrama Hierárquico de Objetivos, referente ao objetivo *manipular as informações científicas*.



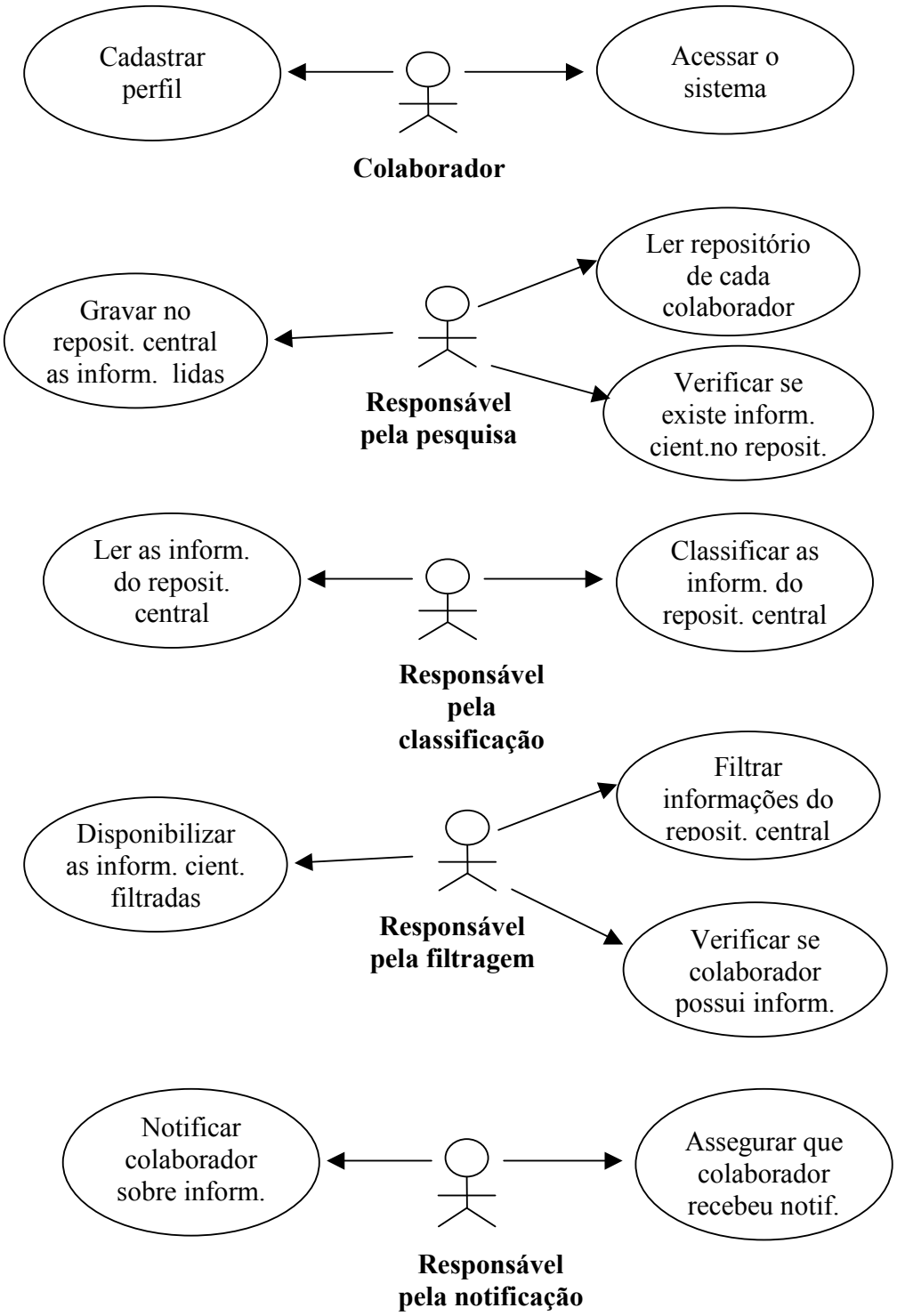
4. Diagrama Hierárquico de Objetivos, referente ao objetivo *manter o colaborador informado sobre atualizações.*

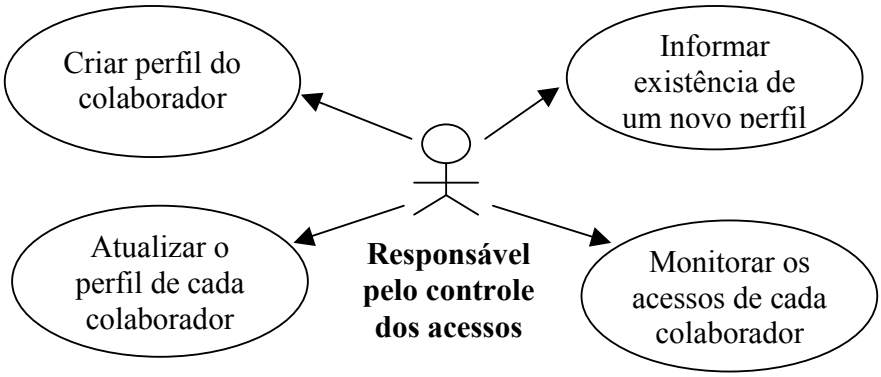


5. Diagrama Hierárquico de Objetivos, referente ao objetivo *controlar os acessos dos colaboradores no repositório central.*

