UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Uma Proposta de Arquitetura de um Ambiente de Desenvolvimento de Software Distribuído Baseada em Agentes

por

MÁRCIA CRISTINA DADALTO PASCUTTI

Dissertação submetida à avaliação, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Carlos Alberto Heuser
Orientador

Profa. Dra. Elisa Hatsue Moriya Huzita
Co-Orientadora

Porto Alegre, outubro de 2002.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Pascutti, Márcia Cristina Dadalto

Uma Proposta de Arquitetura de um Ambiente de Desenvolvimento de Software Distribuído Baseada em Agentes / por Márcia Cristina Dadalto Pascutti. – Porto Alegre: PPGC da UFRGS, 2002.

101f.:il.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2002. Orientador: Heuser, Carlos Alberto; Co-Orientadora: Huzita, Elisa Hatsue Moriya.

1. Ambientes de Desenvolvimento de Software 2. Sistemas Distribuídos. 3. Agentes de Software. 4. Desenvolvimento Cooperativo de Software.
I. Heuser, Carlos Alberto. II. Huzita, Elisa Hatsue Moriya. III. Título.

Dedico este trabalho
aos meus pais,
ao Edenilson,
à Mariana.

Agradecimentos

A Deus, que me deu muita luz e forças para conseguir realizar este trabalho.

Ao meu marido Edenilson, que soube compreender tantos momentos de ausência, sempre me incentivando, principalmente nos momentos em que o desânimo e o cansaço tentaram me achar. Sem você seria mil vezes mais difícil ...

À minha querida filha Mariana, que desde o seu nascimento teve que repartir a minha atenção e o meu tempo com o mestrado. Filha, prometo que daqui para frente terei mais tempo para me dedicar a você e acompanhar o seu crescimento ...

Aos meus pais, José e Maria, que sempre me incentivaram a seguir adiante, acreditando em mim e no meu trabalho. À minha irmã Valéria, pelas orações, pelo apoio e compreensão em todos os momentos.

Aos meus orientadores Carlos Alberto Heuser e Elisa Hatsue Moriya Huzita, pela confiança e dedicação em mim depositadas durante toda a jornada. Em especial, gostaria de agradecer à profª. Elisa por todo o tempo e paciência dispensados a mim durante as orientações.

Aos amigos, Fabrício e Robinson, que estiveram ao meu lado durante essa caminhada, sempre me incentivando. Ao César Moro, pelas contribuições e a Ana Paula por toda a ajuda dada neste trabalho. Que Deus os abençoe.

Ao Cesumar – Centro Universitário de Maringá, em especial ao seu Reitor, Prof. Wilson Matos da Silva, pela confiança no meu trabalho e pela ajuda financeira que permitiu a realização do mestrado. Ao Valdecir, Coordenador do Curso de Processamento de Dados e acima de tudo um grande amigo, pelo apoio e compreensão (e também por emprestar o *notebook* quando o meu foi roubado).

A todos aqueles que, de uma forma ou de outra, ajudaram na realização deste trabalho.

Muito obrigada!

Sumário

Lista de Abreviaturas

ACL	Agent Communication Language
ADCS	Ambiente de Desenvolvimento Cooperativo de Software
ADS	Ambiente de Desenvolvimento de Software
ANSA	Advanced Network Systems Architecture
AUML	Agent Unified Modeling Language
CAGIS PCE	Cooperative Agent in a Global Information Space – Process Centered Environment
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
CORBA	Common Object Request Broker Architecture
CSCW	Computer Supported Cooperative Work
DCOM	Distributed Component Object Model
DIMANAGER	Distributed Software Manager
DiSEN	Distributed Software Engineering Environment
ECMA	European Computer Manufacturers Association
FD	Facilitador de Diretório
FIPA	Foundation for Intelligent Physical Agents
ID	Identificador de Agente
IIOP	Internet Inter-ORB Protocol
IP	Interaction Protocol
IPL	Interaction Protocol Library
KIF	Knowledge Interchange Format
KQML	Knowledge and Query Manipulation Language
KSE	American Knowledge Sharing Efforts
LAN	Local Area Network
MDSODI	Metodologia para Desenvolvimento de Software Distribuído
MOOPP	Metodologia Orientada a Objetos para Desenvolvimento de Software para Processamento Paralelo
OMG	Object Management Group
PA	Plataforma de Agente
PROSYT	Process Support System Capable of Tolerating Deviations
PTM	Protocolo de Transporte de Mensagem
RA	Região de Agente
RMI	Remote Method Invocation
SGA	Sistema de Gerenciamento de Agente
SPA	Servidor de Página Amarela
STM	Serviço de Transporte de Mensagem
UML	Unified Modeling Language
WAN	Wide Area Network

Lista de Figuras

Lista de Tabelas

Resumo

A crescente complexidade das aplicações, a contínua evolução tecnológica e o uso cada vez mais disseminado de redes de computadores têm impulsionado os estudos referentes ao desenvolvimento de sistemas distribuídos.

Como estes sistemas não podem ser facilmente desenvolvidos com tecnologias de software tradicionais por causa dos limites destas em lidar com aspectos relacionados, por exemplo, à distribuição e interoperabilidade, a tecnologia baseada em agentes parece ser uma resposta promissora para facilitar o desenvolvimento desses sistemas, pois ela foi planejada para suportar estes aspectos, dentre outros.

Portanto, é necessário também que a arquitetura dos ambientes de desenvolvimento de software (ADS) evolua para suportar novas metodologias de desenvolvimento que ofereçam o suporte necessário à construção de softwares complexos, podendo também estar integrada a outras tecnologias como a de agentes.

Baseada nesse contexto, essa dissertação tem por objetivo apresentar a especificação de uma arquitetura de um ADS distribuído baseada em agentes (DiSEN – *Distributed Software Engineering Environment*). Esse ambiente deverá fornecer suporte ao desenvolvimento de software distribuído, podendo estar em locais geograficamente distintos e também os desenvolvedores envolvidos poderão estar trabalhando de forma cooperativa.

Na arquitetura proposta podem ser identificadas as seguintes camadas: **dinâmica**, que será responsável pelo gerenciamento da (re)configuração do ambiente em tempo de execução; **aplicação**, que terá, entre os elementos constituintes, a MDSODI (Metodologia para Desenvolvimento de Software Distribuído), que leva em consideração algumas características identificadas em sistemas distribuídos, já nas fases iniciais do projeto e o repositório para armazenamento dos dados necessários ao ambiente; e, **infra-estrutura**, que proverá suporte às tarefas de nomeação, persistência e concorrência e incorporará o canal de comunicação. Para validar o ambiente será realizada uma simulação da comunicação que pode ser necessária entre as partes constituintes do DiSEN, por meio da elaboração de diagramas de *use case* e de seqüência, conforme a notação MDSODI.

Assim, as principais contribuições desse trabalho são: (i) especificação da arquitetura de um ADS distribuído que poderá estar distribuído geograficamente; incorporará a MDSODI; proporcionará desenvolvimento distribuído; possuirá atividades executadas por agentes; (ii) os agentes identificados para o DiSEN deverão ser desenvolvidos obedecendo ao padrão FIPA (*Foundation for Intelligent Physical Agents*); (iii) a identificação de um elemento que irá oferecer apoio ao trabalho cooperativo, permitindo a integração de profissionais, agentes e artefatos.

Palavras-Chave: ambiente de desenvolvimento de software; software distribuído; agentes de software; desenvolvimento cooperativo de software.

TITLE: "ARCHITECTURE OF A DISTRIBUTED SOFTWARE DEVELOPMENT ENVIRONMENT BASED ON AGENTS"

Abstract

The growing complexity of software, the continuous technological evolution and the disseminated use of computer networks have been driving the studies regarding the development of distributed systems.

These systems cannot be easily developed with traditional software technologies, due to restrictions for working with aspects such as distribution and interoperability. The agent-based technology seems to be a promising answer to facilitate the development of these systems, as it was conceived to support these aspects, among others.

It is also necessary that the architecture of the software development environment (SDE) must improve to support new development methodologies that offer the necessary support to the construction of complex software programs and that could also be integrated with other technologies as the one of agents.

Based on that context, this dissertation has the objective of presenting the specification of architecture of a distributed SDE based on agents (DiSEN - Distributed Software Engineering Environment). This environment should support the development of distributed software, which could be in different geographic locations and involving developers who could also be able to work in a cooperative way.

In the proposed architecture the following layers can be identified: **dynamic**, which will be responsible for administering the (re)configuration of the environment in run time; **application**, which will have among the constituent elements, MDSODI (Methodology for Development of Distributed Software), which considers some characteristics identified in distributed systems, as from the project's initial phases and the repository for data storage necessary to this environment; and, **infrastructure**, which will provide support to the naming, persistence and concurrency tasks and will also incorporate the communication channel. To validate the environment there will be a simulation of the communication that could be necessary among the parts of DiSEN, through the elaboration of use case and sequence diagrams, according to the MDSODI notation.

Therefore, the main contributions of this work are: (i) specification of an architecture of a distributed SDE that might be geographically distributed; that will incorporate MDSODI; will provide for distributed development and will have activities executed by agents; (ii) the identified agents for DiSEN should be developed in compliance with the FIPA (Foundation for Intelligent Physical Agents) standards; (iii) identification of an element that will offer support to cooperative work, allowing the integration among professionals, agents and artifacts.

Keywords: software development environment; distributed software; software agents; cooperative software development.

1 Introdução

A disseminação do uso das redes de computadores e a ampla difusão da Internet têm estimulado os estudos referentes ao desenvolvimento de sistemas distribuídos. Esses sistemas consistem em uma coleção de computadores autônomos ligados por uma rede, buscando-se desta forma coordenar as atividades, de maneira eficiente, além de propiciar o compartilhamento de recursos, quer sejam de hardware ou de software [COU 2001]. Sistemas distribuídos oferecem suporte a diversos tipos de aplicações, tais como: aplicações comerciais concorrentes, aplicações que trabalham com acesso concorrente às informações em banco de dados, aplicações que envolvem compartilhamento de informações.

Em comparação aos sistemas convencionais, os sistemas distribuídos podem vir a apresentar maior eficiência devido às suas características diferenciadas, tais como: suporte aos recursos compartilhados; maior grau de abertura do sistema, permitindo a inclusão/ exclusão de serviços compartilhados sem prejudicar o desempenho do sistema; tolerância à falhas; transparência e concorrência. Por isso, no desenvolvimento de um software distribuído é importante que estes aspectos sejam considerados.

A comunidade de engenharia de software tem se preocupado, ao longo dos tempos, em desenvolver estudos a fim de oferecer técnicas/ferramentas adequadas que proporcionem o suporte necessário ao desenvolvimento de software de acordo com diferentes paradigmas. Esta trajetória teve início com a abordagem estruturada, a abordagem orientada a objetos, a abordagem baseada em componentes e, mais recentemente, a abordagem baseada em agentes. Como resultado de estudos realizados por pesquisadores, têm sido propostas metodologias que procuram oferecer o suporte necessário ao desenvolvimento estruturado [YOU 90], ao desenvolvimento orientado a objetos [JAC 99], ao desenvolvimento baseado em componentes [CHE 2001] [D'SO 99] e ao desenvolvimento baseado em agentes [WOO 2000].

A MDSODI (Metodologia para Desenvolvimento de Software Distribuído) [GRA 2000] [HUZ 99] é uma metodologia para desenvolvimento de software que leva em consideração algumas características identificadas em sistemas distribuídos, tais como: paralelismo/concorrência, comunicação, sincronização e distribuição, já nas fases iniciais do projeto, mantém as principais características do *Unified Process* [JAC 99] e utiliza também algumas características propostas na MOOPP (Metodologia Orientada a Objetos para Desenvolvimento de Software para Processamento Paralelo) [HUZ 95], conforme descrito em Pascutti [PAS 2001]. Portanto, a MDSODI será a metodologia que também fará parte da nossa proposta.

A idéia de empregar agentes de software e delegar a eles certas atividades foi introduzida por pesquisadores como Wooldridge [WOO 95] e Jennings [JEN 98] que perceberam as grandes oportunidades desta promissora área. Um agente é uma entidade autônoma que apresenta algumas características [GAR 2001], tais como: autonomia, reatividade, proatividade e mobilidade.

A computação baseada em agentes tem sido vista como a nova revolução no desenvolvimento de software. Agentes estão sendo usados em diversas aplicações que vão desde sistemas comparativamente pequenos como filtros de *e-mail's* a grandes e complexos sistemas críticos, como controle de tráfego aéreo.

Um aspecto que deve ser considerado em aplicações baseadas em agentes é a comunicação entre estas aplicações. É necessário, portanto, que sejam utilizadas

técnicas de engenharia de software que dêem o suporte necessário para o desenvolvimento de sistemas baseados em agentes, principalmente quando se tratar de sistemas distribuídos complexos, devendo tais técnicas oferecer a possibilidade de capturar adequadamente a flexibilidade e as interações de um agente. Nesse sentido têm surgido trabalhos como o que consta em Wooldridge [WOO 2000] e FIPA [FIP 2002]. A FIPA (*Foundation for Intelligent Physical Agents*) é uma organização que se propõe a estabelecer um padrão a ser seguido para o desenvolvimento de software baseado em agentes.

Um ADS tem como objetivo prover um ambiente no qual o software possa ser desenvolvido através da integração de um conjunto de ferramentas que suportam métodos de desenvolvimento, apoiados por uma estrutura que permite a comunicação e cooperação entre as ferramentas e os desenvolvedores segundo Brown, citado por [REI 2000].

Quando os ADS suportam a modelagem e execução de processos de software são denominados ADS orientados a processo[1] (ou centrados em processo). Além de coordenar o processo de desenvolvimento de software, o ADS deve auxiliar a cooperação entre os profissionais envolvidos. Um ambiente de desenvolvimento cooperativo de software suporta de forma automatizada e integrada a cooperação entre os profissionais envolvidos no ciclo de vida de software [REI 2000].

A evolução da arquitetura de ADS cooperativo, tem levado à necessidade de novas ferramentas e metodologias de desenvolvimento a fim de acompanhar o aumento da complexidade do software a ser desenvolvido, principalmente quando se tratar de software distribuído e sua integração com outras tecnologias, como a de agentes.

1.1 Motivação e Objetivos

Visando suprir essa necessidade, encontra-se em desenvolvimento desde 1999, no Departamento de Informática da Universidade Estadual de Maringá, o projeto de pesquisa "Uma Metodologia para o Desenvolvimento Baseado em Objetos Distribuídos Inteligentes" [HUZ 99]. O projeto tem dois objetivos principais: 1) definição de uma metodologia para desenvolvimento de software baseado em objetos distribuídos; 2) construção de um ambiente integrado para o desenvolvimento de software distribuído considerando a metodologia proposta como sendo o modelo de processo. O projeto está sendo financiado pelo CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) e pela Fundação Araucária.

Para a execução deste projeto, têm sido definidos vários trabalhos, cada um deles tratando do desenvolvimento de um módulo ou ferramenta específica do ambiente.

O primeiro objetivo do projeto já foi realizado com a especificação da MDSODI (descrita sucintamente no anexo 1). A MDSODI, à medida que o projeto está sendo desenvolvido, está sofrendo algumas adequações com o objetivo de aprimorar os recursos/representações oferecidos.

Como parte do segundo objetivo, encontra-se a definição da arquitetura do ambiente. Esta arquitetura deverá: incorporar a MDSODI como modelo de processo e

[1] Também conhecido na literatura como PSEE – *Process-Centered Software Engineering Environment.*

também oferecer o suporte necessário através de ferramentas e tecnologias adequadas para o desenvolvimento de software distribuído.

Vários trabalhos encontram-se em andamento, dentre os quais pode-se citar: 1) definição e elaboração de uma ferramenta denominada MDSODI – Requisitos que dará apoio à MDSODI no que se refere à definição de requisitos considerando aspectos de distribuição, paralelismo e trabalho cooperativo e que será integrada no ambiente que está sendo proposto [BAT 2002]; 2) definição e elaboração de uma ferramenta de apoio para gerenciamento de desenvolvimento de software em sistemas distribuídos, denominada DIMANAGER (*Distributed Software Manager*) que oferece suporte à metodologia MDSODI e também será integrada ao ambiente proposto [PED 2002]; 3) elaboração de uma proposta de um repositório de metadados para ambiente de desenvolvimento de software distribuído. Este trabalho pretende fornecer recursos que possibilitem a integração de artefatos gerados pelas ferramentas de desenvolvimento de software ligadas a MDSODI, devendo ser compatível com a arquitetura proposta nesta dissertação [MOR 2002].

Portanto, as características apresentadas pelas atuais aplicações, como por exemplo: telecomunicações, controle de processo, ensino a distância, em grande parte complexas e distribuídas; pelas recentes tecnologias, dentre as quais a de agentes e de trabalho cooperativo, motivam o desenvolvimento de pesquisas/trabalhos cujos resultados possam ser utilizados pelos desenvolvedores de software.

Segundo Jennings [JEN 2001], o uso de agentes é útil na modelagem e construção de software distribuído complexo. Já o trabalho cooperativo surge naturalmente devido às características de software distribuído, já que para o seu desenvolvimento pode, muitas vezes, ser exigido que os responsáveis estejam em locais geograficamente distintos e conseqüentemente a execução do software também ocorre de forma distribuída.

É neste contexto que está sendo definida essa dissertação que tem como principal objetivo especificar uma arquitetura de um ADS distribuído baseada em agentes. O ADS proposto será chamado DiSEN (*Distributed Software Engineering Environment)* e doravante assim referenciado também.

DiSEN estará sendo definido considerando-se, além das características/recursos necessários ao desenvolvimento de software eminentemente seqüencial, também os aspectos relacionados ao trabalho cooperativo e agentes.

A principal contribuição desse trabalho é a especificação do DiSEN que poderá estar distribuído geograficamente, incorporar a MDSODI, proporcionar o desenvolvimento distribuído e possuir atividades executadas por agentes obedecendo às especificações da FIPA.

A arquitetura proposta adotará o estilo camadas, sendo constituída por três camadas: dinâmica, aplicação e infra-estrutura. A camada dinâmica será responsável pelo gerenciamento da configuração do ambiente; a camada de aplicação suportará elementos do gerenciador de objetos, a MDSODI, gerenciamento de *workspace*, gerenciamento de agentes e conterá o repositório para armazenamento dos dados necessários ao ambiente e, no futuro, as ontologias, e a camada da infra-estrutura proverá suporte às tarefas de nomeação, persistência e concorrência e incorporará o canal de comunicação. Os elementos das camadas de aplicação e da infra-estrutura se comunicarão através do canal de comunicação. Para validar o DiSEN será realizada

uma simulação de como as partes se comunicam, elaborando-se os diagramas de *use* case e de seqüência, conforme definidos na MDSODI.

A tarefa de especificar um ADS é bastante complexa, assim não se espera com esse trabalho definir em detalhes como seria a implementação de todos os elementos que constituem o DiSEN e nem tão pouco mostrar a sua implementação. Isto será alcançado em trabalhos futuros.