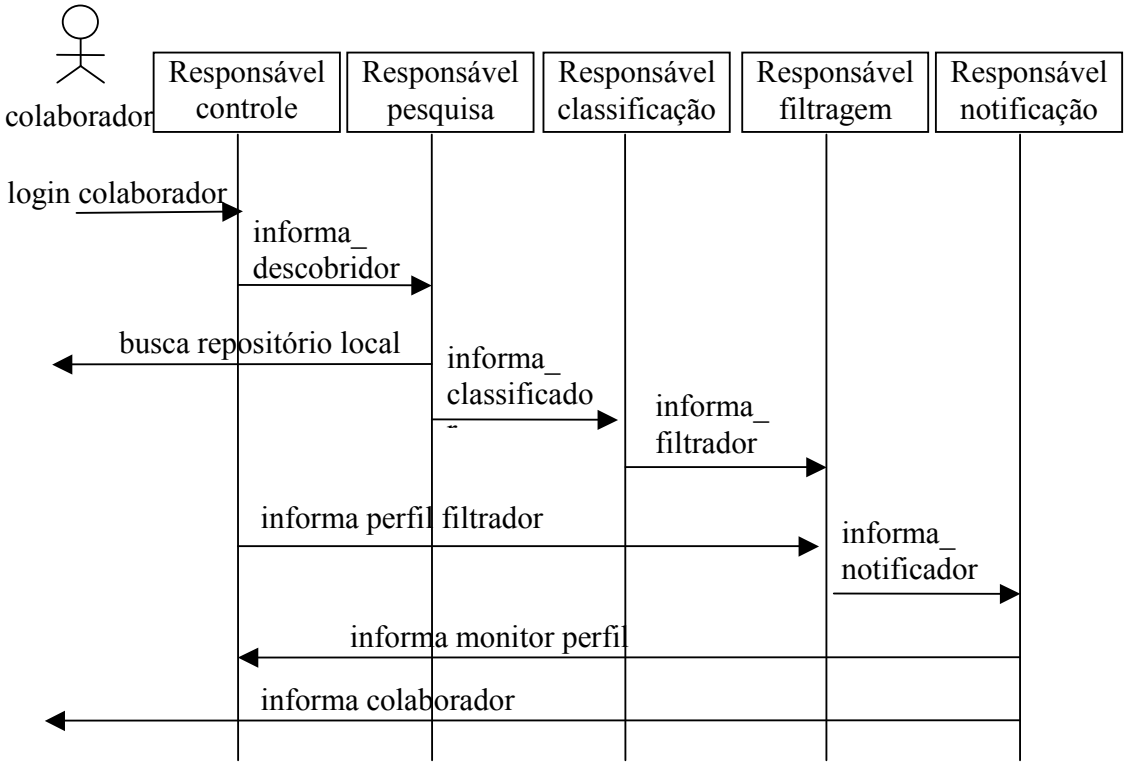


6. Casos de Uso do Sistema SISBIC

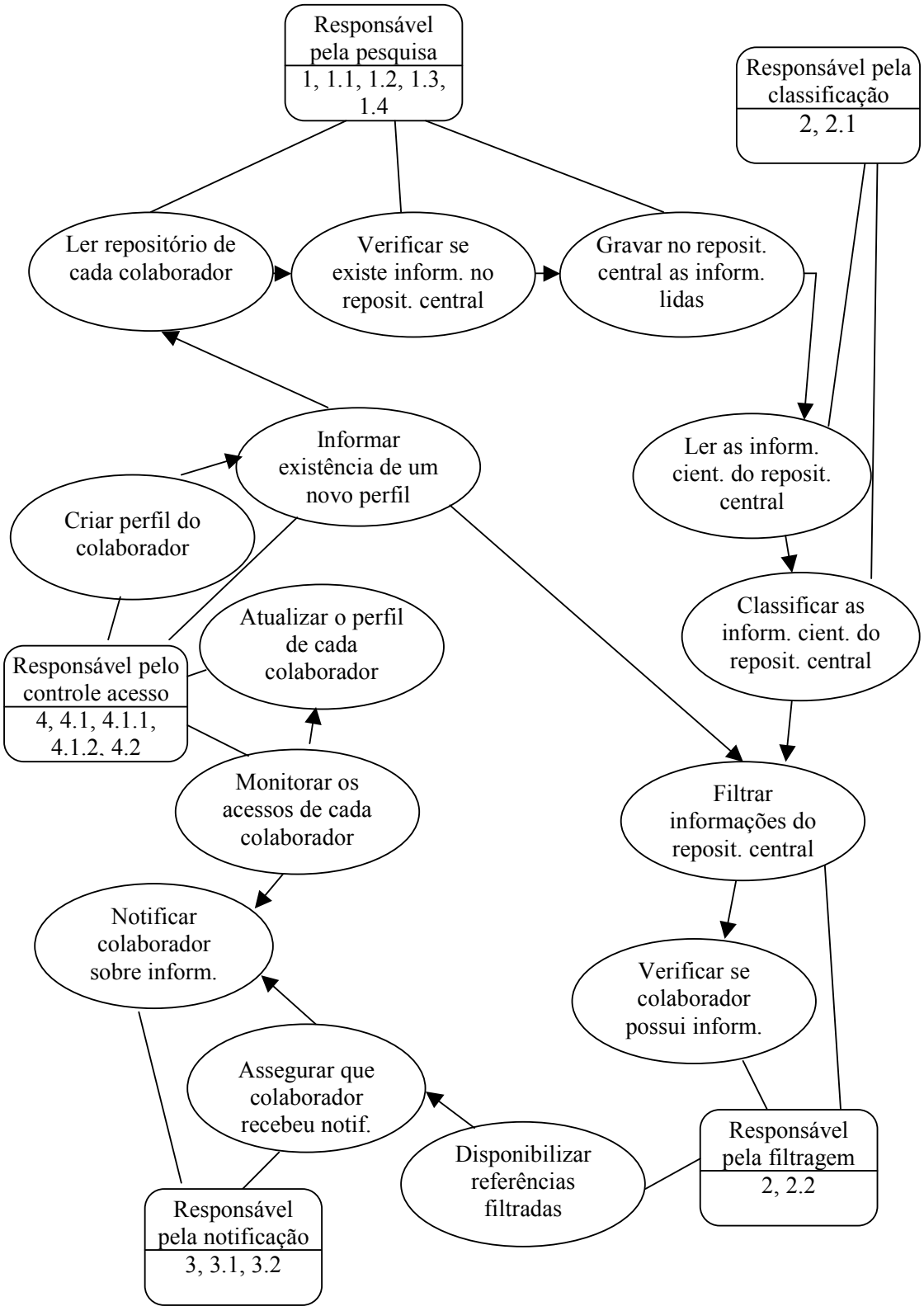




7. Diagrama de Seqüência do Sistema SISBIC.

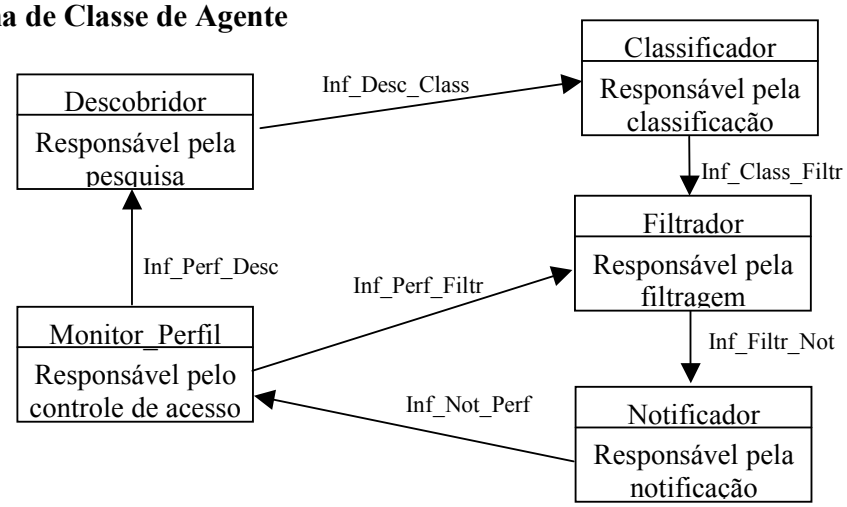


8. Modelo do Papel do Sistema SISBIC

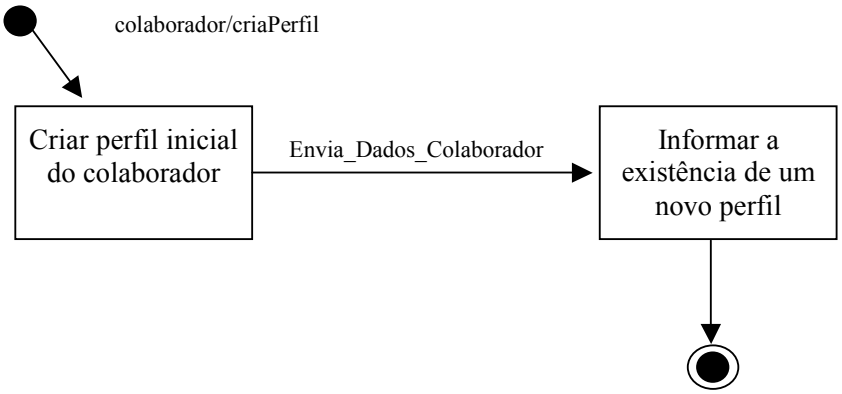


Fase de Projeto

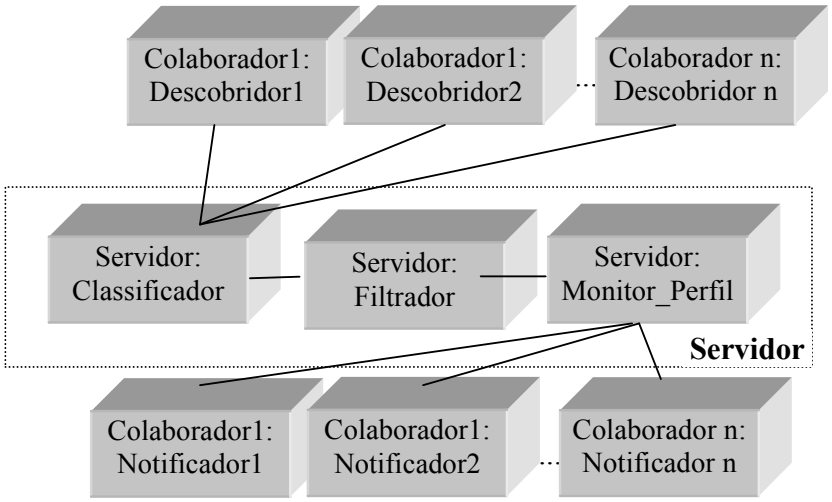
9. Diagrama de Classe de Agente



10. Diagrama de Comunicação de Classe - Iniciador.



11. Diagrama de Distribuição



Metodologia Prometheus

Fase de Especificação do Sistema

1. Objetivos do Sistema SISBIC

Objetivos:

1. Pesquisar informações científicas na área dos colaboradores.
2. Verificar a existência de novas informações científicas.
3. Verificar as alterações nos documentos armazenados no repositório central.
4. Armazenar as informações científicas no repositório central .
5. Manipular informações científicas.
6. Transformar o documento em representação interna.
7. Filtrar as informações científicas.
8. Manter o colaborador informado sobre as atualizações.
9. Notificar o colaborador sobre a existência de novas informações.
10. Acompanhar se o colaborador acessou a informação.
11. Controlar os acessos dos colaboradores ao repositório central.
12. Manter o perfil do colaborador atualizado.
13. Monitorar os acessos de cada perfil.
14. Acompanhar as referências utilizadas.
15. Divulgar a existência de um novo perfil.