1.2 Organização do Trabalho

Esta dissertação está organizada como segue:

No segundo capítulo é apresentado o estado-da-arte relevante para esta dissertação, abrangendo os seguintes tópicos: ambientes de desenvolvimento de software, agentes de software, sistemas distribuídos e desenvolvimento cooperativo de software. Esses tópicos formam uma base importante para o trabalho, já que eles são objetos de estudo e são constantemente referenciados.

No terceiro capítulo é apresentada a proposta de arquitetura do DiSEN. São descritos os elementos que constituem as camadas da arquitetura.

O capítulo 4 descreve os aspectos de comunicação e cooperação que devem ser considerados na interação entre os agentes.

O capítulo 5 descreve a simulação e validação do DiSEN. Na simulação apresentada foram elaborados vários diagramas com base na notação definida na MDSODI.

O capítulo 6 apresenta as conclusões, contribuições, os trabalhos que se encontram em andamento e sugere alguns trabalhos futuros.

O Anexo 1 apresenta a MDSODI, dando uma breve descrição de suas etapas e representações.

2 Estado-da-Arte

Este capítulo descreve o estado-da-arte relevante para a dissertação, para que seja possível o entendimento dos tópicos que envolvem o ambiente que está sendo proposto: ambientes de desenvolvimento de software (ADS), agentes de software, sistemas distribuídos e desenvolvimento cooperativo de software. E descreve também alguns trabalhos relacionados à nossa proposta.

2.1 Ambientes de Desenvolvimento de Software

Um ADS é definido como sendo um sistema computacional que provê suporte para o desenvolvimento, reparo e melhorias em software e para o gerenciamento e controle dessas atividades [MOU 92]. Para tal, um ADS contém um repositório com todas as informações relacionadas ao desenvolvimento do projeto ao longo do seu ciclo de vida, além de ferramentas que oferecem o apoio às várias atividades, técnicas e gerenciais, passíveis de automação que devem ser realizadas no projeto.

Um ADS deve se preocupar com o apoio às atividades individuais e ao trabalho em grupo, o gerenciamento do projeto, o aumento da qualidade geral dos produtos e o aumento da produtividade, permitindo ao engenheiro de software acompanhar o projeto e medir, através de informações obtidas ao longo do desenvolvimento, a evolução dos trabalhos [TRA 94].

A área de ADS cresceu rapidamente durante a última década, quando várias teorias e ambientes experimentais foram propostos para melhorar o apoio à engenharia de software. O entendimento e a comparação dessas teorias e ambientes foram dificultados devido à diversidade de representação dos serviços e elementos dos ambientes e devido à ausência de uma terminologia uniforme. O Modelo de Referência ECMA (*European Computer Manufacturers Association*) foi proposto baseado nessas dificuldades [BRO 92]. A subseção seguinte descreve esse modelo.

2.1.1 Modelo de Referência ECMA

O Modelo de Referência ECMA define uma arquitetura para ambientes integrados de desenvolvimento de software. Um modelo de referência é uma estrutura conceitual e funcional que facilita a descrição e comparação de sistemas [BRO 92].

O modelo ECMA segue uma visão orientada a serviços que se concentra nas capacidades do ambiente e abstrai detalhes de implementação. A figura 2.1 apresenta sua estrutura geral. O diagrama não deve ser interpretado como um conjunto de camadas nem como um modelo de implementação. Os serviços de mensagem permitem a comunicação nos dois sentidos: serviço a serviço, ferramenta a ferramenta e serviço a ferramenta [BRO 92].

Esse modelo oferece uma terminologia uniforme para descrever os mecanismos e serviços que compõem a infra-estrutura de um ambiente. Dessa forma, o modelo facilita a comunicação entre os pesquisadores da área e serve, principalmente, como referência para comparação de ambientes [GIM 94]. A seguir, a figura 2.1 descreve, sucintamente, os elementos do modelo ECMA:

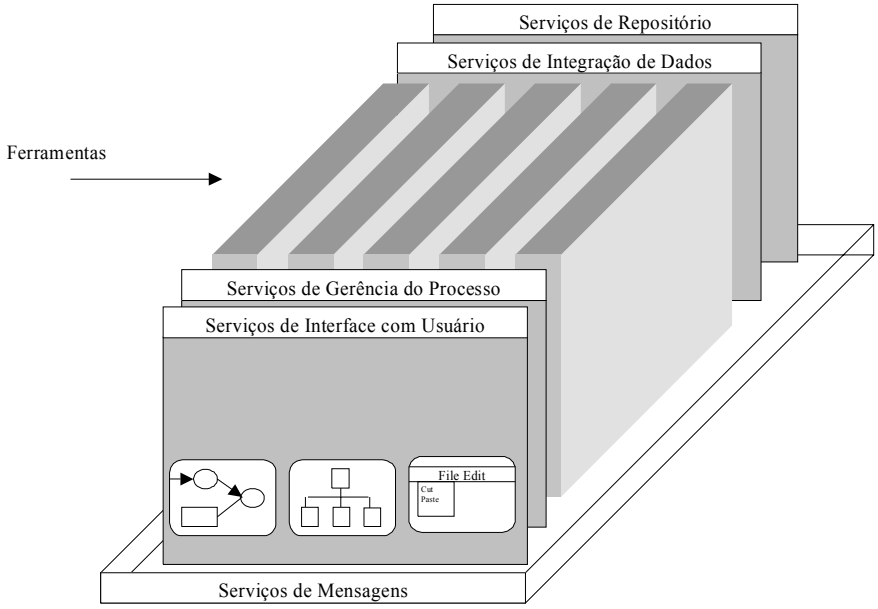


FIGURA 2.1 – Estrutura geral do modelo de referência ECMA

- **Serviços de Mensagens** → permitem a comunicação entre os diversos elementos de um ADS. Fornecem um padrão de comunicação que pode ser usado pelas ferramentas e outros serviços;
- **Serviços de Interface com Usuário** → provêem consistência de interface adotada em todas as ferramentas;
- **Serviços de Gerência do Processo** → permitem definição e execução de atividades de processo de desenvolvimento de software aumentando a "inteligência" do ambiente;
- **Ferramentas** → são os componentes de software que não fazem parte do ADS e oferecem suporte ao desenvolvimento de aplicações utilizando os serviços do ambiente;
- **Serviços de Integração de Dados** → acrescentam maior funcionalidade aos serviços disponibilizados pelo repositório, provendo mecanismos de controle de versões e configurações, consultas e metadados;
- **Serviços de Repositório** → administram o armazenamento dos objetos de dados, provendo mecanismos de localização, controle de concorrência e *backup*. Oferecem suporte básico ao controle e execução de processos.

Além desses serviços, a estrutura ainda provê mecanismos de segurança para utilização de informações privadas ou de acesso controlado. Mais detalhes sobre os serviços providos pelo modelo ECMA podem ser encontrados em Brown [BRO 92].

A próxima subseção trata de ADS orientados a processo. Estes ambientes constituem uma geração de ADS que suportam, além da função de desenvolvimento de

software, as funções de gerência e garantia da qualidade durante o ciclo de vida do software [REI 2000a].

2.1.2 Ambientes Orientados a Processo

Com a definição de ADS e a preocupação de que software não deve ser desenvolvido de forma *ad hoc*, surgiram pesquisas em ambientes de desenvolvimento, em particular os ambientes centrados em processo, onde se defende a idéia de ferramentas integradas e incorporadas a um processo de desenvolvimento de software específico. Um processo de desenvolvimento de software, ou processo de software [GIM 94], corresponde ao conjunto de atividades relacionadas que são desempenhadas pelos desenvolvedores desde a concepção até a liberação do produto. Uma forma de analisar e amadurecer tal processo é através da sua descrição, a qual consiste em um modelo de processo de software. A descrição formal de um processo de software é a atividade que permite que o mesmo seja analisado, compreendido e automatizado (executado). SPADE [BAN 92], EPOS [CON 94], PROSOFT [NUN 94] e OIKOS [MON 94] são alguns exemplos de ADSs orientados a processo encontrados na literatura.

O uso de ambientes orientados a processos força a definição rigorosa da execução do processo. Essa característica traz benefícios, pois permite melhorar a comunicação entre as pessoas envolvidas e a consistência do que está sendo feito. Além disso, enquanto o processo executa o modelo de processo, o ambiente sempre provê informações que guiam o desenvolvedor a realizar seu trabalho com mais eficiência. Neste sentido, algumas ações automáticas também podem ser realizadas pelo ambiente "liberando" os seus usuários de tarefas repetitivas e fornecendo informações sobre o processo quando for necessário [REI 98]. Para tanto, alguns requisitos são considerados indispensáveis e estão presentes na maioria dos ADS orientados ao processo. São eles:

- **Suporte a múltiplos usuários**: mecanismos para atender vários usuários ao mesmo tempo requisitando serviços do ambiente;
- **Gerência de objetos**: controla o acesso e a evolução de objetos compartilhados. As versões dos documentos e artefatos de software devem ser gerenciadas para permitir cooperação e consistência;
- **Gerência de comunicação entre pessoas**: as pessoas que estarão envolvidas no desenvolvimento de software devem ter acesso a mecanismos de comunicação, tais como mensagens eletrônicas e conferência eletrônica;
- **Gerência de cooperação**: a edição cooperativa de documentos e itens de software deve ser gerenciada pelo ambiente de forma que os usuários envolvidos obtenham comunicação síncrona sobre os artefatos que estão manipulando;
- **Gerência de Processo**: o suporte à modelagem e execução do processo de software têm sua importância na gerência das atividades desenvolvidas pelas pessoas e ferramentas durante a construção de software. Dentro desta característica, encontra-se a necessidade de um formalismo de modelagem de processo e uma máquina de execução das definições de processo;
- **Extensibilidade**: permite a extensão do ambiente através da inclusão de novas ferramentas, sejam elas de apoio ao desenvolvimento de software ou com outras funções;

- **Integração entre todos os módulos**: os níveis de integração: carregador, léxico, sintático, semântico e de método propostos por Brown [BRO 92], devem estar disponíveis para viabilizar os outros requisitos. As ferramentas do ambiente devem concordar sobre os tipos de dados, operações, métodos utilizados e o processo de desenvolvimento sendo seguido.

Além dos requisitos citados acima, um ADS deve possuir suporte para medição, ou seja, o ADS deve armazenar, ao longo do processo de desenvolvimento, informações relevantes que possibilitem fazer medidas sobre o processo de desenvolvimento. Essas métricas, estando disponíveis para futuros projetos, possibilitam aperfeiçoamentos no processo de desenvolvimento e, conseqüentemente, melhoria na qualidade dos artefatos.

A subseção seguinte apresenta as considerações finais da seção Ambientes de Desenvolvimento de Software.

2.1.3 Considerações Finais

Esta seção apresentou o conceito de ADS, explicou o modelo de referência ECMA e tratou de ADS orientados a processo, mostrando alguns requisitos que são considerados indispensáveis e estão presentes na maioria dos ADS orientados ao processo.

Em relação a esses requisitos, no DiSEN, a gerência de objetos e gerência de processos serão fornecidas pelo Gerenciador de Objetos; a gerência de cooperação bem como a gerência de comunicação entre pessoas será provida pelo Gerenciador de *Workspace*. O suporte a múltiplos usuários será um requisito fundamental para nossa proposta, já que o ADS será cooperativo e o domínio de aplicação que se pretende considerar é o de sistema distribuído.

A seção seguinte trata sobre a tecnologia de agentes: apresenta o conceito de agentes, as propriedades que permitem identificar uma entidade de software como sendo um agente, a interação de agentes, identifica algumas aplicações que são tipicamente de difícil implementação e para as quais a utilização de agentes aponta soluções viáveis e, finalmente, apresenta, sucintamente, a FIPA e os trabalhos desenvolvidos pela mesma.

2.2 Agentes de Software

A tecnologia de agentes é uma área de pesquisa relativamente nova e que tem atingido um alto nível de reconhecimento na comunidade científica internacional devido ao potencial que possui para o projeto de sistemas complexos e sistemas situados em ambientes dinâmicos. Esses tipos de sistemas são tipicamente difíceis de tratar usando concepções tradicionais de sistemas computacionais [BOR 2001]. A idéia de empregar agentes de software e delegar a eles certas atividades foi introduzida por pesquisadores [WOO 95] [JEN 98] que perceberam as grandes oportunidades desta promissora área. Assim, nossa proposta incorporará a tecnologia de agentes, já que o domínio de aplicação que se pretende considerar é o de sistema distribuído que é inerentemente complexo e dinâmico.

Esta seção apresenta o conceito de agentes, as propriedades e interações de agentes, algumas aplicações onde se pode empregar agentes e, finalmente, a FIPA.

2.2.1 Conceito de Agentes

Um agente de software é definido como uma entidade de software autônoma que tem capacidades, interage com seu ambiente e adapta seu estado e comportamento com base nesta interação. Este ambiente de interação pode ser constituído de máquinas, de humanos, e de outros agentes de software em vários ambientes e através de várias plataformas [OMG 2000]. Um agente de software normalmente não é encontrado sozinho em uma aplicação, mas freqüentemente forma uma organização com outros agentes, constituindo assim o que é denominada aplicação multi-agente. Uma aplicação multi-agente geralmente tem diversos tipos de agentes de software, como agentes de informação, agentes usuários e agentes de interface [NWA 96].

Na subseção seguinte são descritas as propriedades necessárias aos agentes para que seja possível diferenciá-los de um programa convencional.

2.2.2 Propriedades dos Agentes

Para que possamos diferenciar um agente de um programa convencional são necessárias algumas propriedades. Essas propriedades são características comportamentais que um agente pode ter para alcançar seus objetivos. A tabela 2.1 resume as definições para as principais propriedades de agentes [GAR 2001]. Em geral, autonomia, interação e adaptação são consideradas propriedades fundamentais de agentes de software.

TABELA 2.1 - Um *overview* de propriedades de agentes

Propriedade	Definição
Interação	Um agente comunica-se com o ambiente e outros agentes.
Adaptação	Um agente adapta/modifica seu estado de acordo com as mensagens recebidas do ambiente.
Autonomia	Um agente é capaz de agir sem intervenção externa direta; ele possui seu próprio controle e pode aceitar ou recusar uma mensagem.
Aprendizagem	Um agente pode aprender, baseado em experiências anteriores enquanto reage e interage com seu ambiente.
Mobilidade	Um agente é capaz de mover-se de um ambiente para outro.
Colaboração	Um agente pode cooperar com outros agentes para alcançar seus objetivos e os do sistema também.
Reatividade	Um agente é capaz de reagir a mudanças ocorridas no ambiente.
Proatividade	Um agente é capaz de tomar iniciativas que visem alcançar seus objetivos.

Os agentes que estarão presentes no DiSEN deverão ser capazes de executar tarefas de forma autônoma, reagindo aos estímulos do ambiente no qual estão inseridos,

tomando suas próprias iniciativas e interagindo com outros agentes. Além disso, pretende-se seguir o padrão estabelecido pelas especificações FIPA (que será melhor detalhada na subseção 2.2.5) para o desenvolvimento desses agentes.

Na próxima subseção são apresentados os elementos chave que devem ser considerados quando se tratar de interação de agentes.

2.2.3 Interação de Agentes

Interação é uma das características mais importantes de um agente [NWA 96]. Ou seja, os agentes periodicamente interagem para compartilhar informações e executar tarefas para alcançar seus objetivos. Pesquisadores que investigam sobre linguagens de comunicação de agentes mencionam três elementos chave para alcançar interação multi-agente [FLO 99]:

- Uma linguagem e um protocolo de comunicação de agente que sejam comuns;
- Um formato comum para o conteúdo de comunicação;
- Uma ontologia compartilhada.

Linguagens de Comunicação de Agentes

É necessário padronizar as linguagens de comunicação de agentes (ACL), a fim de permitir que desenvolvedores diferentes possam construir seus agentes para interoperar. Além disso, elas devem ter uma semântica formal de forma que diferentes implementações preservem as características essenciais da ACL. Através da especificação de uma ACL, os elementos básicos de interação entre os agentes são efetivamente codificados [OMG 2000].

Mecanismo de Transporte de Mensagem

Em ambientes de agente, mensagens devem ser escalonáveis (*schedulable*), bem como dirigidas a evento. Podem ser enviadas de modo síncrono ou assíncrono. Além disso, o mecanismo de transporte deve suportar endereçamento único bem como endereçamento baseado em papéis (por exemplo, endereçamento páginas brancas versus páginas amarelas). E finalmente, o mecanismo de transporte deve suportar modos *unicast*, *multicast* e *broadcast* e serviços semelhantes como comportamento *broadcast*, não-rejeição de mensagens e *logging* [OMG 2000].

Ontologia

Uma ontologia é um conjunto de símbolos junto com uma interpretação associada que pode ser compartilhado por uma comunidade de agentes ou software. Uma ontologia envolve um vocabulário de símbolos referentes aos objetos e relacionamentos que podem ser evidentes no domínio [FIP 2001b].

Quando tratamos de comunicação entre agentes está implícito que conceitos farão parte desta comunicação, porém alguns agentes podem ter diferentes termos para o mesmo conceito, termos idênticos para conceitos diferentes e classes de sistemas diferentes. Assim, uma ontologia comum é requerida para representar o conhecimento através dos vários domínios [OMG 2000].

O DiSEN deverá ter um servidor de ontologia que tem como função armazenar as ontologias necessárias para que os agentes possam se comunicar no ADS. A definição deste servidor seguirá a especificação Serviço de Ontologia FIPA [FIP 2001d].

A subseção seguinte exemplifica algumas aplicações onde se pode utilizar a tecnologia de agentes.

2.2.4 Aplicações de Agentes

Para Brenner [BRE 98], os agentes de software podem ser diferenciados em três áreas de tarefas gerais: agentes de informação, agentes de cooperação e agentes de transação. O critério para atribuição da área de aplicação em uma das três categorias está baseado em sua tarefa central.

A tarefa primária de um agente de informação diz respeito ao suporte de seu usuário na busca por informação em uma rede ou sistema distribuído. O agente de informação deve ser capaz de executar as seguintes tarefas: localizar fontes de informação; extrair informações dessas fontes; filtrar a informação relevante para o usuário; preparar e apresentar os resultados de forma apropriada.

O agente de cooperação tem como principal tarefa resolver problemas complexos usando mecanismos de comunicação e cooperação com outros objetos, como agentes, humanos ou recursos externos. Agentes de cooperação são usados quando o problema excede as capacidades de um agente individual ou já existem agentes que tenham uma solução e cujo conhecimento pode ser usado por outros agentes.

Os agentes de transação são especialmente apropriados para tarefas como processamento e monitoração de transações em ambientes de rede, comércio eletrônico e banco de dados. Aspectos como segurança, proteção de dados, robustez e integridade têm um papel central no projeto de agentes de transação. A tabela 2.2 mostra as áreas de aplicações individuais e suas atribuições para as três categorias.