2. Funcionalidades do Sistema SISBIC

Funcionalidades	Objetivos Associados
Responsável pela Pesquisa Descobridor	1, 2, 3, 4
Responsável pela Classificação Classificador	5,6
Responsável pela Filtragem Filtrador	7
Responsável pela Notificação Notificador	8, 9, 10
Responsável pelo Controle dos Acessos Monitor_Perfil	11, 12, 13, 14, 15

3. Descritor de Funcionalidade - Descobridor

Nome: Descobridor
Descrição: Pesquisar informações científicas nas áreas dos colaboradores.
Percepções: Existência de nova informação científica no repositório do colaborador InformaPerfil_Descobridor (mensagem)
Ações: Coletar as informações científicas no repositório de cada colaborador Armazenar as informações científicas coletadas no repositório central
Mensagens Enviadas: InformaDescobridor_Classificador
Dados lidos: referencias_bibli_colaborador_db
Dados gerados: referencias_bibli_db
Interações: Classificador, Monitor_Perfil

4. Descritor de Funcionalidade - Classificador

Nome: Classificador
Descrição: Classifica as informações científicas coletadas a partir do repositório de cada colaborador.
Percepções: Existência de novas informações científicas no repositório central InformaDescobridor_Classificador (mensagem)
Ações: Classificar as informações científicas do repositório central de acordo com os critérios pré-estabelecidos
Mensagens Enviadas: InformaClassificador_Filtrador
Dados lidos: referencias_bibli_db
Dados gerados: referencias_bibli_db
Interações: Descobridor, Filtrador

5. Descritor de Funcionalidade - Filtrador

Nome: Filtrador
Descrição: Filtra as informações científicas do repositório central de acordo com o perfil do colaborador
Percepções: Existência de novas referências no repositório central Existência de perfil novo ou atualizado InformaClassificador_Filtr (mensagem) InformaPerfil_Filtr (mensagem)
Ações: Ler as informações científicas do repositório central e filtrá-las de acordo com o perfil de do colaborador
Mensagens Enviadas: InformaFiltrador_Notif
Dados lidos: referencias_bibli_db
Dados gerados: referencias_bibli_perfil_db
Interações: Classificador, Notificador, Monitor_Perfil

6. Descritor de Funcionalidade - Notificador

Nome: Notificador
Descrição: Notifica ao colaborador a existência de uma nova informação científica
Percepções: Existência de nova informação científica armazenada no repositório central que atende ao perfil do colaborador InformaFiltrador_Notif (mensagem)
Ações: Informar ao colaborador que existe uma informação científica que atende seu perfil
Mensagens Enviadas: InformaNotificador_Perfil InformaColaborador
Dados lidos: Referencias_bibli_perfil_db
Dados gerados: Notificacao_db
Interações: Filtrador, Monitor_Perfil, Colaborador

7. Descritor de Funcionalidade – Monitor_Perfil

Nome: Monitor_Perfil
Descrição: Avalia e monitora o perfil de cada colaborador
Percepções: Existência de um novo colaborador no sistema Alteração do perfil de um colaborador Notificação sobre nova informação enviada para o colaborador LoginColaborador (mensagem) InformaNotificador_Perf (mensagem)
Ações: Verificar os acessos feitos por cada colaborador Atualizar o perfil de cada colaborador de acordo com os acessos
Mensagens Enviadas: InformaPerfil_Desc InformaPerfil_Filtr
Dados lidos: Perfil_db, notificacao_db
Dados gerados: Perfil_db
Interações: Descobridor, Notificador, Filtrador

8. Descritor de Cenários parcial do SISBIC

Cenário: Pesquisar informações científicas na área do colaborador

Nome: Pesquisar informações científicas na área do colaborador

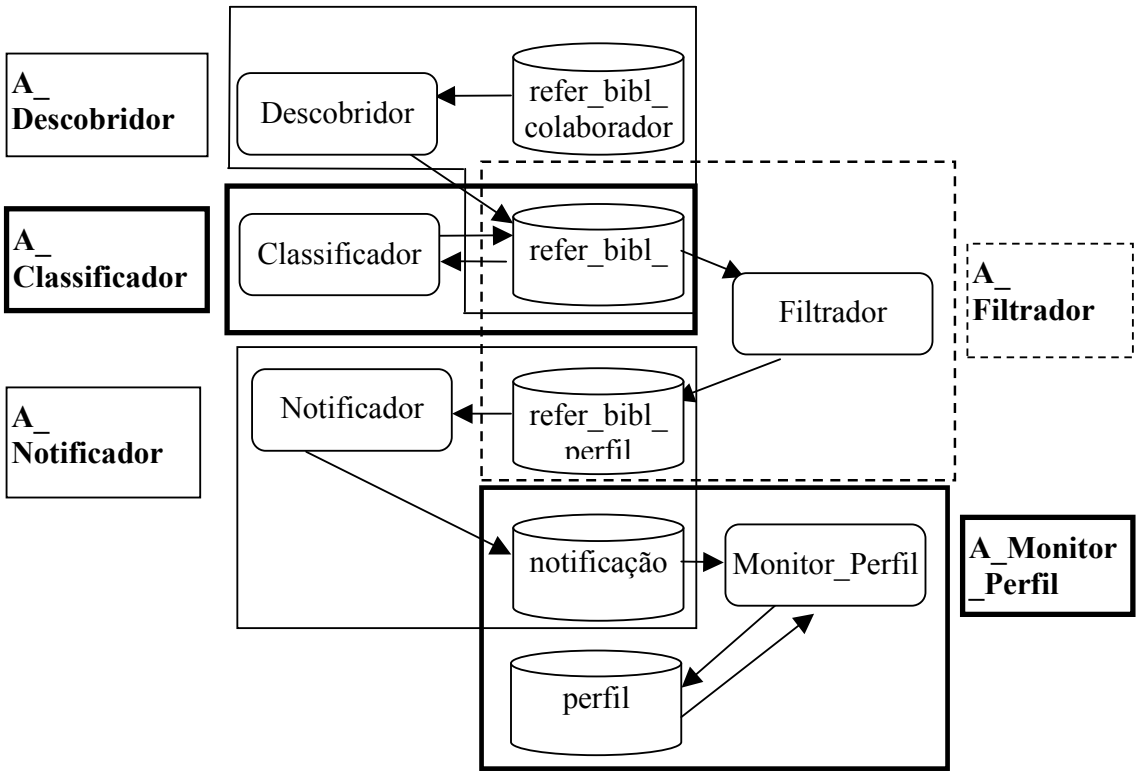
Descrição: Após o colaborador entra no sistema, será feito um rastreamento no seu repositório particular a procura de novas informações bibliográficas. Essas novas informações serão coletadas e armazenadas no repositório central.
Contexto: Assumir que o colaborador possui um repositório particular onde armazena as informações científicas que poderão ser compartilhadas com os demais colaboradores (usuários do sistema).