TABELA 2.2 – Agrupamento das áreas de aplicações individuais com respeito as suas tarefas centrais

Tarefa específica	Área de Aplicação
Agente de Informação	• Recuperação e filtro de informação: auxilia o usuário na busca de informação na Internet.
	• *NewsWatcher*: combina agentes cujo propósito é o fornecimento de informação relevante através da definição de um perfil pessoal do usuário.
	• Notificação e observação: auxilia a busca de informação através da observação do usuário enquanto ele trabalha na WWW.
	• Tráfego: esta área está interessada no suporte dos agentes no fornecimento de serviços como planejamento de viagens e transporte.

Agente de Cooperação	• Entretenimento: usa a cooperação para construir um perfil de interesse com o qual o usuário pode descobrir novas atividades recreativas.
	• *Groupware*: esta área é parte da área administrativa e operacional e a necessidade de comunicação para executar as tarefas pertinentes a essa área força os agentes a cooperarem entre si.
	• Gerenciamento de redes/telecomunicação: esta área foca o suporte de agentes no gerenciamento de tarefas em uma rede.
Agente de Transação	• Comércio eletrônico: envolve agentes que auxiliam o usuário enquanto compra ou vende produtos e serviços.
	• Manufatura: concentra-se na compreensão de sistemas multi-agentes para o planejamento, controle e coordenação de processos de manufatura flexíveis e distribuídos. Esta área de aplicação está interessada somente no suporte de processos de produção.
	• Gerenciamento de processo de negócio: esta área está interessada no suporte de agentes no desenvolvimento de processos de negócio.

A próxima subseção trata, sucintamente, da organização internacional FIPA e também das especificações propostas até o momento por esta organização.

2.2.5 FIPA (*Foundation for Intelligent Physical Agents*)

A FIPA é uma organização internacional, constituída por empresas e universidades que estão atuando no desenvolvimento de aplicações baseadas em agentes e está se propondo a estabelecer um padrão a ser seguido para o desenvolvimento de software baseado em agentes. Foi fundada em 1996 e se dedica ao desenvolvimento de especificações que suportem interoperabilidade entre agentes e aplicações baseadas em agentes, como por exemplo, arquitetura abstrata, gerenciamento de agentes e transporte de mensagens. Dentre estas especificações, já propostas, encontram-se a arquitetura abstrata FIPA [FIP 2001b], a linguagem de comunicação de agentes (ACL - *Agent Communication Language*) [FIP 2001c] e o gerenciamento de agente [FIP 2001].

A especificação arquitetura abstrata FIPA [FIP 2001b] tem o propósito de promover interoperabilidade e reusabilidade através da identificação de elementos da arquitetura que devem ser codificados. Ela define entidades abstratas como serviço de diretório, transporte de mensagem que podem ser instanciadas em outras especificações. A especificação linguagem de comunicação de agentes [FIP 2001c] detalha a sintaxe e a semântica de uma linguagem de comunicação, alto-nível, de agente que é baseada em atos de fala [BOR 2001]. A especificação gerenciamento de agente [FIP 2001] trata o que é necessário para gerenciar agentes em uma plataforma de agente (PA), sendo que o protocolo de transporte básico definido por esta especificação é o IIOP (*Internet Inter-ORB Protocol*).

A OMG (*Object Management Group*), através do *Agent Working Group*, publicou o documento "*Agent Technology – Green Paper*" [OMG 2000], onde ressalta

que está sendo estabelecida uma ligação formal entre a FIPA e o OMG's *Agent* WG para utilização das especificações já desenvolvidas pela FIPA como *background* para futuros esforços de padronização pela OMG.

Segundo a especificação gerenciamento de agente [FIP 2001], uma PA deve prover a infra-estrutura física na qual os agentes podem ser disponibilizados. Um agente deve ser registrado em uma PA para interagir com outros agentes naquela ou em outras plataformas e, neste caso, a comunicação entre as plataformas é feita através do serviço de transporte de mensagem. A FIPA considera a PA um elemento chave na arquitetura de um agente para permitir que uma sociedade de agentes interoperem e sejam gerenciados. Uma PA consiste, no mínimo, de um sistema de gerenciamento de agentes (SGA), um facilitador de diretório (FD) e um sistema de transporte de mensagem (STM) [OMG 2000]. A figura 2.2 mostra o modelo de referência de uma PA [FIP 2001].

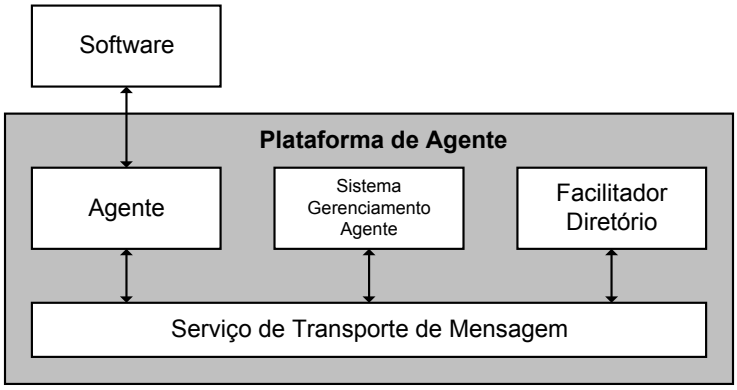


FIGURA 2.2 – Modelo de referência FIPA de uma plataforma de agente

A seguir estão descritos os elementos que constituem o modelo de referência de uma PA:

- **Software** → descreve toda coleção de instruções executáveis (não agentes) que são acessadas através de um agente. Agentes podem acessar software, por exemplo, para adicionar novas tarefas, adquirir novos protocolos de comunicação, novos protocolos/algoritmos de segurança, novos protocolos de negociação, acessar ferramentas que suportam migração, etc.
- **Agente** → é um ator fundamental em uma PA que combina uma ou mais tarefas em um modelo de execução unificado e integrado que pode incluir acesso a software externo, usuários humanos e facilidades de comunicação.
- **Sistema de Gerenciamento de Agente (SGA)** → é um agente que supervisiona o acesso e o uso da PA e, também, mantém um diretório de identificadores de agentes que contém, por exemplo, endereços de transporte para agentes registrados na PA. O SGA é responsável por gerenciar o ciclo de vida dos agentes na plataforma.
- **Facilitador de Diretório (FD)** → é um agente que provê serviços de páginas amarelas para outros agentes. O FD armazena descrições dos agentes e os serviços que eles oferecem. Assim, os agentes podem registrar seus serviços com o FD ou podem consultá-lo para descobrir quais serviços são oferecidos por outros agentes.

- **Serviço de Transporte de Mensagem (STM)** → o STM distribui mensagens entre agentes residentes em uma PA e para agentes em outras PAs. Todos os agentes têm acesso a pelo menos um STM e somente mensagens endereçadas para um agente podem ser enviadas para o STM.

A FIPA estabelece um modelo de referência para o transporte de mensagens de agentes. Este modelo inclui três níveis, conforme mostra a figura 2.3 [FIP 2001a].

1. O **Protocolo de Transporte de Mensagem** (PTM) é usado para efetuar a transferência física de mensagens entre dois canais de comunicação de agentes. O canal de comunicação de agente é um agente que usa a informação provida pelo SGA para estabelecer a rota das mensagens entre agentes da mesma PA e agentes que residem em outras PAs. Deve suportar IIOP para interoperabilidade entre diferentes PAs.

2. O **Serviço de Transporte de Mensagem** (STM) é um serviço fornecido pela PA na qual um agente está ligado. O STM auxilia o transporte de mensagens FIPA ACL entre agentes em uma determinada PA e entre agentes em diferentes PAs.

3. A ACL representa o conteúdo das mensagens transportadas pelos PTM e STM.

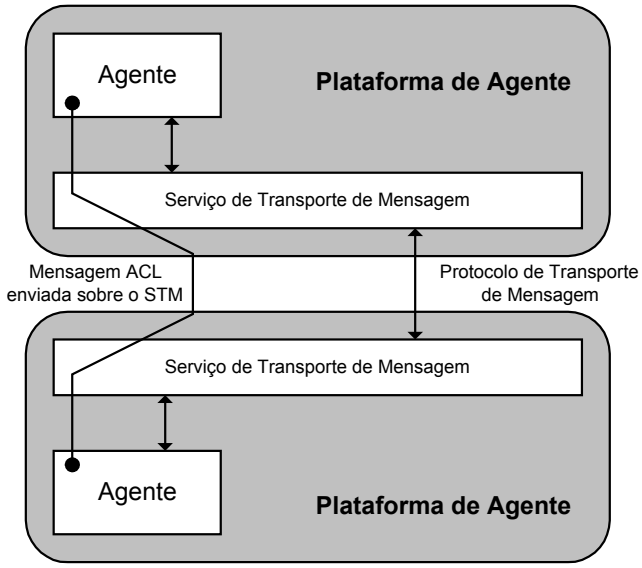


FIGURA 2.3 – Modelo de referência FIPA de transporte de mensagem

O servidor de ontologia é responsável pelo armazenamento das ontologias. Pode existir mais de um servidor de ontologia, sendo que um agente de ontologia será responsável por prover o acesso a este(s) servidor(es). O agente de ontologia deve ser capaz de participar na comunicação das tarefas abaixo relacionadas, possivelmente respondendo quando não for capaz de executar qualquer uma delas:

- Ajudar um agente na seleção de uma ontologia compartilhada para comunicação;
- Criar e atualizar uma ontologia, ou somente alguns termos de uma ontologia;
- Traduzir expressões entre diferentes ontologias (diferentes nomes com o mesmo significado);
- Responder consultas de relacionamentos entre termos ou entre ontologias, e,

- Descobrir ontologias públicas e acessá-las.

Além disso, o agente de ontologia permite que o servidor de ontologia torne suas ontologias publicamente disponíveis no domínio dos agentes [FIP 2001d]. A figura 2.4 mostra o modelo de referência FIPA de serviço de ontologia. Esse modelo não impede que, em algumas implementações, o agente de ontologia possa envolver diretamente um servidor de ontologia. No caso do DiSEN, o servidor de ontologia estará embutido no repositório.

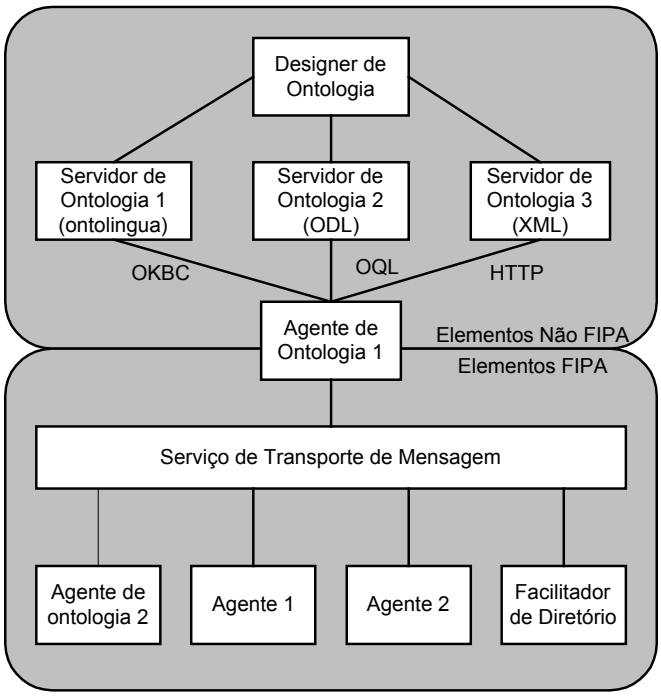


FIGURA 2.4 – Modelo de referência FIPA de serviço de ontologia

A subseção seguinte apresenta as considerações finais da seção Agentes de Software.

2.2.6 Considerações Finais

Nesta seção foram apresentados o conceito de agente, um conjunto de características que permitem identificar entidades de software como sendo agentes, os elementos chave para alcançar interação entre agentes, exemplos de áreas nas quais a tecnologia de agentes tem sido ou será aplicada. Foi também tratado sobre a FIPA, que tem sido considerada uma das mais interessantes respostas ao esforço de padronização das aplicações baseadas em agentes. Esse esforço começou com o trabalho feito pela *Knowledge Sharing Effort* [PAT 92]. O trabalho de padronização da FIPA está direcionado para permitir uma fácil interoperabilidade entre sistemas baseados em agentes [BEL 2000].

Vale ainda ressaltar que a utilização da tecnologia de agentes tem uma importância muito grande em um mundo competitivo como o atual, pois os usuários são liberados de realizar tarefas repetitivas ou que possam ser automatizadas para fazerem outras mais importantes. Os agentes realizarão tais tarefas para seus usuários sempre buscando o melhor resultado no menor tempo possível.

Um outro ponto a considerar ainda é que a crescente complexidade das aplicações, a contínua evolução tecnológica e o uso cada vez mais disseminado de redes de computadores têm impulsionado os estudos referentes ao desenvolvimento de sistemas distribuídos. Esse tipo de sistema oferece suporte a uma gama de aplicações, tais como: aplicações comerciais concorrentes, aplicações que trabalham com acesso concorrente às informações em banco de dados e aplicações que envolvem compartilhamento de informações.

Assim, a próxima seção apresenta o conceito e as principais características de sistemas distribuídos e também de objetos distribuídos.

2.3 Sistemas Distribuídos

Os avanços tecnológicos, a busca por inovações e as aplicações que apresentam complexidade cada vez maior têm impulsionado os estudos referentes a sistemas distribuídos. Esses sistemas, quando comparados aos sistemas convencionais, podem vir a apresentar maior eficiência, devido às suas características diferenciadas como, por exemplo, suporte aos recursos compartilhados, tolerância à falhas e concorrência.

A nova era da computação distribuída surge como uma evolução da tradicional tecnologia cliente/servidor, promovendo o fortalecimento desse conceito com a utilização de sistemas computacionais nos quais objetos são distribuídos em máquinas diferentes, podendo estar em diferentes locais, sendo executados sob diferentes plataformas e comunicando-se mediante a utilização de componentes responsáveis por gerenciar as chamadas remotas.

A tecnologia de objetos distribuídos permite simplificar o desenvolvimento e a manutenção de sistemas, criando peças reutilizáveis de código - conhecidos como objetos - e fazendo com que objetos instalados em diferentes computadores possam se comunicar através de uma rede.

Esta seção apresenta o conceito e as principais características de sistemas e objetos distribuídos.

2.3.1 Conceito de Sistemas Distribuídos

O termo sistema distribuído originou com o desenvolvimento de computadores multi-usuário e das redes de computadores. Foi estimulado pelo desenvolvimento das *workstations* de baixo custo, das redes locais e do sistema operacional UNIX.

Sistemas distribuídos consistem em uma coleção de computadores autônomos ligados por uma rede, buscando-se, desta forma, coordenar as atividades de maneira eficiente além de propiciar o compartilhamento de recursos, quer sejam de hardware ou de software [COU 2001]. Esse tipo de sistema oferece suporte a diversos tipos de aplicações, tais como: aplicações comerciais concorrentes, aplicações que trabalham com acesso concorrente às informações em banco de dados e aplicações que envolvem compartilhamento de informações.

Um sistema distribuído é constituído basicamente por: servidores de arquivo, computadores pessoais, estações de trabalho e outros servidores, como por exemplo, impressão e *login* ligados em rede.

A seção seguinte apresenta as principais características de um sistema distribuído.

2.3.2 Principais Características de Sistemas Distribuídos

Os sistemas distribuídos se distinguem dos tradicionais, basicamente, devido às características descritas a seguir.

Compartilhamento de recursos

Em um sistema distribuído, diversos recursos podem ser compartilhados. Estes recursos podem ser componentes de hardware, como discos, impressoras, *scanners*, ou de software, como banco de dados, arquivos e outros objetos de dados.

É importante lembrar que esse compartilhamento de recursos é permitido entre todos os usuários que estejam conectados ao sistema, tornando possível o compartilhamento de dados que sejam de interesse comum a mais de um usuário.

Os recursos estão fisicamente num dos computadores e podem somente ser acessados pelos outros através de comunicação. Para que o compartilhamento seja efetivo, para cada recurso deve haver um programa gerenciador que ofereça uma interface de comunicação, capacitando-o para ser manipulado, acessado e atualizado de forma confiável e consistente.

Abertura

Um sistema aberto possui interface pública e pode ser construído a partir de hardware e software de diferentes fornecedores, sem com isso acarretar problemas para o sistema.

Os sistemas distribuídos abertos permitem que novos serviços compartilhados sejam adicionados sem prejudicar os serviços existentes. Essa característica torna o sistema receptivo a modificações, mais flexível que os sistemas convencionais, sem que para isso sejam necessárias modificações complexas no sistema.

Concorrência

Os processos em um sistema distribuído podem trabalhar concorrentemente e paralelamente o que permite a execução de vários programas simultaneamente, sem com isto afetar o desempenho do sistema.

Quando se trata de concorrência em sistemas distribuídos devem-se tratar os aspectos relacionados à sincronização. Os acessos que ocorrem de forma concorrente assim como as atualizações feitas em recursos compartilhados devem ser atividades sincronizadas.

Escalabilidade

Um sistema distribuído é capaz de funcionar eficientemente em diversas escalas. Ele pode ser composto por apenas duas *workstations* e um servidor de arquivos ou até

centenas delas e muitos servidores de arquivos, de impressão e outros, possibilitando que recursos sejam compartilhados entre todos eles. Essa característica tenta garantir que o sistema e a aplicação não necessitem de mudanças quando a escala do sistema aumentar.

Tolerância a falhas

Quando ocorrem falhas, em hardware ou software, os programas podem produzir resultados incorretos ou eles podem parar antes de ter completado a atividade que vinha sendo realizada. O projeto de sistemas de computadores tolerantes a falhas é baseado em duas abordagens: redundância de hardware (uso de componentes redundantes ou em excesso) e restabelecimento de software (programas que recuperam as falhas ocorridas).

Diferente de um sistema centralizado onde o sistema se torna não disponível frente a uma falha de hardware, nesses sistemas o usuário pode simplesmente mudar de estação e continuar a realização de suas atividades normalmente a partir do ponto em que estava quando a falha ocorreu.

A recuperação de falhas de software utiliza a técnica conhecida como *rollback*, que permite que o estado em que o sistema se encontrava possa ser recuperado a partir do ponto em que estava quando uma falha foi detectada.

Transparência

A transparência permite ao usuário enxergar um sistema distribuído como uma única máquina e não como uma coleção de componentes independentes. Segundo o Manual da *ANSA (Advanced Network Systems Architecture)* [COU 2001], as duas formas de transparência mais importantes quando se trata de um sistema distribuído são: transparência de acesso e transparência de localização as quais estão brevemente descritas a seguir:

Transparência de acesso: permite que as informações de objetos, independentemente de sua localização, sejam acessadas através de operações similares;

Transparência de localização: permite que se acesse informações de objetos sem que para isso seja necessário saber a sua localização no sistema.

Tendo em vista as características de sistemas distribuídos, é notória a necessidade de se pensar em uma forma mais adequada de desenvolver softwares que possam usufruir destas. A idéia de objetos aliada a de sistema distribuído resultou no conceito de objetos distribuídos que se encontra descrito na próxima subseção.

2.3.3 Objetos Distribuídos

Objetos distribuídos são unidades de composição com interfaces e dependências de contexto especificadas de forma contratual [SZY 99]. Podem ser utilizados independentemente e são passíveis de composição por terceiros. Segundo Orfali [ORF 99], objetos distribuídos não são aplicações completas; podem ser usados em combinações imprevisíveis; têm interface bem definida; fornecem mecanismos para a notificação de eventos; possibilitam o gerenciamento de configuração e propriedades; são introspectivos, isto é, oferecem informações a respeito de si mesmos.

A característica principal dos objetos distribuídos refere-se à localização: eles podem estar localizados em uma única máquina ou distribuídos em máquinas distintas de uma rede de computadores sendo executados em diferentes plataformas. Em conjunto, esses objetos são capazes de executar funções para um sistema (sistema distribuído).

O objeto distribuído é uma evolução do objeto convencional. Entretanto possui uma interface específica onde os compiladores geram um código a mais para serem acessados por outros objetos de maneira que o programa/objeto que o solicite desconheça o local onde o objeto chamado está localizado, o sistema operacional que está sendo utilizado e a linguagem na qual foi criado.

A tecnologia de objetos distribuídos permite a criação de sistemas cliente/servidor mais flexíveis, visto que os dados são encapsulados nos objetos que ficam distribuídos pela rede o que facilita a localização destes em qualquer parte do sistema [ORF 96].

Para a construção e execução de objetos distribuídos é necessária uma estrutura de apoio adequada. Dentre as mais populares na literatura corrente, podem-se citar: CORBA (*Common Object Request Broker Architecture*) [OMG 2002], DCOM (*Distributed Component Object Model*) [MIC 2002] e Java/RMI (Java *Remote Method Invocation*) [SUN 2002]. São arquiteturas estruturadas diferentemente, mas que têm o mesmo objetivo. O CORBA é um padrão da OMG que propõe a completa homogeneidade entre softwares, o DCOM é uma tecnologia da *Microsoft Corporation* que oferece muito mais vantagens se aplicada para interagir com produtos da *Microsoft* e o Java/RMI é uma extensão da linguagem Java para criação de objetos distribuídos.

Para desenvolver objetos distribuídos, devem ser utilizadas linguagens específicas, sendo que cada uma proporciona uma solução diferente. As linguagens Java, *Microsoft Visual Basic*, *Microsoft Visual C++*, *PowerBuilder*, *Micro Focus Visual Object* COBOL e *Delphi* implementam objetos DCOM; enquanto que, *Delphi*, Java, C, C++, *Ada* e *Smaltalk* implementam objetos CORBA e objetos Java/RMI são implementados apenas em Java.

A subseção seguinte mostra as principais características dos objetos distribuídos.

2.3.4 Principais Características dos Objetos Distribuídos

Um objeto possui muitas características que favorecem a sua utilização quando se pensa em especificação e implementação de um sistema. As principais são: encapsulamento, abstração, herança, polimorfismo e, principalmente, o reuso.

Existem também muitos outros motivos, particulares à distribuição dos objetos, que tornam interessante associar objetos a sistemas distribuídos. Segundo Sametinger [SAM 98] e Orfali [ORF 96], os principais seriam:

Flexibilidade e distribuição dos dados: tudo o que se refere a dados é encapsulado dentro dos objetos, permitindo-se dessa forma que sejam localizados em qualquer parte de um sistema distribuído por qualquer outro objeto ou processo. Isso aumenta a interação entre os objetos já que a informação não fica restrita a um local

específico, ao contrário, o objeto pode ser acessado em qualquer parte da rede onde estiver localizado;

Independência da plataforma e do sistema operacional: permite que um mesmo objeto tenha seus métodos executados em diferentes plataformas e diferentes sistemas operacionais;

Independência no gerenciamento de informações: os objetos ficam distribuídos em locais variados na rede, o que lhe confere a independência no sentido de que cada qual pode se autogerenciar, além de gerenciar os recursos sob seu controle. Um outro fator associado à independência é o fato de poder alterar ou mesmo modificar a localização de um objeto sem que isso afete seu desempenho no sistema como um todo;

Transparência: os clientes não precisam de informações a respeito de onde o objeto se localiza ou mesmo em que sistema operacional é executado, pode ser na mesma máquina ou em qualquer outra distribuída pela rede, sem interferir no desempenho;

Interoperabilidade entre os objetos, independente dos sistemas operacionais;

Reuso: os objetos podem ser reutilizados em outras aplicações sem a necessidade de mudanças significativas.

A subseção seguinte apresenta as considerações finais da seção Sistemas Distribuídos.

2.3.5 Considerações Finais

Sistemas distribuídos têm adquirido uma importância crescente na indústria da computação devido à ampla difusão da Internet e às características diferenciadas quando comparados aos sistemas monousuários.

Esta seção apresentou essas características e também as de objetos distribuídos.

O aumento na complexidade das aplicações construídas atualmente tem levado ao aumento da quantidade de profissionais envolvidos no processo de desenvolvimento de software. Assim, o ciclo de vida de software caracteriza-se por um processo essencialmente cooperativo que envolve profissionais com habilidades e experiências distintas durante as várias fases do processo [BOR 95]. A seção seguinte tratará sobre desenvolvimento cooperativo de software, abrangendo os conceitos de suporte por computador ao trabalho cooperativo e ambientes de desenvolvimento cooperativo de software.

2.4 Desenvolvimento Cooperativo de Software

O desenvolvimento de sistemas complexos demanda cooperação intensiva entre os vários membros de um projeto com diferentes responsabilidades. O processo de desenvolvimento é freqüentemente distribuído através do tempo e espaço e realiza-se dentro e entre grupos de trabalho especializados [ALT 99].

Ambientes de desenvolvimento que, explicitamente, suportam trabalho em grupo são um importante pré-requisito para a produção de sistemas de software de alta qualidade. Deste modo, uma das áreas que tem despertado o interesse de pesquisadores refere-se ao estabelecimento de estudos relacionados ao apoio efetivo do envolvimento de diversos profissionais que atuam cooperativamente no processo de desenvolvimento de software [REI 98a].

Essa seção tem por objetivo apresentar o conceito de Suporte por Computador ao Trabalho Cooperativo e também de Ambientes de Desenvolvimento Cooperativo de Software.

2.4.1 Suporte por Computador ao Trabalho Cooperativo

O aumento na complexidade das aplicações desenvolvidas atualmente exige cada vez mais que as pessoas interajam e cooperem umas com as outras. A disponibilização de redes de computadores de alta velocidade, cada vez mais acessíveis, aliada à evolução dos estudos sobre o comportamento dos grupos de pessoas ao desempenhar uma atividade, permitiu o surgimento de uma disciplina que se convencionou chamar Suporte por Computador ao Trabalho Cooperativo (CSCW – *Computer Supported Cooperative Work*) [REI 98a].

Há aspectos que devem ser levados em consideração para que essa cooperação seja mais bem entendida, possibilitando que os problemas gerados por ela (como por exemplo, forma de comunicação desordenada) sejam minimizados. A figura 2.5 mostra esses aspectos [NUI 2001] .

O processo de cooperação, ou seja, o entendimento ou convergências de idéias entre as pessoas na realização de uma atividade de trabalho cooperativo compreende a:

- **Comunicação**: conhecimento dos canais disponíveis para troca de informações (correio eletrônico, por exemplo), para que as pessoas possam interagir mesmo estando em locais diferentes;
- **Coordenação**: controle das atividades do grupo para que a cooperação ocorra de forma ordenada;
- **Memória de Grupo**: informações armazenadas que podem ser úteis para futuras tomadas de decisões ou simplesmente para atualização de pessoas que ficaram ausentes do grupo por um determinado tempo;
- ***Awareness***: conhecimento de cada participante do grupo sobre as atividades que estão sendo realizadas, quem está trabalhando em cada atividade e quais são as futuras tarefas.

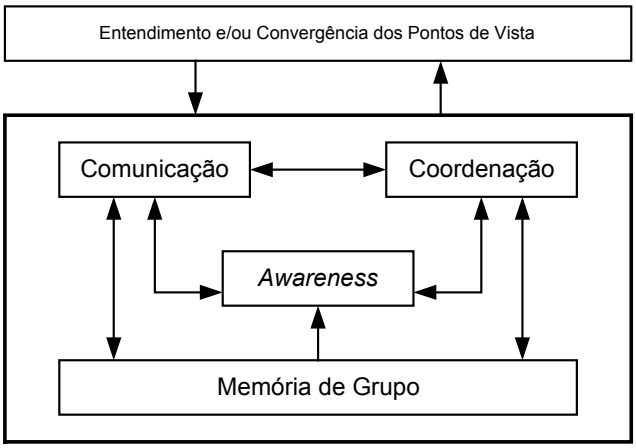


FIGURA 2.5 – Aspectos da cooperação

Os sistemas baseados em computador que suportam grupos de pessoas trabalhando juntas são chamados de *groupware.*

Groupware

Segundo Borges [BOR 95], o termo *groupware* costuma ser usado praticamente como sinônimo de CSCW, porém alguns autores identificam uma tendência diferenciada no emprego desses termos. Enquanto CSCW é usado para designar a pesquisa na área do trabalho em grupo e como os computadores podem apoiá-lo, *groupware* tem sido usado para designar a tecnologia (hardware e/ou software) gerada pela pesquisa em CSCW. Assim, sistemas de correio-eletrônico, teleconferências, suporte a decisão e editores de texto colaborativos são exemplos de *groupware*, na medida em que promovem a comunicação entre os membros de um grupo de trabalho, e que contribuem com isso para que o resultado seja maior que a soma das contribuições individuais de cada membro do grupo.

Um dos requisitos fundamentais de sistemas de *groupware* é que os sistemas sejam altamente configuráveis para se adaptarem às necessidades dos usuários. Na área de suporte ao trabalho cooperativo, o modelo de aplicações genéricas deve ser substituído pelo modelo de aplicações virtuais, que podem ser moldadas de acordo com um contexto particular de usuários, tarefas e ambientes. E, pelo potencial desse contexto de sofrer contínuas mudanças, essas aplicações virtuais devem ainda ter um caráter evolutivo de modo a permanecerem apropriadas ao seu contexto [BOR 95].

Portanto, os principais requisitos de CSCW são [BOR 95]:

- O sistema deve facilitar a cooperação entre indivíduos, ao invés de impor práticas que causem mudanças radicais na forma de trabalho;
- Sistemas de CSCW devem reconhecer que mudanças são freqüentes neste contexto e que, por isso, devem ser capazes de permitir a redefinição de procedimentos e processos, além de disseminar estas mudanças entre os participantes;

- A construção de aplicações menores e interrelacionadas é preferível ao desenvolvimento de aplicações monolíticas que incluem o conjunto completo de tarefas;
- Informações que serão usadas no trabalho cooperativo precisam estar fora do domínio de um indivíduo.

Esses requisitos também são importantes no momento da escolha do *groupware* mais apropriado para cada tarefa e para cada grupo de usuários, auxiliando, dessa forma, a distribuição das tarefas e melhorando o andamento do trabalho [NUI 2001].

Para Borges [BOR 95], as aplicações CSCW podem ser assim classificadas:

Sistemas de Mensagem. Suportam a troca assíncrona de mensagens textuais entre grupos de usuários. O correio eletrônico é um exemplo desse tipo de aplicação.

Sistemas de Co-autoria (Editores Cooperativos) . Podem ser usados por um grupo para compor e editar um objeto conjuntamente. Possuem uma área de trabalho comum onde os usuários atuam e podem visualizar a atuação dos outros.

Sistemas de Coordenação. Têm por objetivo gerenciar tarefas complexas e inter-relacionadas, juntamente com as informações por elas geradas e utilizadas. ADSs que suportam a modelagem e execução do processo de software são exemplos de sistemas de coordenação.

Sistemas de Suporte a Reuniões. Apóiam as sessões de trabalho em grupo face-a-face, suportando o entendimento, discussão ou decisão em grupo [REI 98a].

Desde o seu princípio, a área de CSCW vem buscando meios de suportar adequadamente o trabalho em equipe. Contudo um dos problemas apresentados por este suporte é a falta de contexto entre os participantes que ocorre quando os membros de um grupo de trabalho desconhecem o que seus colegas estão fazendo, ou não sabem onde suas atividades se encaixam no trabalho como um todo. O fornecimento deste contexto aos membros de um grupo é chamado *awareness* ou percepção [PIN 2001].

CSCW *e Awareness*

Awareness pode ser conceituada como sendo a contextualização das atividades individuais através da compreensão das atividades realizadas por outras pessoas [ARA 97]. Refere-se a ter conhecimento das atividades do grupo, saber o que aconteceu, o que está acontecendo e/ou poderá vir a acontecer, além do próprio conhecimento do que é este trabalho e o grupo [PIN 2001].

Sem *awareness*, o trabalho cooperativo coordenado é quase impossível, pois percepção é imprescindível para qualquer forma de cooperação, uma vez que perceber, reconhecer e compreender as atividades dos outros é requisito básico para a interação humana e a comunicação em geral [SOH 98]. No caso do desenvolvimento cooperativo de software, é necessário que os membros tenham noção do contexto de suas atividades no contexto geral do processo para que possam perceber o andamento das atividades realizadas pelos demais e compreender como os resultados gerados por estas atividades podem ser conjugados com os seus próprios, para mais rapidamente chegarem ao resultado final [ARA 97]. Neste caso, a ausência de suporte a *awareness* torna quase impossível a produção eficiente de um software consistente e de qualidade.

O DiSEN possui *workspaces* que permitirão a edição cooperativa de documentos e itens de software. Porém, para que um *workspace* dê suporte a *awareness* é necessário levar em consideração um conjunto de questões, como por exemplo, quais mudanças os participantes estão fazendo?, onde as mudanças estão sendo feitas?, o que os participantes podem fazer?, quais objetos os participantes estão usando? [GUT 96] [GUT 96a]. Neste momento, nossa proposta não irá tratar essas questões, deixando-as como trabalho futuro.

A próxima subseção apresenta o conceito de ambientes de desenvolvimento cooperativo de software e mostra quais tipos de *groupware* podem apoiar as fases do ciclo de vida do software.

2.4.2 Ambientes de Desenvolvimento Cooperativo de Software

Ambiente de desenvolvimento cooperativo de software (ADCS) é a denominação genérica utilizada para indicar os ambientes que suportam de forma automatizada e integrada a cooperação entre os profissionais envolvidos no ciclo de vida de software. É o produto-resultado da aplicação da tecnologia CSCW em ambientes de engenharia de software [REI 98a].

O apoio ao desenvolvimento cooperativo de software corresponde à disponibilização de técnicas e ferramentas que forneçam o apoio a grupos de engenheiros de software nas tarefas que são realizadas cooperativamente. Assim sendo, os ADCS surgem com o objetivo de proporcionar uma estrutura computacional que gerencie o intercâmbio de informações entre os desenvolvedores, mesmo que estes estejam em localidades geograficamente dispersas [REI 98a].

Groupware e Engenharia de Software

As primeiras tentativas de integrar os conceitos de CSCW em ferramentas de Engenharia de Software apoiavam a etapa de implementação, permitindo a inspeção e edição cooperativa de código. Atualmente, ferramentas *groupware* vêm sendo desenvolvidas para auxiliar todas as etapas que constituem o ciclo de vida do software. A tabela 2.3, proposta por Borges [BOR 95] mostra a correspondência entre os tipos de *groupware* e as fases do ciclo de vida do software que podem apoiar.

Observando a tabela 2.3, verifica-se que os sistemas de co-autoria podem ser utilizados durante praticamente todo o ciclo de vida de software. Por exemplo, em um ambiente de programação distribuído faz-se necessário um controle de versões dos programas-fonte, pois dois programadores podem estar trabalhando sobre o mesmo código-fonte de forma síncrona ou assíncrona, dependendo da atividade a ser realizada [REI 98a] [BOR 95].

TABELA 2.3 – *Groupware* para apoiar o ciclo de vida do software

Fase do Ciclo de Vida	*Groupware*
Análise	• Sistemas de suporte a decisão • Sistemas de co-autoria • Salas eletrônicas
Projeto	• Sistemas de suporte a decisão • Sistemas de co-autoria • Salas eletrônicas
Implementação	• Sistemas de co-autoria • Sistemas de conferência
Verificação	• Sistemas de suporte a decisão • Salas eletrônicas • Sistemas de conferência
Manutenção	• Sistemas de suporte a decisão • Sistemas de conferência

2.4.3 Arquitetura de Ambientes de Desenvolvimento Cooperativo de Software

Os ambientes que suportam desenvolvimento cooperativo de software possuem os requisitos comuns de um ADS como integração, gerência dos dados e processo, já descritos no item 2.1.1, e também devem permitir cooperação de forma eficiente.

Em Vessey [VES 95] é apresentado um estudo que define uma arquitetura para ADCS. A arquitetura mostra as diversas abordagens exploradas na literatura para configurar um ADCS, conforme ilustra a figura 2.6:

- **Tecnologia de Coordenação**
  - **Controle de acesso** provê os mecanismos de segurança necessários para apoiar o envolvimento dos diversos profissionais no ciclo de vida de software;
  - **Compartilhamento de informações** é um requisito imperativo para um ADCS, fornecendo os serviços para compartilhamento, garantia de consistência e controle de concorrência na manipulação de objetos de engenharia de software;
  - **Monitoração** envolve o controle das atividades realizadas pelos usuários sobre os objetos.
- **Tecnologia de Cooperação**
  - Apoio à **comunicação** entre os profissionais de desenvolvimento de software é um dos requisitos de um ADCS e é alcançado através de sistemas de correio eletrônico, conferências eletrônicas, dentre outros;
  - A **gerência de reuniões e horários** está relacionada com o controle das atividades cooperativas realizadas pelos usuários.