Passos do Cenário:

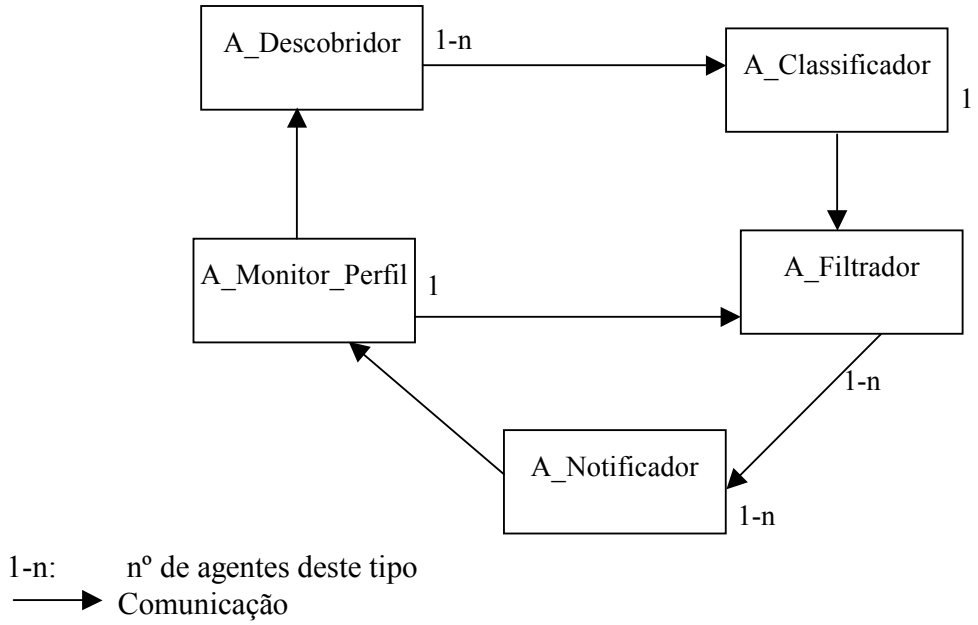
No. Passo	Descrição	Funcionalidades	Dados Lidos	Dados gerados
1	Informa_Descrobridor	Monitor_Perfil	-	-
2	Ler repositório de cada colaborador	Descobridor	referencias_bibli _colaborador_db	-
3	Ação: Gravar no repositório central as novas informações bibliográficas lidas do repositório de cada colaborador	Descobridor		referencias_bibli_db

Fase de Projeto Arquitetural

9. Diagrama de Ligação de Dados



10. Diagrama de Relacionamento de Agentes do SISBIC



11. Descritor da Percepção "Colaborador atualiza repositório" do agente A_Monitor_Perfil do SISBIC

Nome: Existe_informacao_colaborador
Descrição: Existência de nova informação científica no repositório do colaborador
*Origem:*A_Monitor_Perfil
Estrutura: identificação_colaborador, nome_arquivo_atualizado
Eventos importantes da percepção: informa_perfil_desc
Dados auxiliares: não existe

12. Descritor da Ação "Salvar informação científica no repositório central" do agente A_Descobridor do SISBIC.

Nome: Salva_arquivo_rep
Descrição: Armazenar as informações científicas coletadas no repositório do colaborador no repositório central
Parâmetros: Tamanho do arquivo
Tempo: tempo necessário para salvar o arquivo
Detenção falha: Perder a conexão com o banco de dados (ref_bibli_db)
Mudanças parciais: não diagnosticadas
Falha: repositório central não está disponível

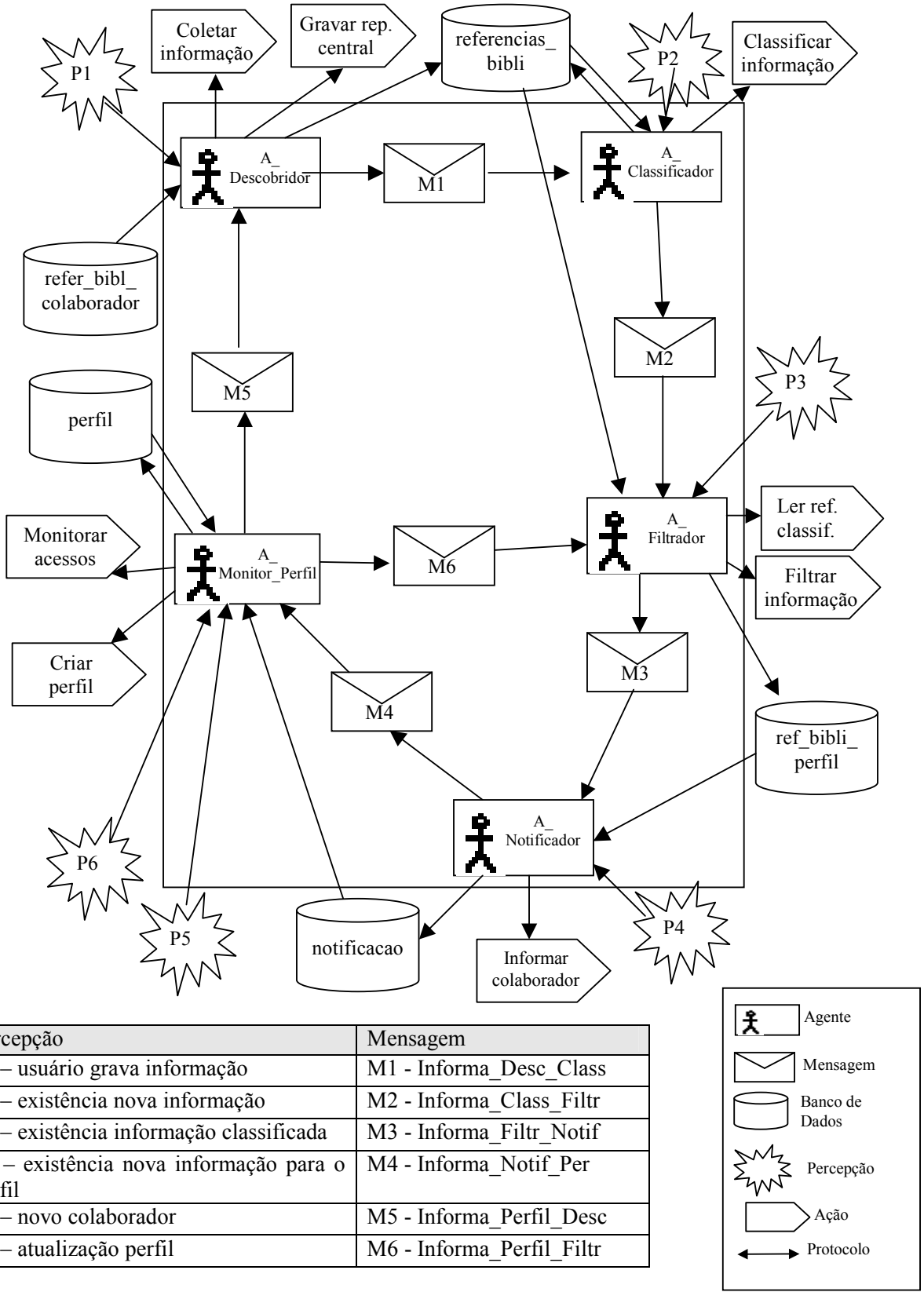
13. Descritor de Mensagem "Informa_Descobridor_Classificador" do SISBIC

Nome: Informa_Descobridor_Classificador
Descrição: Informa ao classificador que existe uma nova informação científica armazenada no repositório central
Informação transportada: Nome do arquivo armazenado no repositório central
Protocolo / conversação pertence a:A_Descobridor
Agente origem: A_Descobridor
Agente destino: A_Classificador
Finalidade: Manter arquivos do repositório central atualizados