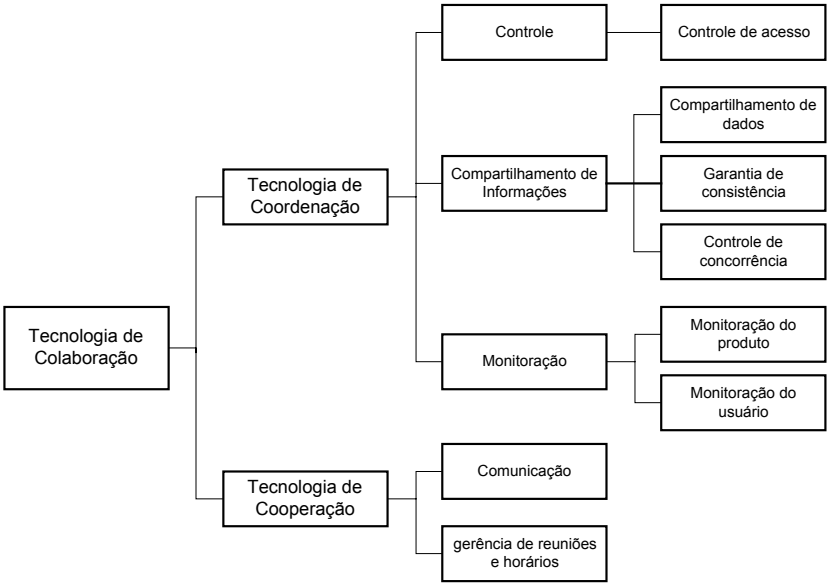


FIGURA 2.6 – Arquitetura de ambientes de desenvolvimento cooperativo de software

Ainda no estudo de Vessey [VES 95], são definidos os três principais elementos de ADCS:

- ***Taskware***. Corresponde à execução da tarefa principal dos ADSs, ou seja, a capacidade de apoiar a construção dos modelos de engenharia de software;
- ***Teamware.*** Elemento do ADCS encarregado de apoiar a coordenação do processo de software, tanto em nível organizacional quanto dos usuários (membros dos grupos). Isso é obtido através do controle de acesso aos objetos de software, o compartilhamento de informações e a monitoração dos produtos e profissionais envolvidos no processo;
- ***Groupware***. Gerencia a comunicação explícita entre os membros dos grupos e provê mecanismos para organizar o trabalho dos grupos de trabalho.

A subseção seguinte traz as considerações finais sobre Desenvolvimento Cooperativo de Software.

2.4.4 Considerações Finais

Essa seção apresentou conceitos relativos ao desenvolvimento cooperativo de software: suporte por computador ao trabalho cooperativo, *groupware* e ambientes de desenvolvimento cooperativo de software. O DiSEN deverá suportar trabalho cooperativo.

A próxima seção apresenta alguns trabalhos relacionados ao DiSEN: o ambiente PROSOFT, o CAGIS PCE (*Cooperative Agent in a Global Information Space – Process Centered Environment*) e o PROSYT (*PROcess Support sYstem capable of Tolerating deviations*). Apresenta também as diferenças e similaridades em relação à nossa proposta.

2.5 Trabalhos Relacionados

Vários trabalhos relacionados ao presente existem na literatura. Nessa seção, apresentamos alguns deles, enfatizando suas diferenças e similaridades em relação à abordagem proposta.

O ambiente PROSOFT, conforme descrito em Reis [REI 2000], é um ambiente de desenvolvimento de software que tem como objetivo principal apoiar o desenvolvimento formal de software, fornecendo integração de dados, de controle e de apresentação entre suas ferramentas. As ferramentas do PROSOFT visam auxiliar o desenvolvedor de software desde a especificação de requisitos até a implementação do sistema. O uso de métodos formais é enfatizado na construção das ferramentas do ambiente PROSOFT com paradigma próprio construído dentro do projeto.

PROSOFT tem como principais extensões propostas o PROSOFT Cooperativo e o Gerenciador de Processos de Software. A primeira extensão foi especificada com o objetivo de suportar o uso concorrente de objetos de software (diagramas, códigos, etc.), permitindo que um mesmo objeto fosse manipulado por vários desenvolvedores. A segunda extensão proposta é composta por uma máquina de processos de software que coordena processos de software modelados e instanciados, permitindo modificação dinâmica dos mesmos, dentre outras características, e utiliza serviços fornecidos pelo PROSOFT Cooperativo [REI 2000].

O formalismo presente no PROSOFT não será considerado no DiSEN. Os modelos a serem elaborados tomarão como base a notação MDSODI. Um outro ponto previsto no DiSEN é a utilização da tecnologia de agentes na definição do ambiente.

Wang [WAN 2001] apresenta um ambiente centrado em processo, denominado CAGIS PCE que provê suporte ao processo para pessoas que estão distribuídas, trabalhando com atividades cooperativas bem como atividades individuais. O CAGIS PCE combina um sistema *workflow* com um sistema multi-agente e provê um framework para suportar evolução do processo e permitir a interação entre partes autônomas do processo (fragmentos do processo). O CAGIS PCE é composto de:

- **Sistema *Workflow* suportando Processo Móvel Distribuído**. O sistema *workflow* permite a um modelo *workflow* instanciado ser distribuído como diversos fragmentos do processo em diferentes *workspaces*. Um benefício de permitir diversas instâncias de um modelo *workflow* ser distribuído e fragmentado em diversos *workspaces* é a possibilidade de adaptar o *workflow* às condições ambientais locais.

- **Agentes de Software para suportar Processos Dinâmicos e Cooperativos**. Agentes de software tipicamente cuidam das atividades inter-*workspace* como atividades de negociação (por exemplo, sobre alocação de recursos), coordenação de artefatos e elementos *workflow* entre *workspaces*, geração de idéias (*brain-storming*), votação, etc. A arquitetura multi-agente, proposta por Wang, consiste de quatro elementos principais: agentes, *workspaces*, *agent meeting place* (onde os agentes se encontram e interagem) e repositórios.

- ***Agent-Workflow Glue Server***. O *Agent-Workflow Glue Server* facilita o modo de especificar como o sistema *workflow* e o sistema multi-agente devem interagir. O *glue model* define o relacionamento entre os elementos *workflow* e

agentes de software. O *Glue Server* proverá então suporte, de maneira que uma atividade *workflow* ativa um agente e vice-versa.

DiSEN também irá suportar desenvolvimento cooperativo de software e também utilizará agentes para determinadas atividades do ambiente. Porém, a definição dos agentes seguirá as especificações propostas pela FIPA com o objetivo de alcançar interoperabilidade entre as aplicações. Outro ponto importante a ser ressaltado é a camada dinâmica que está presente no DiSEN.

Cugola descreve em [CUG 99] o projeto e implementação de um sistema de suporte a processo (PROSYT) que pretende prover orientação na execução de processos de negócio e cooperação entre pessoas em um determinado local ou em locais geograficamente distribuídos. A base para o PROSYT é um framework baseado em eventos para a integração e cooperação dos componentes e o uso de agentes móveis. Deste modo, os componentes cooperam através do envio e recebimento de eventos e podem migrar de um *host* para outro *host* em uma rede local ou de longa distância. Os componentes que constituem o ambiente PROSYT são:

- Uma máquina de processo para cada repositório e um modelo de processo que representa o processo de negócio atual.
- Um *project browser* para cada usuário, usado para pesquisar e acessar as entidades (isto é, os artefatos, pastas, e repositórios) que compõem o processo de execução, invocando as operações que elas exportam.
- Um *login manager* que controla o *login* e *logout* dos usuários. Gerencia todas as informações sobre os agentes humanos que estão envolvidos no processo.
- A *administrative tool* usada pelo gerenciador de processo para alterar políticas de execução e adicionar ou remover usuários.
- As ferramentas invocadas pelo sistema para executar tarefas específicas do processo como editores e compiladores.
- A *monitoring tool* encarregada de monitorar a execução através da identificação de invocação de ações de desvio, junto com a *reconciling tool*, uma ferramenta que analisa o resultado deste monitoramento para suportar atividades de reconciliação.

No DiSEN, adotaremos o estilo arquitetural em camadas, pois, conforme descrito em Silva [SIL 2001], o estilo camadas oferece subsídios para melhor gerenciar a complexidade intrínseca aos sistemas distribuídos, proporcionando uma maior flexibilidade de desenvolvimento através da subdivisão do sistema em camadas especializadas, onde uma camada interage somente com suas camadas adjacentes. PROSYT é baseado na noção de objetos ativos. Um objeto ativo é uma entidade autônoma que executa uma tarefa específica de uma aplicação e pode se mover de um *host* a outro mantendo seu estado interno, ou seja, comporta-se como agentes móveis. Portanto PROSYT trata apenas a questão da mobilidade, não levando em consideração nenhuma outra característica dos agentes nem mesmo um padrão para a definição desses agentes. Como dito anteriormente, os agentes presentes no DiSEN seguirão as especificações propostas pela FIPA e possuirão características como autonomia e interação.

3 Especificação da Arquitetura do Ambiente para Desenvolvimento de Software Distribuído

Esse capítulo descreve a arquitetura do DiSEN, um ambiente que está sendo projetado para suportar a MDSODI e trabalho cooperativo. Incorporará também a tecnologia de agentes, sendo que, para o desenvolvimento desses agentes será seguido o padrão estabelecido pela FIPA. A arquitetura do DiSEN adotará o estilo camadas, sendo constituída por três camadas: dinâmica, aplicação e infra-estrutura. A camada dinâmica será responsável pelo gerenciamento da configuração do ambiente; a camada de aplicação irá suportar a MDSODI e conterá, entre outros elementos, o repositório para armazenamento dos dados necessários ao ambiente e a camada da infra-estrutura proverá suporte às tarefas de nomeação, persistência e concorrência e irá incorporar também o canal de comunicação. Os elementos constituintes das camadas de aplicação e da infra-estrutura se comunicarão através deste canal.

3.1 Arquitetura Proposta

O estilo arquitetural a ser adotado para a arquitetura proposta é baseado no estilo camadas, pois ele proporciona uma maior flexibilidade de desenvolvimento organizando o sistema em camadas hierárquicas [JAC 97]. Uma camada pode ser definida como um conjunto de sub-sistemas com o mesmo grau de generalidade. Tradicionalmente, as interações ocorrem somente entre camadas adjacentes [SIL 2001].

Assim, a arquitetura do DiSEN possuirá uma camada dinâmica que irá permitir, por exemplo, que sejam adicionados, excluídos, configurados ou modificados componentes de software e serviços de forma dinâmica, isto é, em tempo de execução; uma camada de aplicação que irá suportar a MDSODI, gerenciamento de *workspace*, gerenciamento de agentes e irá conter o(s) banco(s) de dados necessário(s) para armazenamento dos dados sobre o ambiente (todas as informações geradas durante o desenvolvimento de software e as informações da aplicação), bem como a base de conhecimento e, no futuro, as ontologias; e, finalmente, a camada da infra-estrutura que define o alicerce da arquitetura e proverá suporte às tarefas de persistência, nomeação e concorrência e, além disso, incorpora o canal de comunicação. Abaixo dessa camada, haverá uma camada de software básico do sistema que irá conter, por exemplo, interfaces para hardware específico, sistema operacional, memória compartilhada e comunicação. Porém não trataremos detalhes desta última camada em nossa arquitetura proposta.

O canal de comunicação é um elemento fundamental da arquitetura já que toda a comunicação entre os elementos constituintes da camada de aplicação e da camada da infra-estrutura será realizada através desse canal. É constituído pelo *middleware* e pelo *middle-agent*, sendo que, quando a comunicação envolver somente objetos, esta será feita pelo *middleware*, e quando os agentes estiverem envolvidos, devido às suas propriedades, a comunicação deverá ser gerenciada pelo *middle-agent*.

A figura 3.1 mostra a representação gráfica da arquitetura do DiSEN.

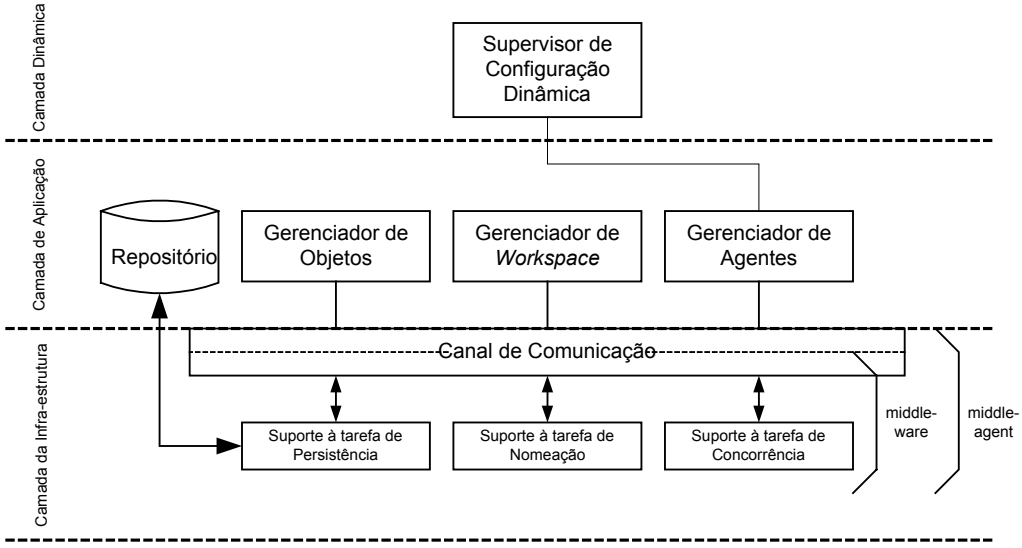


FIGURA 3.1 – Arquitetura de um ambiente de desenvolvimento de software distribuído baseada em agentes

A seguir uma breve descrição dos elementos da arquitetura proposta, para, nas seções subseqüentes (3.2 a 3.6) apresentar detalhes de cada um.

- **Supervisor de Configuração Dinâmica:** responsável pelo controle e gerenciamento da configuração do ambiente, bem como dos serviços que podem ser acrescentados ao ambiente em tempo de execução;
- **Gerenciador de Objetos:** responsável pelo controle e gerenciamento do ciclo de vida dos artefatos. Um artefato pode ser um diagrama, modelo, manual, código fonte, código objeto. Caso haja necessidade de ressaltar que um artefato é um programa o mesmo será tratado como produto de software;
- **Gerenciador de *Workspace*:** responsável pelo controle e gerenciamento da edição cooperativa de documentos e itens de software;
- **Gerenciador de Agentes:** responsável pela criação, registro, localização, migração e destruição de agentes;
- **Repositório:** responsável pelo armazenamento dos artefatos, dados das aplicações geradas pelo ADS, bem como o conhecimento necessário para a comunicação entre os agentes;
- **Canal de Comunicação:** responsável pela comunicação entre as camadas da infra-estrutura (*middleware* e *middle-agent*) e a camada de aplicação. Toda comunicação entre os elementos da arquitetura será por intermédio do canal de comunicação.

A camada da infra-estrutura será responsável pelo alicerce da arquitetura e oferecerá suporte às tarefas de (i) persistência: deverá fornecer uma interface única para os elementos acessarem os vários mecanismos de persistência, tais como, bases de dados relacionais, bases de dados OO, bases de conhecimento e simples arquivos; (ii) nomeação: deverá permitir que elementos possam referenciar e localizar outros elementos pelo nome, e (iii) concorrência deverá assegurar ao banco de dados que

somente um cliente por vez tenha acesso a um registro ou arquivo. Deve fazer o controle dos recursos compartilhados, evitando *deadlocks*.

Entenda-se que, para o acesso a este ambiente, existirá uma interface que permitirá, por exemplo, a inclusão de um novo projeto ou uma nova ferramenta no ambiente.

Assim, o DiSEN irá oferecer o suporte necessário ao desenvolvimento de software distribuído; poderá estar distribuído em locais geograficamente distintos; os desenvolvedores poderão estar trabalhando de forma cooperativa, com uma metodologia para desenvolvimento de software distribuído. Estão também sendo identificados alguns agentes para o ambiente (seção 3.8).

3.2 Supervisor de Configuração Dinâmica

Em razão da crescente complexidade das aplicações desenvolvidas onde mais e mais dispositivos são controlados por computador e da ampla difusão da Internet onde mais e mais computadores estão interconectados, devem-se levar em consideração dois aspectos principais: (i) é necessário considerar que um ambiente de desenvolvimento é dinâmico em relação à sua arquitetura, ou seja, seus elementos, sua organização e suas interconexões mudam durante a execução do sistema; (ii) deve ser considerado também que novos serviços podem ser acrescentados ao ambiente também de forma dinâmica. Assim, é necessário que o supervisor de configuração considere estes aspectos. Para tanto, o supervisor será subdividido em Supervisor de Configuração que terá como tarefa básica o controle e gerenciamento da configuração do ambiente e Supervisor de Serviços que deverá controlar e gerenciar os serviços que podem ser acrescentados ao ambiente de forma dinâmica. Essa subdivisão está ilustrada na figura 3.2. Essas tarefas do Supervisor de Configuração Dinâmica serão executadas pelo agente monitor (ver seção 3.8) que deverá ser capaz de acessar o gerenciador de agentes para realizá-las.

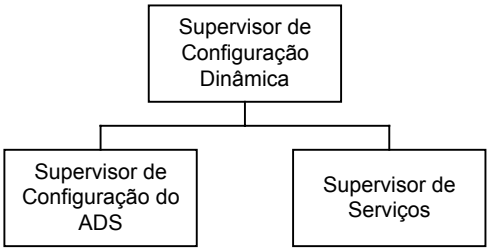


FIGURA 3.2 – Entidades agregadas ao supervisor de configuração dinâmica

3.2.1 Supervisor de Configuração

É necessário (re) configurar o ambiente de forma remota e, principalmente, dinâmica, isto é, em tempo de execução, pois devido à necessidade de alta disponibilidade do sistema, não é possível pará-lo ou interrompê-lo para uma reconfiguração. Isso inclui instalação e desinstalação dinâmica de componentes do sistema e configuração dinâmica de propriedades e interações de componentes via uma conexão internet, por exemplo. Além da configuração remota e dinâmica, o ambiente tem de fornecer informação sobre seu estado para o administrador, isto é, informação

sobre quais subsistemas da aplicação são atualmente supervisionados ou que métrica está atualmente ativa.

Os dados a respeito do estado do ambiente devem poder ser coletados de forma ativa quer seja seqüencial ou paralelamente. É interessante, por exemplo, que seja possível medir o consumo de CPU de certas tarefas que são executadas em vários *hosts* ao mesmo tempo. Um outro aspecto a ser observado é o de que algumas medidas podem ser realizadas em vários dias ou horas. Assim, deve ser possível também a emissão de relatórios temporários mesmo que seja apenas com dados parciais.

3.2.2 Supervisor de Serviços

O supervisor de serviços será responsável por, dinamicamente, descobrir, alocar e unir diferentes serviços. Deve também ser capaz de alterar atributos dos serviços, começar serviços em diferentes máquinas e rastrear a comunicação entre serviços e clientes, o que pode levar à necessidade de definição de políticas para disponibilização de serviços/funcionalidades. Estes serviços podem ser, por exemplo, telefone móvel, *laptop*, impressora. Serviços precisam ser capazes de se comunicar com outros serviços para executar tarefas mais complexas.

3.3 Gerenciador de Objetos

Uma das principais tarefas dos ambientes de desenvolvimento de software é armazenar, estruturar e controlar, além do próprio software, os artefatos gerados durante o desenvolvimento. Esses artefatos são, por natureza, mais complexos que os itens tratados por sistemas de banco de dados tradicionais, pois são mais estruturados e requerem operações complexas.