14. Descritor do Agente A_Descobridor

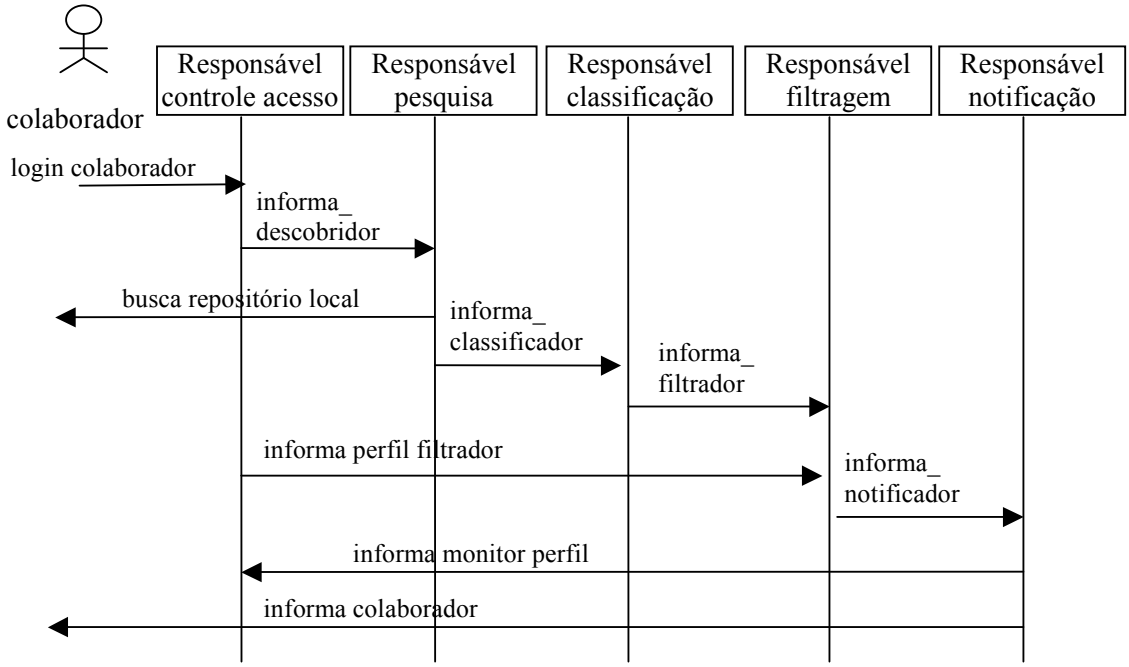
Nome: A_Descobridor
Descrição: Busca informações científicas (referência bibliográfica) na área de cada colaborador
Objetivos: Pesquisar áreas dos colaboradores Verificar a existência de novas informações científicas Verificar as alterações nos documentos armazenados no repositório central Armazenar as informações científicas, coletadas nas áreas dos colaboradores, no repositório central
Instâncias: n, onde n é igual ao número de colaboradores.
Capacidades necessárias: Leitura no repositório de cada colaborador Gravação no repositório central as referências bibliográficas lidas no repositório de cada colaborador
Ciclo de vida do agente: Inicia-se quando o colaborador entra no sistema e é destruído quando o colaborador sai do sistema ou fica muito tempo inativo.
Inicialização: Mensagem recebida informando que o agente entrou no sistema. Abre a conexão com o banco de dados (repositório) do colaborador.
Término: Fecha a conexão com o banco de dados do colaborador
Percepção: Existência de nova informação científica no repositório do colaborador InformaPerfil_Descobridor (mensagem)
Ação: Coletar as informações científicas no repositório de cada colaborador Armazenar as informações científicas coletadas no repositório central
MensagensRecebidas: Informa_Perfil_Desc
Mensagens Enviadas: Informa_Descobridor_Class
Interações: A_Classificador, A_Monitor_Perfil
Dados lidos: referencias_bibli_colaborador_db
Dados gravados: referencias_bibli_db

15. Diagrama de Revisão do Sistema SISBIC



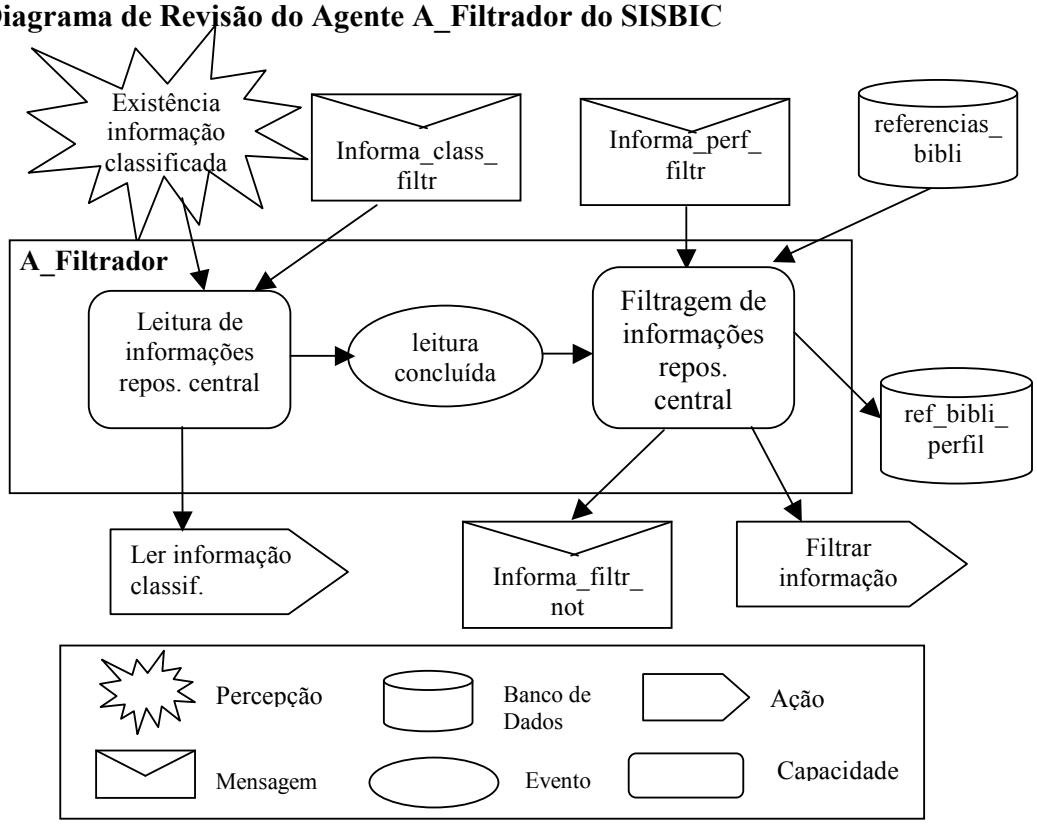
Percepção	Mensagem
P1 – usuário grava informação	M1 - Informa_Desc_Class
P2 – existência nova informação	M2 - Informa_Class_Filtr
P3 – existência informação classificada	M3 - Informa_Filtr_Notif
P4 – existência nova informação para o perfil	M4 - Informa_Notif_Per
P5 – novo colaborador	M5 - Informa_Perfil_Desc
P6 – atualização perfil	M6 - Informa_Perfil_Filtr