O Gerenciador de Objetos é constituído pelo gerenciador de acesso, gerenciador de atividades, gerenciador de recursos, gerenciador de artefatos, gerenciador de projetos, gerenciador de processos e gerenciador de versões e configurações, conforme mostra a figura 3.3.

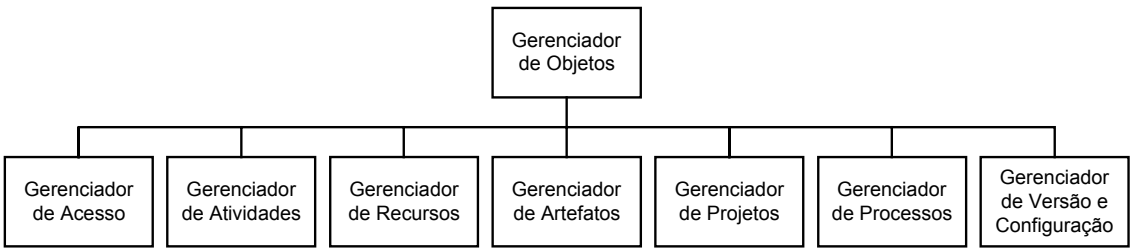


FIGURA 3.3 – Entidades agregadas ao gerenciador de objetos

3.3.1 Gerenciador de Acesso

O gerenciador de acesso será responsável pelo controle e gerenciamento das regras de acesso dos usuários às informações do ambiente. Para apoiar o compartilhamento de informações, é necessário que estejam definidos, no ambiente, as

funções desempenhadas pelos usuários e seus grupos. As funções dos usuários irão determinar as permissões de acesso para cada objeto em relação aos usuários do ambiente. Então, com base nessas permissões, o agente de acesso (ver seção 3.8) pode permitir que o usuário acesse ou não o ambiente, ou seja, o agente de acesso irá monitorar e gerenciar a segurança do ADS.

3.3.2 Gerenciador de Atividades

Atividades são os passos do processo de software. Atividades incorporam e implementam procedimentos, regras e políticas e têm como objetivo gerar ou modificar um dado conjunto de artefatos.

As atividades estão associadas a atores, ferramentas e artefatos. Uma atividade aloca recursos (por exemplo, máquinas e orçamento), é escalonada, monitorada e atribuída a atores, sendo que estes, por sua vez, podem utilizar ferramentas para executá-la. Uma atividade também pode ser executada somente por ferramentas automatizadas sem intervenção humana. Neste caso, a atividade é automática. A figura 3.4, adaptada de [LIM 98], mostra os elementos envolvidos em atividades do processo de software.

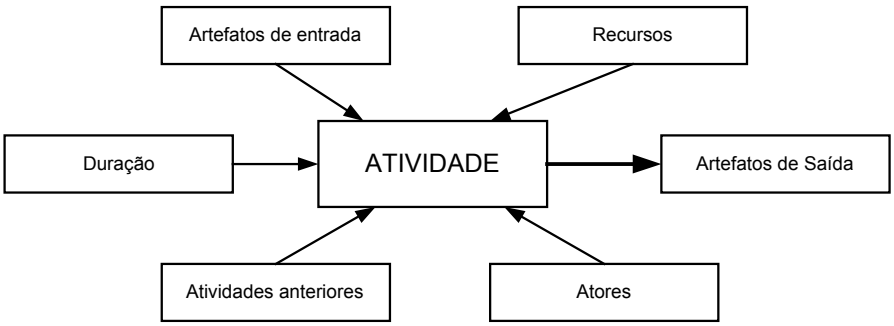


FIGURA 3.4 – Elementos envolvidos em atividades do processo de software

Toda atividade possui uma descrição, a qual pode especificar os artefatos necessários, as relações de dependência com outras atividades, a data de início e fim planejadas, os recursos a serem alocados e os atores responsáveis pela mesma.

Os principais atributos de uma atividade são:

- **Nome**: nome que a identifica no ambiente;
- **Descrição:** descrição textual da atividade;
- **Recursos:** consiste em um conjunto de recursos que devem ser alocados para que a atividade seja executada. Os recursos envolvem:
    - **Ferramentas:** consiste do conjunto de ferramentas que são necessárias para a execução da atividade;
    - **Materiais:** consiste em um conjunto de recursos materiais, por exemplo, computadores, impressoras, salas de reunião que são necessários para a execução da atividade;
- **Atividades anteriores:** consiste em um conjunto de atividades que devem ser executadas imediatamente antes da atividade em questão;

- **Artefatos entrada:** consiste em artefatos que são necessários para a execução da atividade;
- **Artefatos saída:** consiste em artefatos que serão produzidos em conseqüência da execução da atividade;
- **Data início (previsão):** data prevista para o início da execução da atividade;
- **Data término (previsão):** data prevista para o término da execução da atividade;
- **Data início (andamento):** data em que realmente a atividade foi iniciada;
- **Data término (andamento):** data em que realmente a atividade foi encerrada.

As atividades serão realizadas pelo agente de atividades (ver seção 3.8) que faz parte do gerenciador de agentes. Portanto, o gerenciador de atividades conterá somente o *script* para a realização das mesmas.

Um ator está relacionado às atividades de um processo e pode ser uma pessoa ou uma ferramenta automatizada (quando a atividade é automática). Diferentes atores terão percepções diferentes acerca do que acontece durante o processo de software. Um gerente, por exemplo, perceberá os aspectos de controle e alocação de recursos e cronogramas para atividades, enquanto um desenvolvedor perceberá as suas atividades como atribuições que devem ser feitas para produzir um resultado.

Os principais atributos de um ator são:

- **Nome**: nome que o identifica no ambiente;
- **Habilidades**: descrição do conjunto de habilidades que o ator possui para desenvolver software;
- **Custo por hora**: valor que deve ser contabilizado por hora de trabalho do ator, permitindo assim contabilização nos custos do projeto em que ele trabalha;
- **Cargo**: cargo ocupado pelo ator;
- **Agenda**: lista de atividades atribuídas ao ator em vários projetos de desenvolvimento de software da organização. Cada atividade da agenda do ator é composta de:
  - **Nome:** nome da atividade;
  - **Estado:** corresponde ao estado do andamento da atividade que pode ser: em andamento, suspensa, terminada;
  - **Data de início:** corresponde à data em que o ator iniciou a atividade;
  - **Data de término (previsão):** corresponde à data prevista para que o ator termine a atividade;
  - **Data de término:** corresponde à data em que o ator terminou a atividade;
- **Senha**: senha do ator. Essa senha vai permitir o controle de acesso aos objetos do ADS, ou seja, somente os atores devidamente autorizados poderão acessar as informações do ambiente.

- **Caixa postal**: conjunto de mensagens eletrônicas enviadas e recebidas pelo ator. Um dos requisitos necessários para ADS orientados a processo é a comunicação entre pessoas.

Um ator sempre ocupa um cargo quando realiza uma atividade. Um cargo é um conjunto de permissões e obrigações associadas a um objetivo funcional, segundo Lonchamp apud [LIM 98]. Projetista, programador e gerente são exemplos de cargos.

Os principais atributos de um cargo são:

- **Nome**: nome que o identifica no ambiente;
- **Descrição**: descrição que identifica as características do cargo;
- **Qualificações**: descrição dos requisitos necessários para que um determinado ator possa ocupar o cargo.

3.3.3 Gerenciador de Recursos

Os recursos necessários para a realização das atividades podem ser materiais (computador, impressora, sala de reunião) ou ferramentas (programas de computador destinados ao suporte ou automação de parte do trabalho relacionado com uma atividade). A atualização do estado de um recurso, bem como a busca por um recurso similar quando um recurso solicitado por uma determinada atividade estiver bloqueado ou em uso, será feita pelo agente de recursos (ver seção 3.8).

Os principais atributos de um recurso são:

- **Nome**: nome que o identifica no ambiente;
- **Descrição**: descrição que identifica as características técnicas como, por exemplo, modelo, marca, capacidade ou observações relacionadas ao recurso;
- **Estado**: identifica o estado atual do recurso que pode ser: livre (indica que o recurso está disponível para uso, ou seja, não está sendo utilizado por nenhuma atividade), bloqueado (indica que o recurso não pode ser alocado para nenhuma atividade. O recurso pode estar, por exemplo, com defeito), em uso (indica que o recurso está alocado para alguma atividade do processo);
- **Custo por hora**: valor que deve ser contabilizado por hora de uso do recurso;

Se o recurso for ferramenta, serão acrescentados os seguintes atributos:

- **Fabricante**: nome do fabricante da ferramenta;
- **Configuração:** configuração necessária para a instalação da ferramenta;
- **Versão**: versão da ferramenta.

3.3.4 Gerenciador de Artefatos

Um artefato é um produto criado ou modificado durante um processo. Tal produto é resultado de uma atividade e pode ser utilizado posteriormente como matéria-prima para aquela ou para outra atividade a fim de gerar novos produtos. Dessa forma,

uma atividade pode consumir produtos (de entrada) e gerar novos produtos (de saída). Os produtos são freqüentemente persistentes e possuem versões.

Os principais atributos de um artefato são:

- **Nome**: nome que o identifica no ambiente;
- **Descrição**: descrição que identifica as características do artefato;
- **Versão**: versão do artefato;
- **Data criação**: data em que o artefato foi criado;
- **Responsáveis**: nome dos responsáveis pelo artefato.

3.3.5 Gerenciador de Projetos

Segundo Conradi [CON 94], um projeto é a instância de um processo com objetivos e restrições específicos. Pode-se dizer que um projeto é um esforço para desenvolver um produto de software, ou seja, envolve uma estrutura organizacional, prazos, orçamentos, recursos e um processo de desenvolvimento.

Os principais atributos de um projeto são:

- **Nome:** nome que o identifica no ambiente;
- **Data de início:** data de início do projeto. Essa data deve ser automaticamente atualizada quando iniciar a primeira atividade relacionada ao projeto;
- **Data de término:** data de término do projeto.Essa data deve ser atualizada automaticamente quando a última atividade relacionada ao projeto terminar de ser executada;
- **Objetivos:** descrição textual dos objetivos do projeto;
- **Processo de desenvolvimento:** consiste no modelo de processo instanciado pelo projeto;
- **Métricas:** coleção automática de dados sobre o processo e o produto. O objetivo da aplicação de medição de software é fornecer aos gerentes e engenheiros de software um conjunto de informações tangíveis para planejar o projeto, realizar estimativas, gerenciar e controlar os projetos com maior precisão [FER 95]. Cada métrica é composta de:
  - **Nome:** nome que a identifica no projeto;
  - **Valor:** valor da medição;
  - **Unidade:** unidade de medição, por exemplo, linhas de código ou pontos por função;
  - **Método:** método utilizado para efetuar a medição, por exemplo, método de análise de pontos por função;
- **Estimativas:** estimativas são necessárias para o controle de projetos, alocação de recursos e distribuição de tarefas. É desejável que as estimativas de um projeto sejam realizadas com precisão maior que as estimativas de projetos anteriores. O armazenamento das estimativas permite que se possa compará-las

às métricas obtidas e que se possa "calibrá-las" para serem utilizadas em outros projetos. Uma estimativa é composta de:

- **Nome:** nome que a identifica no projeto;
- **Valor:** valor da estimativa;
- **Unidade:** unidade da estimativa.

No contexto deste trabalho estará sendo considerado, até o momento, somente um processo de desenvolvimento, a MDSODI.

3.3.6 Gerenciador de Processos

Informalmente, o processo de desenvolvimento de software (processo de software) pode ser compreendido como o conjunto de todas as atividades necessárias para transformar os requisitos do usuário em software, segundo Humphrey apud [LIM 98]. Um processo de software é formado por um conjunto de passos de processo parcialmente ordenados, relacionados com conjuntos de artefatos, pessoas, recursos, estruturas organizacionais e restrições e tem como objetivo produzir e manter os produtos de software finais requeridos.

O gerenciador de processos preocupa-se em construir, analisar e verificar modelos de processo, para isso obtendo informações durante a execução desse processo a fim de evoluir tais modelos para que possam ser usados posteriormente.

Os principais atributos de um processo são:

- **Nome:** nome que o identifica no ambiente;
- **Descrição:** descrição textual que informa as características e propósitos do processo que podem ser úteis na recuperação do processo para reutilização;
- **Versão:** versão do processo.

A MDSODI é uma instância de processo e nessa proposta será a única.

3.3.7 Gerenciador de Versão e Configuração

Os artefatos são sensíveis ao tempo. Isto é, durante o desenvolvimento de software é inevitável que alguns artefatos sejam produzidos em várias versões. Em um ambiente de desenvolvimento de software cooperativo, as modificações efetuadas nos artefatos devem ser gerenciadas de forma efetiva, pois os mesmos são compartilhados por muitos profissionais. Para efetivar o envolvimento de vários profissionais nas tarefas que constituem o ciclo de vida de software, um mecanismo de versões de artefatos deve estar disponível.

Uma versão representa um estado identificável de um artefato considerado pelo desenvolvedor como semanticamente significante.

Considerando um artefato composto em que os itens podem ser versionados, uma configuração associa exatamente uma versão para cada um dos itens.

Os principais atributos do gerenciador de versão e configuração são:

- **Nome:** nome que identifica o artefato no ambiente;
- **Descrição:** descrição textual do artefato;
- **Data da última alteração:** data em que foi executada a última alteração no artefato;
- **Ator da última alteração:** nome do ator que efetuou a última alteração no artefato;
- **Artefato ascendente:** nome do artefato que originou o artefato atual (se houver);
- **Estado**: identifica o estado atual do artefato, ou seja, o grau de maturidade do mesmo. O estado pode ser: preliminar (quando o artefato é criado sendo que, neste estado, ele é de propriedade exclusiva do ator que o criou), experimental (quando o artefato pode ser compartilhado por vários atores sendo que, neste estado, o artefato pode ser excluído, porém não pode ser modificado), padrão (quando a versão do artefato torna-se pública e não é permitido nem alterar nem excluir o artefato) ou obsoleto (quando o artefato não está mais sendo utilizado pelo ADS);
- **Histórico:** relação das últimas alterações efetuadas sobre o artefato, tornando possível, assim, retornar a um estado anterior do artefato.

3.4 Gerenciador de *Workspace*

As diferentes versões dos artefatos são mantidas em um repositório pelo gerenciador de versão. Pelo fato destas versões serem imutáveis, não é permitido aos desenvolvedores trabalharem diretamente no repositório. Eles têm de tirar uma cópia do documento, modificá-la e adicionar a cópia modificada no repositório [ASK 2001].

O Gerenciador de *Workspace* deve prover funcionalidades para criar um ou mais *workspace* de um repositório sendo que, no caso do DiSEN, isso será feito através do canal de comunicação e do suporte à tarefa de persistência, conforme mostra a figura 3.5. No caso mais simples, isso consiste em copiar um arquivo simples, porém mais freqüentemente uma configuração inteira é copiada do repositório para o *workspace* possibilitando ao desenvolvedor ter todos os módulos e documentos necessários para o sistema à sua disposição localmente. Assim, o desenvolvedor não tem de decidir o que deveria ser mantido globalmente no repositório e o que é necessário localmente para carregar suas mudanças. Além disso, ele está isolado das alterações das outras pessoas para o repositório – e outras pessoas estão isoladas das suas alterações. Isso significa que ele tem um controle completo do seu mundo e sabe exatamente o que foi alterado e por que.

Quando o desenvolvedor termina de efetuar suas modificações, ele precisa adicionar os módulos e documentos modificados no repositório. Essa operação, no caso mais simples, consiste em adicioná-las ao repositório, usando a funcionalidade do gerenciador de controle de versão. Entretanto, quando ele tem uma configuração completa no seu *workspace*, há provavelmente alguns arquivos que permaneceram inalterados e por essa razão não é preciso que sejam adicionados ao repositório. O Gerenciador de *Workspace* deve ser capaz de descobrir quais arquivos foram alterados e ter certeza de que todos esses – e somente esses – são adicionados no repositório [ASK

2001]. Isto poderia ser feito através da sincronização dos *workspaces* com o repositório, como sugerido por Wang [WAN 2000], porém esse trabalho não trará detalhes de como isso poderá ser implementado.

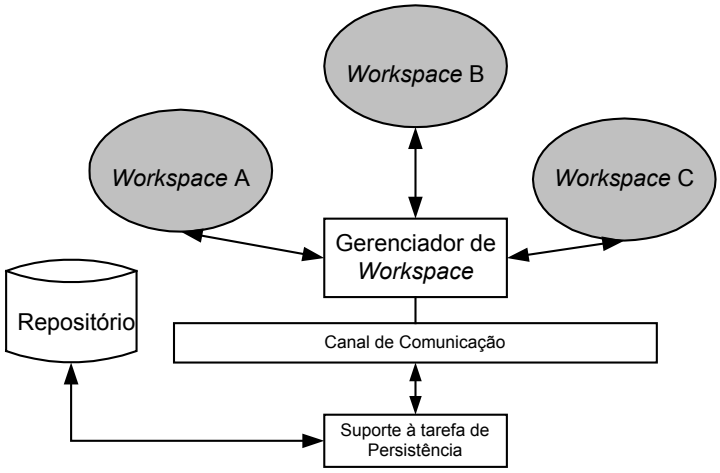


FIGURA 3.5 – Representação do gerenciador de *workspace*

Em cada *workspace* haverá agentes locais que auxiliarão durante a execução das atividades, agentes de interação que auxiliarão nos trabalhos cooperativos e agentes de sistema que auxiliarão na coleta de métricas do sistema (ver seção 3.8). Esses agentes interagirão com os demais enviando mensagens através do canal de comunicação.

3.5 Gerenciador de Agentes

O gerenciador de agentes é responsável pela criação, registro, localização, migração e destruição de agentes e consiste na região de agente (RA) e no servidor de ontologia, conforme mostra a figura 3.6.

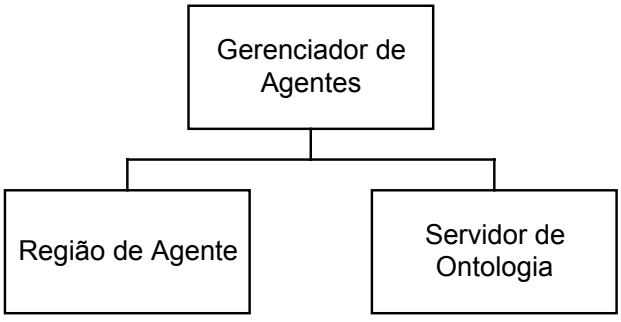


FIGURA 3.6 – Entidades agregadas ao gerenciador de agentes

Uma RA provê a infra-estrutura física na qual os agentes podem ser desenvolvidos. Uma RA consiste de coordenador de região (CR), servidor de página amarela (SPA) e dos agentes que nela residirão. É de responsabilidade do CR gerenciar as operações de uma RA. O SPA irá armazenar e tornar disponíveis informações sobre

serviços anunciados pelos agentes numa mesma RA. Uma descrição mais detalhada da RA e de seus elementos constituintes será dada na seção 3.7.

Cada RA tem somente um CR. Para que um agente possa residir numa região, ele deve ter-se registrado com o coordenador daquela região. Para que um agente possa acessar outros agentes no ambiente, ele deve solicitar ao CR. Os agentes podem anunciar e encontrar serviços de outros agentes através do SPA. Num mesmo local, quando há mais de uma região, a comunicação entre elas se dará através do canal de comunicação.

O DiSEN deverá ter um servidor de ontologia, que tem como função armazenar as ontologias necessárias para que os agentes possam se comunicar no ADS. A definição deste servidor deverá seguir a especificação Serviço de Ontologia FIPA [FIP 2001d], conforme descrito na seção 2.2.5, porém não está contida no escopo deste trabalho, ficando como sugestão para trabalhos futuros.

3.6 Canal de Comunicação

Toda troca de informação entre os elementos do DiSEN obrigatoriamente deverá ser por intermédio do canal de comunicação, que será responsável pela comunicação entre as camadas da infra-estrutura (*middleware* e *middle-agent*) e a camada de aplicação. Em uma aplicação que envolva somente objetos, a comunicação será estabelecida pelo *middleware*. Na literatura corrente podem ser encontradas várias, dentre as quais podemos citar: CORBA [OMG 2002], Jini [SUN 2002], DCOM [MIC 2002]. Já no caso em que envolver também agentes, esta será gerenciada pelo *middle-agent.* Neste caso, além dos serviços convencionais de nomeação, por exemplo, entende-se que outras propriedades, tais como: colaboração, negociação, coordenação e mobilidade, devem ser consideradas. Existem também na literatura várias estruturas que oferecem o suporte ao desenvolvimento de agentes. Alguns exemplos: JADE [BEL 99] [BEL 2000], Grasshopper [IKV 2002], ZEUS [NWA 99]. JADE e ZEUS foram desenvolvidos em conformidade com as especificações FIPA, porém a escolha de qual *middleware*/estrutura para desenvolvimento de agentes será de responsabilidade dos desenvolvedores, uma vez que deve ser levada em consideração a experiência destes e a disponibilidade/cultura na instituição.

3.7 Mapeamento dos Elementos FIPA para a Arquitetura Proposta

Nas seções anteriores, foram descritos os elementos FIPA que devem estar presentes em uma aplicação baseada em agentes a fim de permitir que os agentes interoperem e sejam gerenciados bem como os elementos que fazem parte do gerenciador de agentes do DiSEN. O passo seguinte é efetuar o mapeamento dos elementos FIPA para a arquitetura que está sendo proposta neste trabalho. A tabela 3.1 mostra esse mapeamento.

TABELA 3.1 – Mapeamento dos elementos FIPA para a arquitetura proposta

Elemento FIPA	Elemento da Arquitetura Proposta
Plataforma de Agente	Região de Agente
Sistema de Gerenciamento de Agente	Coordenador de Região
Facilitador de Diretório	Servidor de Página Amarela
Serviço de Transporte de Mensagem	Canal de Comunicação

Região de Agente

A arquitetura do DiSEN possuirá várias regiões que poderão estar em locais geograficamente distintos onde os agentes existirão. A figura 3.7 mostra a representação gráfica da RA.

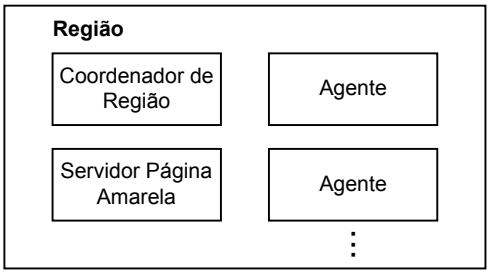


FIGURA 3.7 – Representação gráfica de região de agente

Os agentes existem fisicamente numa RA e utilizam as facilidades oferecidas pela RA para realizar suas funcionalidades. Nesse contexto, um agente, como um processo de software físico, tem um ciclo de vida que tem que ser gerenciado pela RA. Os estados que podem ser assumidos pelo agente durante o seu ciclo de vida podem ser: ativo, iniciado/esperando/suspenso, trânsito ou desconhecido, conforme mostra a figura 3.8. O ciclo de vida do agente é:

- **Limitado pela RA**: um agente é fisicamente gerenciado dentro de uma RA e o ciclo de vida de um agente estático é, por essa razão, sempre limitado a uma RA específica.
- **Independente de Aplicação:** o modelo de ciclo de vida é independente de qualquer aplicação e define somente os estados e as transições no ciclo de vida.
- **Orientado a Instância:** o agente descrito no modelo de ciclo de vida é assumido como uma instância, isto é, um agente tem nome único e é executado independentemente.
- **Único:** cada agente tem somente um estado no ciclo de vida da RA em qualquer tempo e dentro de somente uma RA.

Os principais atributos de uma região de agente são:

- **Nome**: nome da região;
- **Dinâmico**: indica se a região suporta registro dinâmico de agentes;
- **Mobilidade**: indica se a região suporta mobilidade;

- ***Profile* de comunicação (*transport-profile*)**: descrição das capacidades de comunicação da região.

Coordenador de Região

O CR é um elemento obrigatório de uma RA e existirá somente um coordenador por região, o qual atua como um facilitador para outros agentes dentro da sua região. O CR é responsável por gerenciar as operações de uma região, como criação e exclusão de agentes, decidir se um agente pode dinamicamente registrar-se numa RA e supervisiona a migração de agentes de uma região para outra (quando a mobilidade de agente é suportada pela região).

O CR mantém uma lista de todos os agentes que atualmente residem em uma região, juntamente com o estado atual do ciclo de vida do agente. Portanto, para que um agente possa residir em uma região ele deve registrar-se com o CR.

Um CR deve ser capaz de executar as seguintes funções:

- Registrar agente;
- Excluir registro de agente;
- Modificar agente;
- Pesquisar agente.

A seguir estão relacionadas as responsabilidades que um CR, como representante da RA, tem com respeito à entrega de mensagem em cada estado do ciclo de vida de um agente A figura 3.8 mostra a representação gráfica do ciclo de vida do agente.

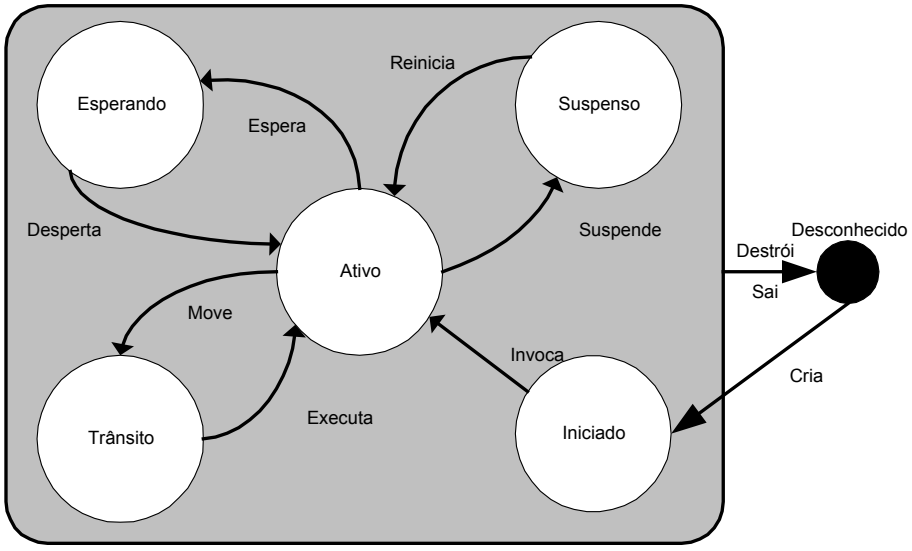


FIGURA 3.8 – Ciclo de vida do agente

- **Ativo**: o canal de comunicação entrega mensagens para o agente como normal.
- **Iniciado/Esperando/Suspenso**: o canal de comunicação armazena temporariamente as mensagens até que o agente retorne ao estado ativo ou

despacha as mensagens para uma nova localização (se essa nova localização estiver estabelecida para o agente).

- **Trânsito**: o canal de comunicação armazena temporariamente as mensagens até que o agente torne-se ativo (isto é, a função MOVE falha na RA original ou o agente foi iniciado com êxito na RA destino) ou despacha as mensagens para uma nova localização (se essa nova localização estiver estabelecida para o agente). Somente agentes móveis podem entrar no estado Trânsito. Isso assegura que um agente estacionário executa todas as suas instruções no nó onde foi invocado.
- **Desconhecido**: o canal de comunicação armazena temporariamente as mensagens ou rejeita-as, conforme especificações do mesmo.

As transições de estados dos agentes podem ser descritas como:

- **Cria**: criação ou instalação de um novo agente.
- **Invoca**: invocação de um novo agente.
- **Destrói**: término forçado de um agente. Isso pode somente ser iniciado pelo CR e não pode ser ignorado pelo agente.
- **Sai**: término natural de um agente. Isto pode ser ignorado pelo agente.
- **Suspende**: coloca o agente no estado suspenso. Isto pode ser iniciado pelo agente ou pelo CR.
- **Reinicia**: traz o agente de um estado suspenso. Isto pode somente ser iniciado pelo CR.
- **Espera**: coloca o agente em um estado de espera. Isto pode somente ser iniciado pelo agente.
- **Desperta**: traz o agente de um estado de espera. Isto somente pode ser iniciado pelo CR.
- **Move**: coloca o agente em um estado transitório (somente pode ser usado pelos agentes móveis). Isto somente pode ser iniciado pelo agente.
- **Executa**: traz o agente de um estado transitório (somente pode ser usado pelos agentes móveis). Isto somente pode ser iniciado pelo CR.

O principal atributo do CR é o ID do agente que é o coordenador, sendo que, através do relacionamento entre CR e agente é possível obter os agentes residentes numa determinada região bem como o estado do ciclo de vida desse agente.

Servidor de Página Amarela (SPA)

Será responsável por armazenar e tornar disponíveis informações sobre serviços anunciados pelos agentes numa mesma região, possibilitando a colaboração e delegação de tarefas.

Todo agente que deseja tornar público seus serviços para outros agentes deve requerer o registro de sua descrição de agente com o SPA. A qualquer tempo e por qualquer razão, o agente pode solicitar ao SPA a modificação de sua descrição de agente bem como a sua exclusão. Um agente pode pesquisar o SPA em busca de

informação sobre um determinado serviço. Porém, o SPA não garante a validade da resposta dada ao agente, pois não é imposta nenhuma restrição na informação que pode ser registrada no SPA.

As seguintes funções são suportadas pelo SPA:

- Registrar serviços;
- Excluir registro serviços;
- Modificar serviços;
- Pesquisar determinado serviço.

Os principais atributos do SPA são:

- **ID do agente**: identificador do agente;
- **Protocolo**: lista de protocolos de interação suportados pelo agente;
- **Ontologia**: lista de ontologias conhecidas pelo agente;
- **Linguagem**: lista de conteúdo de linguagem conhecido pelo agente.

Canal de Comunicação

O canal de comunicação já foi descrito na seção 3.6.

Agente

Um agente é o ator fundamental em uma RA. Pode combinar uma ou mais capacidades de serviços num modelo de execução integrado e único que pode incluir acesso a um software externo, usuários humanos ou facilidades de comunicações. Os agentes podem acessar um software, por exemplo, para adicionar novos serviços, adquirir novos protocolos de comunicação, adquirir novos protocolos/algoritmos de segurança, adquirir novos protocolos de negociação, acessar ferramentas que suportam migração, etc.

Cada agente do ambiente deve possuir um identificador de agente (ID) que deve ser único e imutável, não podendo ser trocado durante o tempo de vida do agente. Um agente pode ser registrado com um número de endereços de transporte nos quais ele pode ser contatado.

Os principais atributos do agente são:

- **ID do agente**: identificador do agente ;
- **Descrição**: descrição do agente;
- **Endereço**: seqüência ordenada de endereços de transporte onde o agente pode ser contatado. A ordem implica a relação de preferência do agente para receber mensagens naquele endereço;
- **Estado**: estado do ciclo de vida do agente (iniciado, ativo, suspenso, esperando, trânsito);
- **Serviços**: lista de serviços suportados pelo agente.

Um serviço está associado a um agente e é definido em termos de um conjunto de ações suportadas pelo agente. Cada ação define uma interação entre o serviço e o agente usando o serviço [FIP 2001b].

Os principais atributos de um serviço são:

- **Nome**: nome do serviço;
- **Tipo**: tipo do serviço;
- **Protocolo**: lista de protocolos de interação suportados pelo serviço;
- **Ontologia**: lista de ontologias suportadas pelo serviço;
- **Linguagem**: lista de linguagens de conteúdo suportadas pelo serviço
- **Propriedades**: lista de propriedades que descrevem o serviço.

3.8 Agentes do Ambiente

Em nossa proposta, um agente será uma entidade de software autônoma criado pelo usuário e que atuará em nome do usuário (ou de algum outro agente). Será configurado para alcançar um objetivo e possuirá as propriedades de autonomia, interação, reatividade e proatividade. Dentre os elementos identificados no DiSEN, definiremos, a seguir, os que serão agentes.

O Gerenciador de *Workspace* será responsável por administrar um conjunto de *workspaces*. Em cada *workspace* haverá **agentes locais**, **agentes de interação** e **agentes de sistema** [WAN 2000a]. Os **agentes locais** darão assistência aos atores durante a execução de suas tarefas, atuando como secretárias pessoais. Os **agentes de interação** ajudarão os atores em seus trabalhos cooperativos. Podem ser vistos como agentes de processos compartilhados e incluem 3 subclasses: agentes de comunicação são usados para trazer mensagens ou dados de um agente para outro agente situado em uma mesma região ou em uma outra região; agentes de negociação ajudarão outros agentes a alcançar um entendimento (acordo); agentes de mediação serão usados para ajudar em um processo de negociação entre agentes para alcançar um entendimento, fazendo uso de experiência prévia para sugerir soluções. Os **agentes de sistema** serão responsáveis por coletar as métricas durante o andamento do processo de desenvolvimento. Essas métricas deverão ser comparadas com as estimativas de cada projeto a fim de permitir a detecção de falhas no andamento do projeto e o seu possível ajuste.

O Gerenciador de Acesso é uma das partes constituintes do Gerenciador de Objetos e será responsável por controlar o acesso ao ambiente. Essa tarefa será executada pelo **agente de acesso** que irá monitorar e gerenciar a segurança do ADS, verificando quem está necessitando de acesso ao ambiente e qual a sua permissão de acesso.

Devido à necessidade de alta disponibilidade do ambiente, não sendo possível pará-lo ou interrompê-lo para uma reconfiguração ou inclusão/exclusão de um serviço, as tarefas do Supervisor de Configuração Dinâmica serão executadas pelo **agente monitor**. Este agente será responsável pela camada dinâmica da arquitetura e será capaz de acessar o gerenciador de agentes na camada de aplicação para a realização de suas tarefas.

O **agente de atividades** será responsável por controlar a execução das atividades de um processo de software. Este agente deve consultar o **agente de recursos** a fim de verificar se o recurso necessário para a execução da atividade está livre. O agente de atividades deverá também atualizar a data de início de andamento da atividade quando esta for iniciada e quando a atividade for encerrada, deverá atualizar a sua data de término. Na medida em que as atividades vão sendo realizadas, o **agente de atividades** deverá, também, manter atualizado o estado da atividade na agenda de cada ator. E, quando a última atividade relacionada a um projeto for executada, o agente deverá atualizar a data de término do projeto. O **agente de atividades** deverá ser capaz de informar quais os artefatos resultantes de uma determinada atividade, para isso deverá acessar o gerenciador de artefatos através do gerenciador de objetos na camada de aplicação.

O Gerenciamento dos Recursos do sistema será feito pelo **agente de recursos** que será responsável por manter atualizado o estado de cada recurso quando este estiver livre, bloqueado ou em uso. Quando um recurso estiver sendo solicitado pelo agente de atividades e tal recurso está em uso por outra atividade, o agente de recursos deverá verificar se há disponibilidade de um recurso com a mesma descrição do solicitado e, em caso positivo, efetuar a sua alocação para o agente de atividades.

O **agente de ontologia** será responsável por acessar o servidor de ontologia (que estará embutido no repositório), sendo que este acesso será feito através do envio de mensagem para o canal de comunicação.

3.9 Exemplo de Visualização de Comunicação no DiSEN

O DiSEN é um ambiente que consiste de regiões. Cada região terá um coordenador que é um agente que atua como um facilitador para outros agentes dentro da sua região, um ou mais SPA e agentes, sendo que esses agentes podem ser identificados como estando dentro de uma região se eles tiverem se registrado com o CR. Os agentes usarão os serviços de coordenadores para acessar outros agentes no ambiente. Também, os agentes podem anunciar serviços e encontrar serviços de outros agentes através do SPA.

O ADS poderá estar fisicamente distribuído, ou seja, pode-se ter um ou vários atores trabalhando em um local A, outro(s) em um local B e outro(s) em um local C. Cada local poderia ser visto como uma LAN (*Local Area Network*), contendo pelo menos uma RA, pelo menos um *workspace*, o gerenciador de objetos, o supervisor de configuração dinâmica, o canal de comunicação e o repositório. A comunicação entre regiões num mesmo local se dará através do canal de comunicação. As LAN's de cada local poderiam estar interligadas através de uma WAN (*Wide Area Network*), possibilitando a comunicação entre locais distintos.

Os agentes irão se comunicar com o CR sempre através do canal de comunicação. Por exemplo, um agente de atividades constata que, para executar uma atividade, é necessário um determinado recurso. Ele então envia uma mensagem para o CR a fim de verificar a disponibilidade de tal recurso. O CR verifica se há um agente de recursos naquela região. Em caso positivo, o agente de recursos é consultado e é retornada uma mensagem para o agente de atividades com a resposta à sua solicitação.

Caso contrário, o CR comunica-se com outra região até conseguir encontrar um agente de recursos.

Caso seja necessário armazenar algum objeto no repositório ou buscar informações a respeito de algum objeto armazenado, a comunicação também se dará pelo canal de comunicação que acionará o suporte à tarefa de persistência e este, por sua vez, irá acessar o repositório para atender à solicitação feita. Além disso, se for necessária a reconfiguração do ADS ou o acréscimo/solicitação de algum serviço, o agente monitor enviará uma mensagem para o gerenciador de agentes que envia uma mensagem para o canal de comunicação a fim de executar a tarefa solicitada conforme mostra a figura 3.9.