16. Diagrama de Seqüência do SISBIC

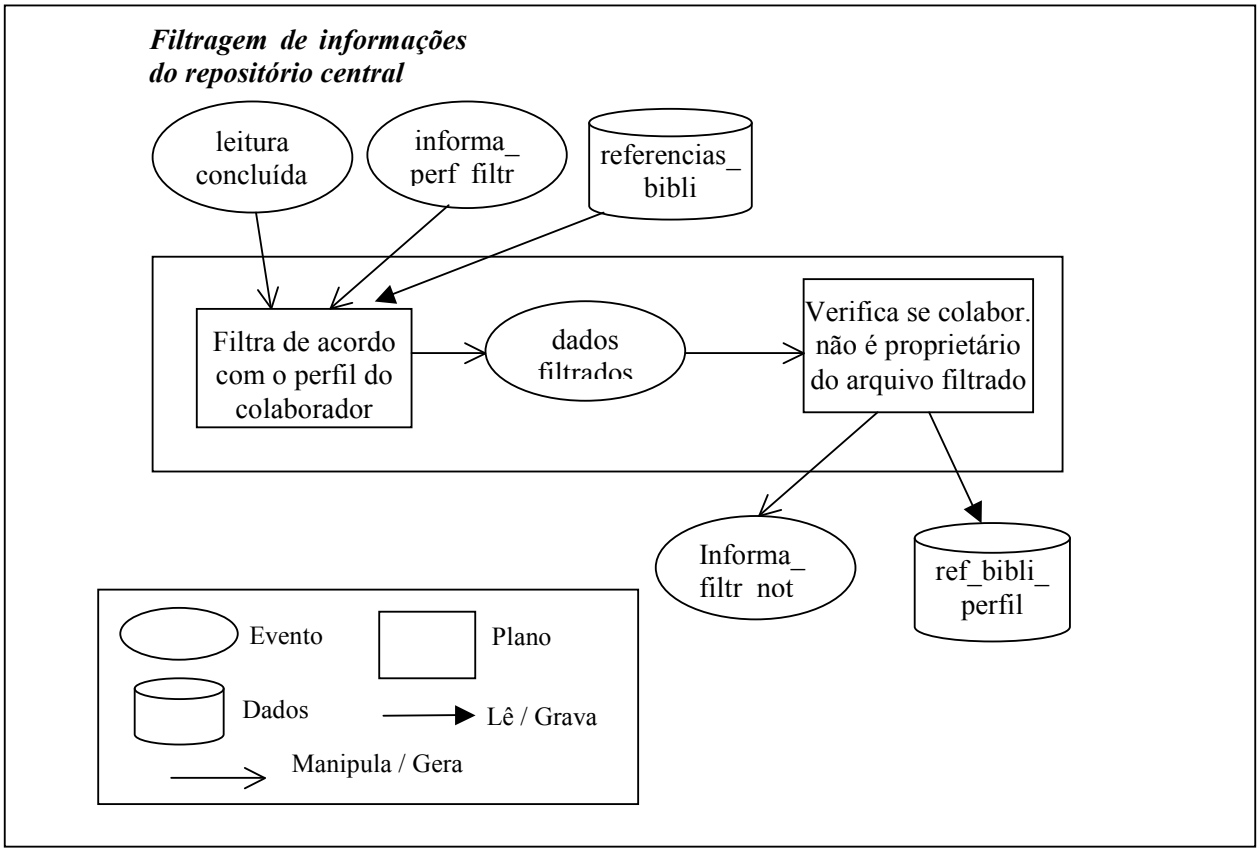


Fase de Projeto Detalhado

17. Diagrama de Revisão do Agente A_Filtrador do SISBIC



18. Diagrama de Revisão da Capacidade "Filtragem de Informações" do SISBIC



19. Descritor de Tipo de Plano "Filtrar informações do repositório central" do SISBIC

Documentação: Filtrar informações do repositório central de acordo com o perfil do colaborador
Tipo de Evento Manipulado: leitura concluída / dados filtrados e mensagem informa_filtrador_not
Contexto: Este plano deve ser executado assim que o agente A_Classificador classificar as informações no repositório central e que o agente A_Monitor_Perfil envia o perfil do colaborador.
Fracasso: (quando ou como) Repositório central vazio ou perda de conexão
Método de Falha: (o que fazer se o plano falhar) Aguardar e tentar novamente
Sub-tarefas geradas: Consultar banco de dados sobre as informações que atendam o perfil do colaborador Gravar no banco de dados temporário o resultado da consulta
Eventos de mensagens enviadas: informa_filtrador_not
Novos objetivos gerados: não diagnosticados
Procedimentos: Receber informação sobre o perfil do colaborador Fazer uma consulta no banco de dados referencias_bibli_db, com as informações que atendam o perfil do colaborador Gravar no banco de dados temporário referencias_bibli_perfil_db o resultado da consulta
Informação lida: referencias_bibli_db
Informação gravada: referencias_bibl_perfil_db

20. Descritor de Tipo de Evento "dados filtrados" do agente A_Filtrador do SISBIC

Nome: dados filtrados
Finalidade: informar que a filtragem dos dados já terminou e que pode passar para o próximo passo
Argumentos: status (filtragem realizada com sucesso)
Cobertura de Segurança (se existe sempre um plano para manipular este evento): Sim. Esse evento é garantido pelo plano "filtrar de acordo com o perfil do colaborador".
Sobreposição (Se existem várias situações que podem ter múltiplas respostas para este evento. Se sim, explicar): Não.

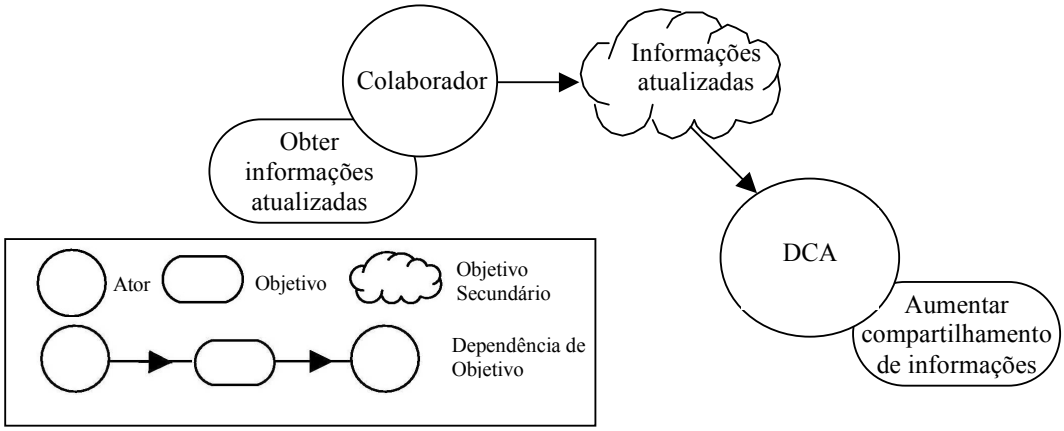
21. Dicionário de projetos parcial para o SISBIC

Agentes: A_Descobridor, A_Classificador, A_Filtrador, A_Notificador, A_Monitor_Perfil
Capacidades: - Leitura de informações no repositório central - Filtragem de informações no repositório central
Planos: Filtrar de acordo com o perfil do colaborador Verificar se o colaborador possui dado filtrado
Ações: Receber informação sobre perfil do colaborador Fazer uma consulta no banco de dados referencias_bibli_db, com as informações que atendam ao perfil do colaborador Gravar no banco de dados temporário referencias_bibli_perfil_db o resultado da consulta
Eventos: leitura concluída / dados filtrados
Mensagens: informa_filtrador_not

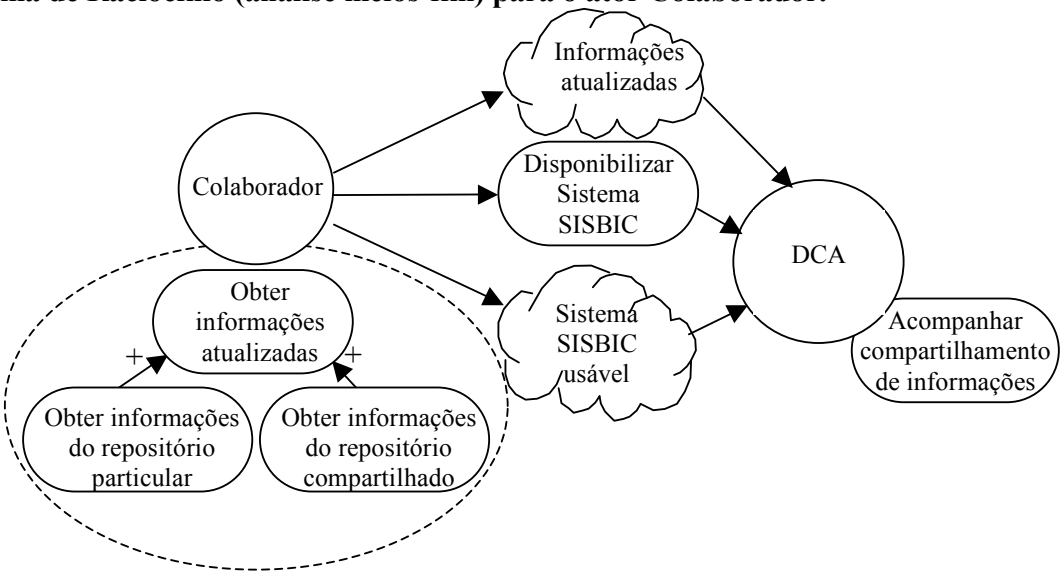
Metodologia Tropos

Fase de Requisitos Iniciais

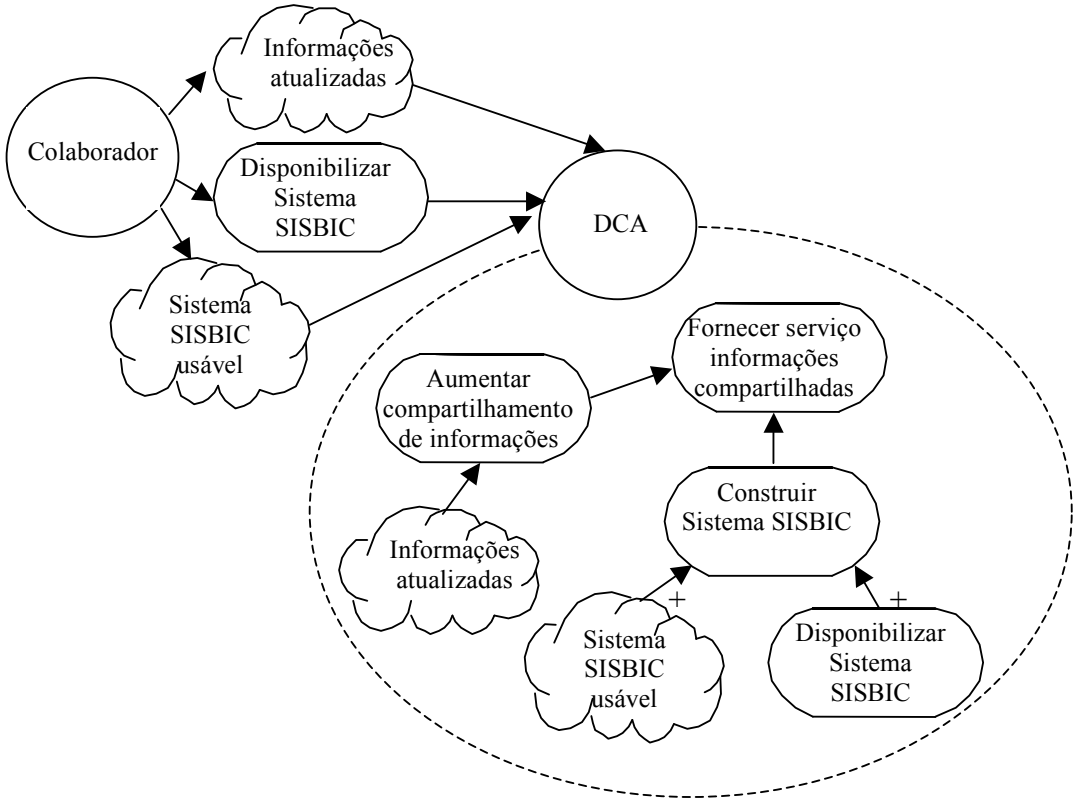
1. Diagrama de Ator: projeto SISBIC, incluindo os atores Colaborador e DCA (Departamento de Engenharia de Computação e Automação Industrial).



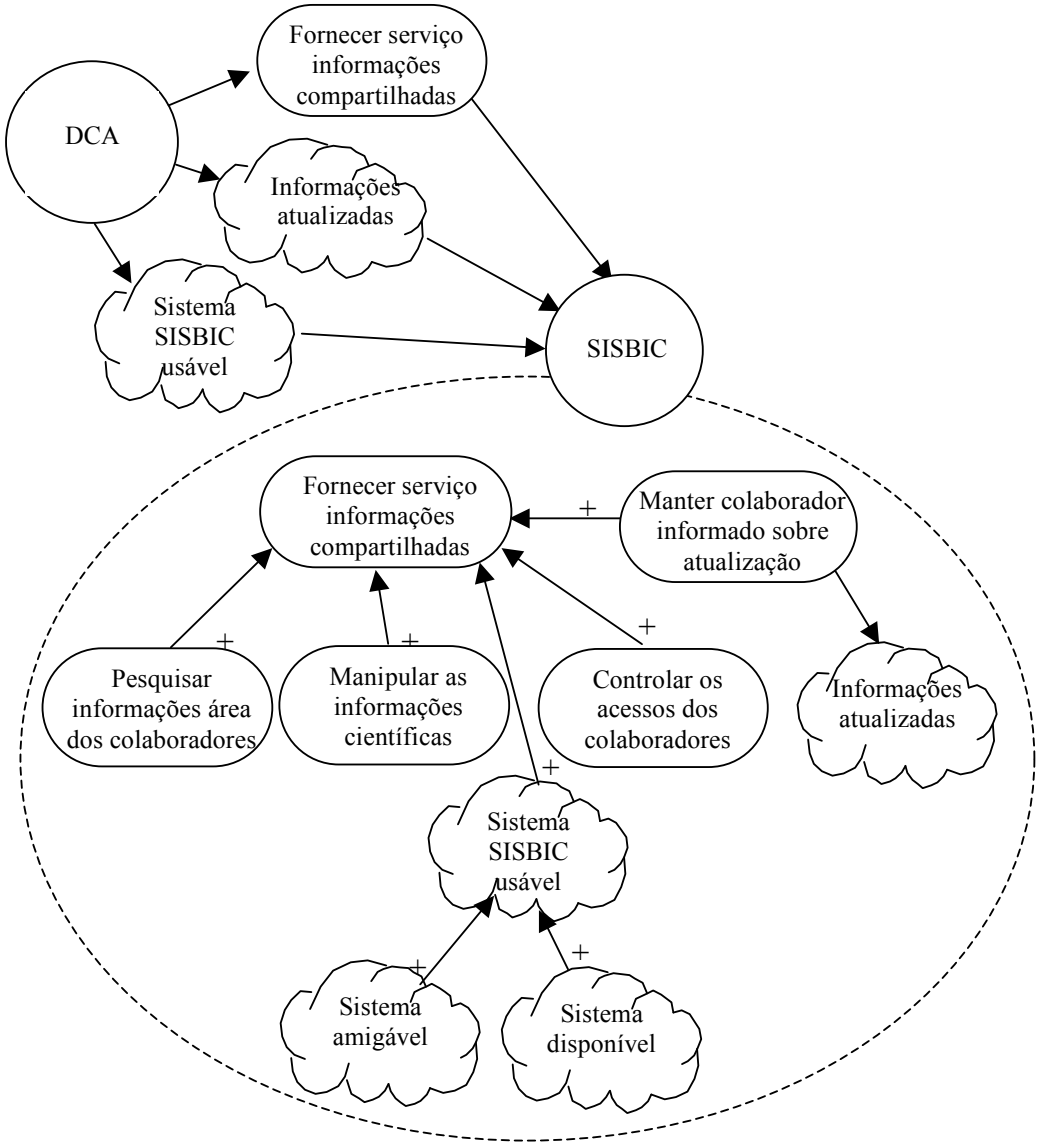
2. Diagrama de Raciocínio (análise meios-fim) para o ator Colaborador.



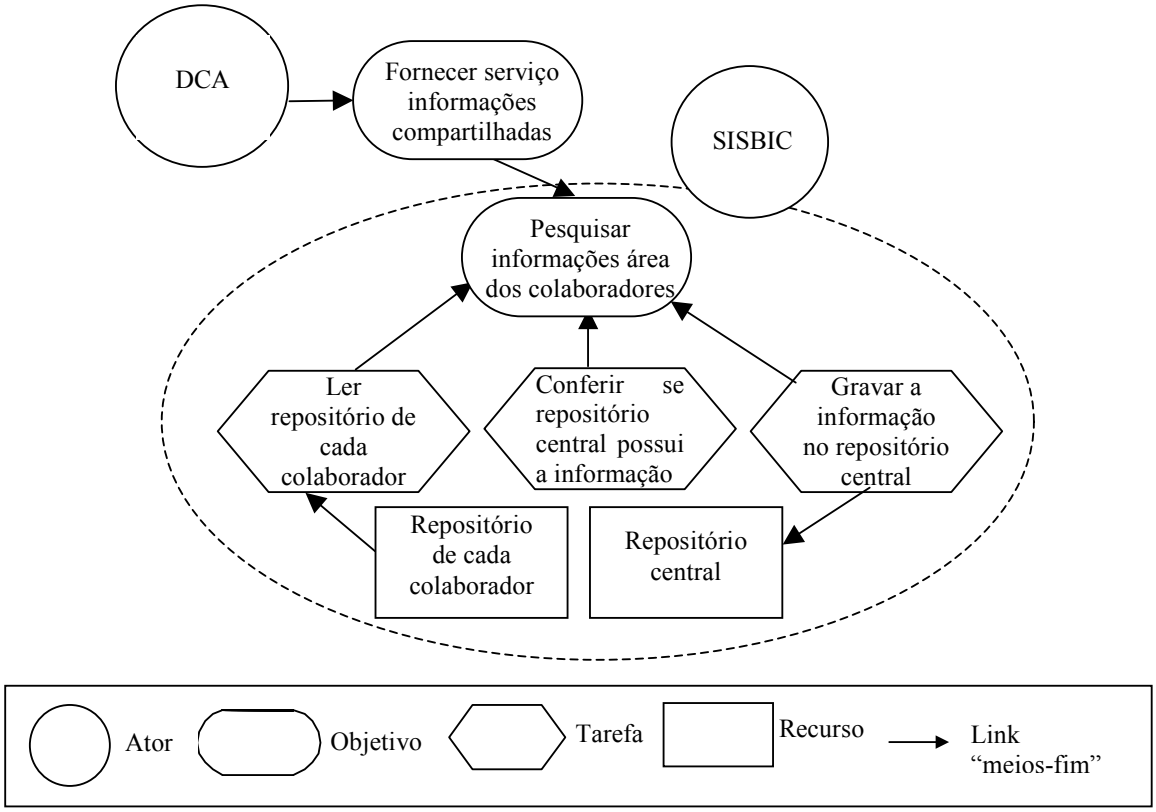
3. Diagrama de Raciocínio (análise meios-fim) para o ator DCA.



Fase de Requisitos Finais
4: Diagrama de Raciocínio (análise meios-fim) para o ator SISBIC.

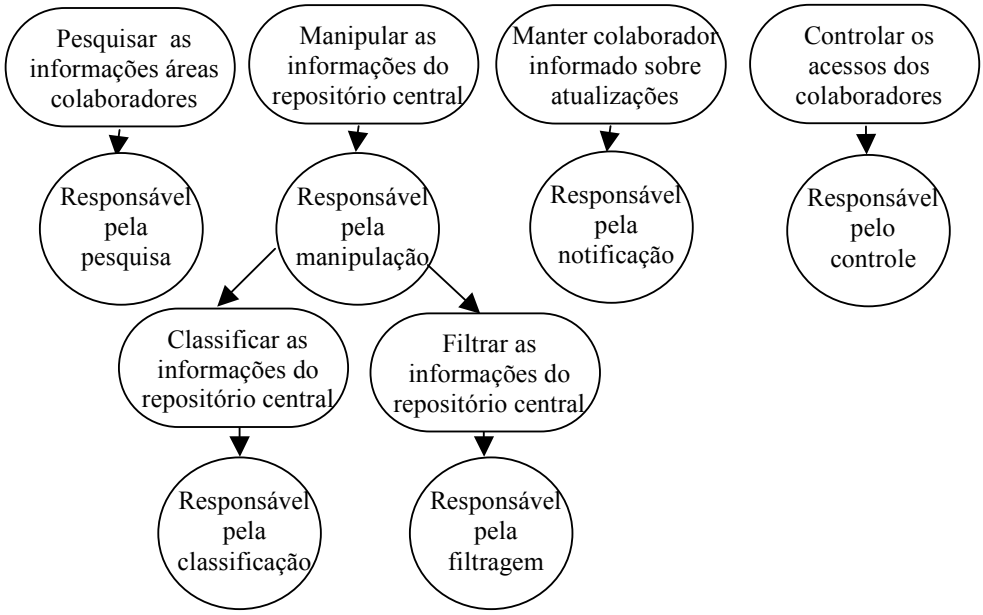


5. Diagrama de Raciocínio parcial para o objetivo "fornecer serviço de informações compartilhadas" que o ator DCA depende do ator SISBIC



Fase de Requisitos Finais

6. Decomposição de sub-atores para o Sistema SISBIC.



7. Tabela de capacidades do Sistema SISBIC.

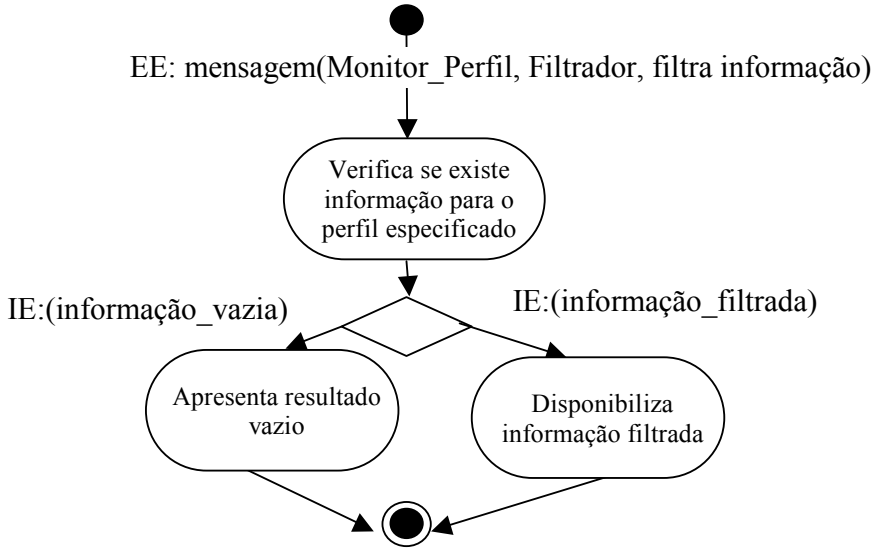
Nome	Número	Capacidades
Responsável pela pesquisa	1	- Ler repositório específico de cada colaborador
	2	- Gravar no repositório central as referências bibliográficas lidas do repositório central de cada colaborador
	3	- Verificar se a referência já não foi gravada no repositório central
Responsável pela classificação	4	- Ler as referências bibliográficas armazenadas no repositório central
	5	- Classificar as referências bibliográficas armazenadas o repositório central por categoria (área, autor, palavra-chave)
Responsável pela filtragem	6	- Filtrar as informações científicas do repositório central de acordo com o perfil de cada colaborador
	7	- Verificar se o colaborador não possui a informação filtrada
	8	- Disponibilizar as informações científicas filtradas
Responsável pela notificação	9	- Receber a mensagem sobre a existência de uma nova informação científica no repositório central que atende a um determinado perfil
	10	- Notificar o colaborador a existência de uma nova referência bibliográfica
	11	- Assegurar que o colaborador recebeu a notificação
Responsável pelo controle dos acessos	12	- Criar perfil inicial do colaborador
	13	- Informar aos demais agentes do sistema a existência de um novo perfil
	14	- Monitorar as informações científicas acessadas por cada colaborador
	15	- Atualizar o perfil de cada colaborador

8. Tabela de Tipos de Agentes do Sistema SISBIC

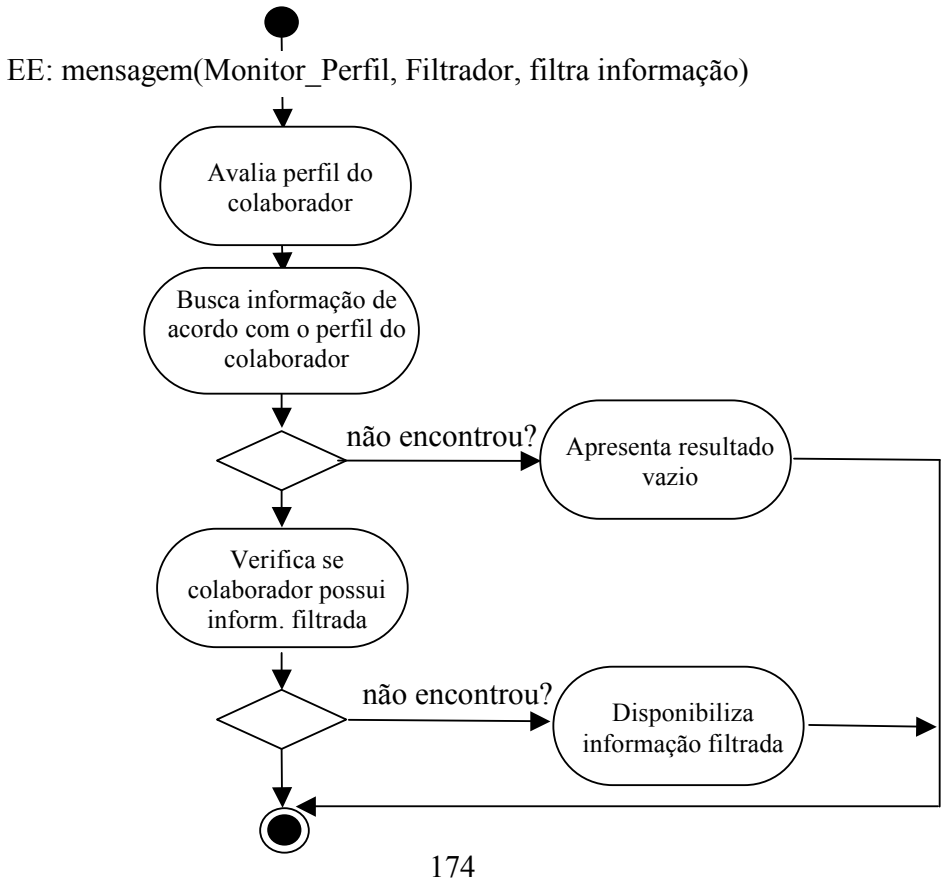
Agente	Capacidades
A_Descobridor	1, 2, 3
A_Classificador	4, 5
A_Filtrador	6, 7, 8
A_Notificador	9, 10, 11
A_Monitor_Perfil	12, 13, 14, 15

Fase de Requisitos Detalhados

9. Diagrama da Capacidade "Filtrar as informações científicas do repositório central de acordo com o perfil do colaborador".



10. Diagrama de Planos para a capacidade "Filtrar as informações científicas do repositório central de acordo com o perfil do colaborador".



11. Diagrama de Interação dos Agentes do SISBIC.