O canal de comunicação será, então, o elemento do DiSEN que será responsável por gerenciar toda a comunicação num local. Como se trata de um ambiente distribuído, este canal permitirá que um elemento do ambiente que pode ser um objeto ou um agente invoque transparentemente outro elemento localizado local ou remotamente, sendo que, neste último caso, a comunicação poderia ser feita através de uma WAN. O canal intercepta a chamada e é responsável por encontrar um elemento no ambiente que possa implementar a requisição, passar-lhe os parâmetros e retornar seus resultados.

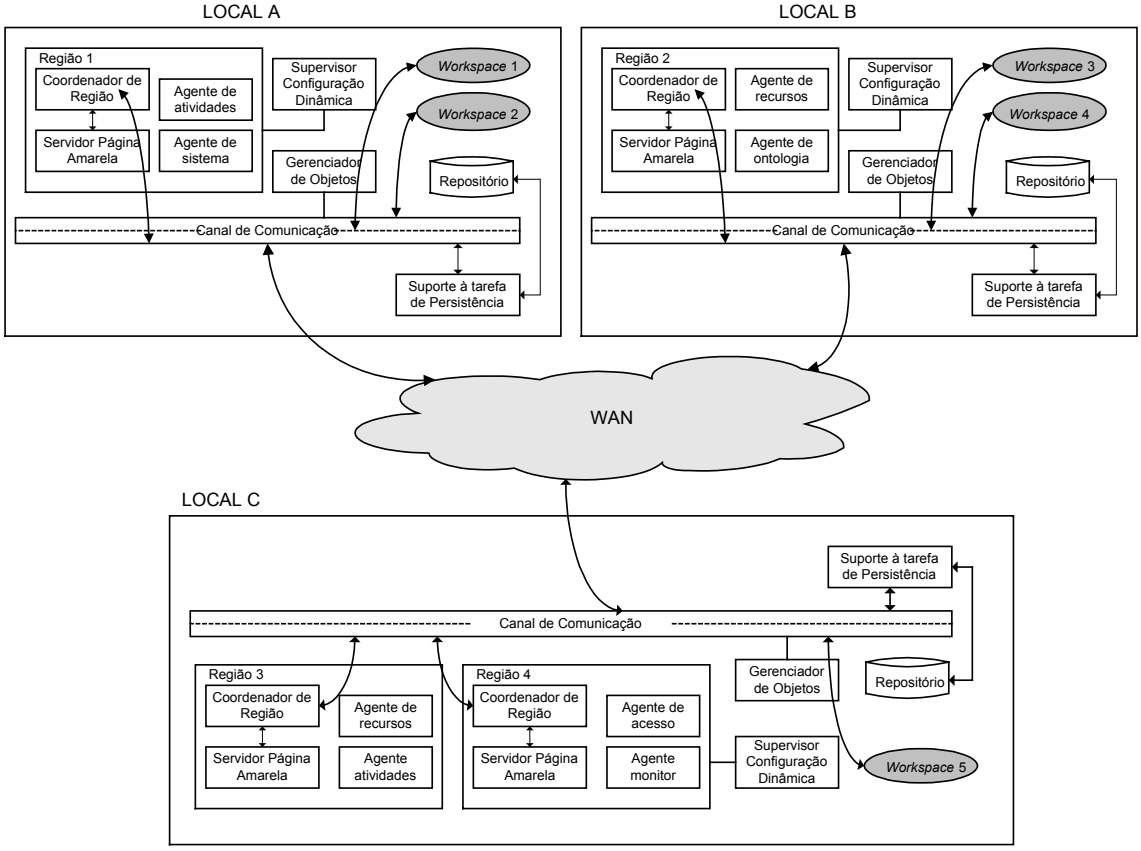


FIGURA 3.9 – Um exemplo de visualização de comunicação entre as partes constituintes do DiSEN

3.10 Considerações Finais

Este capítulo apresentou o projeto da arquitetura de um ambiente de desenvolvimento de software distribuído. Essa arquitetura é baseada no estilo camadas, sendo composta pelas camadas dinâmica, de aplicação e da infra-estrutura. A camada dinâmica é responsável pelo controle e gerenciamento da configuração do ambiente, bem como dos serviços que podem ser acrescentados ao ambiente em tempo de execução. A camada de aplicação possui um gerenciador de objetos, um gerenciador de *workspace* e um gerenciador de agentes, contendo também o repositório para armazenamento das informações necessárias ao ambiente. E, a camada da infra-estrutura provê funcionalidades e suporte para as tarefas de nomeação, de persistência e de concorrência e, também, incorpora o canal de comunicação. A comunicação entre as camadas de aplicação e da infra-estrutura se dá através do canal de comunicação, ou seja, este canal é responsável pela comunicação entre os elementos da arquitetura.

No próximo capítulo serão apresentados aspectos de comunicação e cooperação que devem ser considerados na interação entre os agentes.

4 Aspectos de Comunicação e Cooperação entre Agentes

Neste ponto, já se tem a definição das propriedades de um agente, ou seja, das propriedades que possibilitam a identificação de uma entidade de software como sendo um agente. Como um agente não vai existir sozinho em uma aplicação, é necessário que eles sejam capazes de cooperar e comunicar-se uns com os outros a fim de atingir um determinado objetivo. Para que isso aconteça primeiramente é necessário definir a arquitetura do agente.

A arquitetura de um agente mostra como ele está implementado em relação às suas propriedades, sua estrutura e como os módulos que o compõem podem interagir, garantindo sua funcionalidade. Em suma, a arquitetura de um agente especifica sua estrutura e funcionamento.

De acordo com Wooldridge [WOO 95], a arquitetura de um agente pode ser cognitiva ou deliberativa, reativa ou híbrida. Uma arquitetura deliberativa contém um modelo simbólico do mundo representado explicitamente, e as decisões sobre que ações tomar são feitas via raciocínio lógico, baseado em reconhecimento de padrões e manipulação simbólica. Em uma arquitetura reativa, os agentes somente reagem às ações que ocorrem no ambiente, não estando incluído nenhum tipo de modelo do mundo simbólico ou raciocínio simbólico complexo. Em uma arquitetura híbrida os agentes são dotados de comportamento reativo com relação aos eventos que ocorrem no ambiente e comportamento deliberativo onde existe uma definição simbólica do mundo para a tomada de decisões.

4.1 Métodos de Comunicação

Em aplicações baseadas em agentes, a comunicação, a cooperação e a negociação entre eles são de suma importância para que possam atingir as metas desejadas. Para que isso ocorra deve haver algum meio de comunicação entre eles a fim de realizarem um intercâmbio de informações e interagirem entre si.

Segundo Brenner [BRE 98], os principais métodos de comunicação são: sistema de quadro-negro, passagem de mensagem e sistema federativo que serão brevemente discutidos a seguir.

Sistema de Quadro-Negro (*Blackboard System*)

O sistema de quadro-negro surgiu antes das aplicações baseadas em agentes, utilizando o conceito de compartilhamento de memória. Nesse tipo de sistema não há uma comunicação direta entre os agentes, mas sim um espaço de memória ou área de trabalho através do qual os agentes podem trocar informações, escrevendo e lendo as mensagens. O quadro-negro pode ser utilizado para compartilhamento de tarefas e resultados. A figura 4.1 ilustra a estrutura desse modelo.

Uma aplicação baseada em agentes pode conter mais de um quadro-negro e um agente para acessá-los, deve registrar-se num *site* central responsável pela administração de todos os agentes autorizando a entrada dos registrados e bloqueando a dos demais.

A comunicação inicia-se quando um agente escreve alguma informação no quadro-negro, disponibilizando-a a todos os outros agentes que têm acesso a ele. Embora o agente tenha acesso a todas as informações que são deixadas, ele apenas lê as que lhe convier, fazendo uso de um filtro de informações ou escolhendo as mensagens provenientes de agentes com os quais mantém contato.

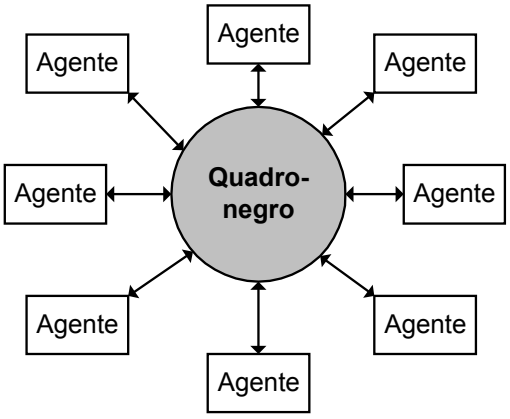


FIGURA 4.1 – Estrutura de sistemas de quadro-negro

Os agentes são responsáveis pela resolução dos subproblemas, sendo as respostas disponibilizadas no quadro-negro. Um agente que acessa a área de trabalho em intervalos constantes de tempo pode obter informações a respeito do andamento das atividades dos outros agentes, com isso, torna-se capaz de verificar possíveis situações de conflitos ou soluções erradas a fim de solucioná-las.

Por permitir que qualquer agente escreva e leia mensagens no quadro-negro, este tipo de sistema demanda grandes recursos de rede e não tem controle sobre a veracidade das soluções e para sistemas em tempo real, a espera por uma resposta é muito grande.

Para otimização desse modelo de comunicação, desenvolveu-se o quadro-negro estendido que possui regiões ou níveis de informações que facilitam a busca das mesmas pelo agente, pois ele se registra apenas nas regiões que contêm as informações que lhe interessam. Para se ter um maior controle de qualidade nas soluções das tarefas, foi introduzido o agente moderador. Ele escreve no quadro-negro os subproblemas e, através de sua base de conhecimentos, seleciona os agentes mais apropriados para solucioná-los.

A figura 4.2 ilustra essa estrutura. Qualquer agente pode usar o quadro-negro para ler um subproblema. Se o agente tem interesse em um subproblema específico, isto deve ser informado usando-se uma base de dados para criação do KSAR (*Knowledge Source Activation Record*). O moderador seleciona o agente mais apropriado e usa o quadro-negro para informar ao agente que ele recebeu a tarefa de resolver o subproblema.

O despachante informa, dentre os agentes registrados no quadro-negro, aqueles que são mais aptos a solucionarem determinado subproblema e também sobre a chegada de novas informações, evitando assim que os agentes tenham que estar sempre fazendo consultas para verificar se há ou não novas mensagens.

A seção seguinte descreve o segundo método de comunicação entre agentes - passagem de mensagem.

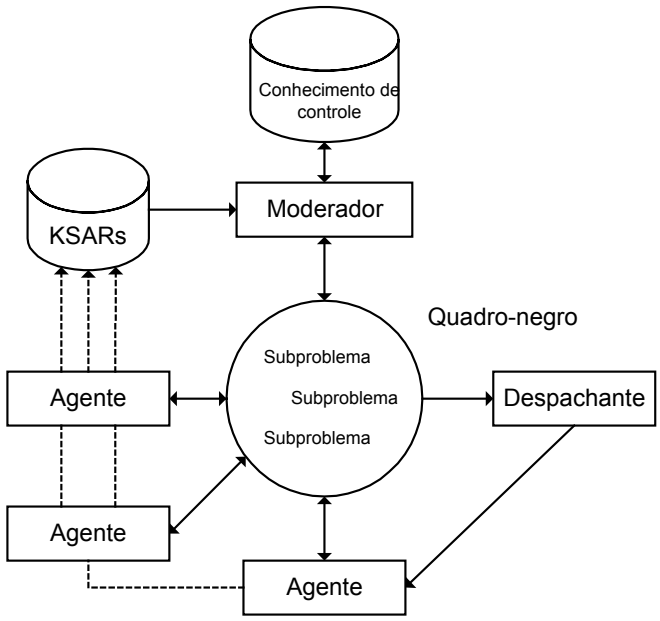


FIGURA 4.2 – Estrutura de quadro-negro estendida

Passagem de Mensagem (*Message Passing*)

A comunicação usando mensagens forma uma base flexível para a implementação de estratégias de coordenação complexas. Neste modelo, as mensagens que os agentes trocam uns com os outros podem ser usadas para estabelecer mecanismos de comunicação e cooperação usando protocolos definidos.

Neste tipo de modelo, a comunicação entre os agentes é feita de maneira direta com os agentes enviando e recebendo mensagens assincronamente, utilizando um protocolo de comunicação definido que será responsável pela especificação do formato das mensagens, do envio e recebimento das mesmas e da definição de uma linguagem de comunicação.

A figura 4.3 ilustra o princípio básico de um sistema de agentes baseado em mensagens. Para que o envio das mensagens seja feito de maneira correta, os agentes devem conhecer os endereços uns dos outros [BRE 98]. Um agente, também chamado de emissor, transfere uma mensagem específica para outro agente, o receptor. Ao contrário dos sistemas de quadro-negro, as mensagens são trocadas diretamente entre dois agentes. Um agente só está apto a ler as mensagens endereçadas a ele, a menos que a mensagem seja enviada a todos os agentes, conhecida como *broadcasting*.

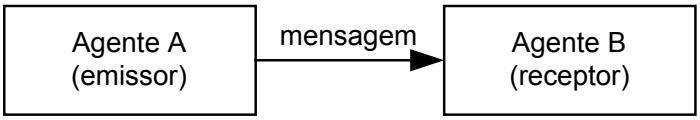


FIGURA 4.3 – Princípio de transmissão de mensagem

Todos os agentes devem estar aptos a entender o conteúdo semântico das mensagens. O que se espera como resposta a uma mensagem não é apenas verdadeiro

ou falso, mas sim que o agente receptor da mensagem tome ações que mudem o ambiente a fim de agir segundo os objetivos que o outro agente tinha em mente quando enviou a mensagem.

Este tipo de modelo de comunicação possui tempo de resposta bem menor que o do quadro-negro, porém sua implementação é muito difícil devido à grande quantidade de mensagens trocadas.

No padrão FIPA, os agentes comunicam-se uns com os outros através do envio de mensagens. Dois aspectos fundamentais da comunicação entre agentes que devem ser levados em consideração são: a estrutura da mensagem e o transporte de mensagem [FIP 2001b].

Estrutura da Mensagem

A estrutura de uma mensagem, conforme mostrada na figura 4.4, é escrita em uma linguagem de comunicação de agente como FIPA ACL. O conteúdo da mensagem é expresso em uma linguagem de conteúdo como, por exemplo, KIF (*Knowledge Interchange Format*). O conteúdo da linguagem pode referenciar uma ontologia a qual fundamenta os conceitos que estão sendo descritos no conteúdo. As mensagens contêm também os nomes do remetente e do destinatário, expressos como nome-de-agente que é um identificador único para um agente (ID do agente). Uma mensagem pode recursivamente conter outras mensagens.

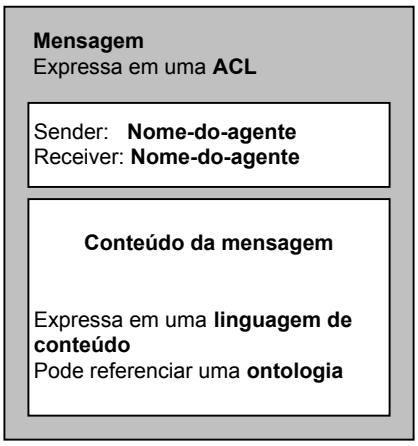


FIGURA 4.4 – Estrutura de uma mensagem

Transporte de Mensagem

Quando uma mensagem é enviada, ela é transformada numa carga (*payload*) e incluída em um transporte de mensagem (*transport-message*). A carga é codificada usando uma representação de codificação apropriada para o transporte. Por exemplo, se a mensagem vai ser enviada através de um transporte banda larga (como uma conexão sem fio) uma representação de bit pode ser usada ao invés de uma representação caracter a fim de permitir uma transmissão mais eficiente.

O transporte de mensagem é a carga mais o envelope. O envelope inclui as descrições de transporte do remetente e do destinatário. As descrições de transporte contêm a informação sobre como enviar a mensagem (como por exemplo, se IIOP,

SMTP ou HTTP, para qual endereço). O envelope pode conter também informações adicionais, como representação da codificação, segurança dos dados relacionados e outras especificações de dados que necessitam ser visíveis para o transporte. A figura 4.5 mostra como uma mensagem é transformada em um transporte de mensagem.

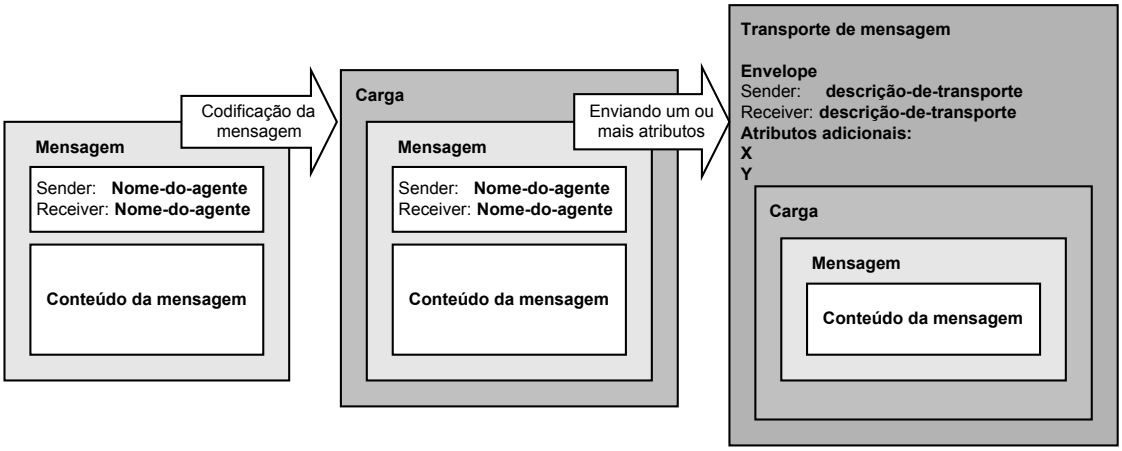


FIGURA 4.5 – Transformação de uma mensagem em um transporte de mensagem

Sistema Federativo (*Federal System*)

Este modelo de comunicação pode ser tido como uma otimização do modelo de passagem de mensagem (*message passing*). Seu desenvolvimento foi com base no fato de que muitas mensagens trocadas pelos agentes continham informações desnecessárias, causando um congestionamento no fluxo da rede, o que poderia acarretar numa demora no envio de mensagens que continham informações realmente necessárias para que os agentes e o sistema como um todo atingissem suas metas.

Os agentes do sistema são divididos em federações segundo um critério de agrupamento escolhido. Cada federação possui agentes denominados facilitadores, incumbidos de selecionar as mensagens destinadas apenas a agentes de sua federação. Uma mensagem enviada a todos os agentes (*broadcasting*) chega aos facilitadores, e estes apenas enviam a mensagem se verificarem que pode ser útil aos agentes de sua federação, diminuindo o fluxo de mensagens no sistema.

Para que a comunicação entre os agentes seja possível, é necessário que exista um entendimento por ambas as partes, ou seja, é necessário que seja definido um protocolo de comunicação. A seção seguinte trata dos protocolos KQML (*Knowledge and Query Manipulation Language*) e FIPA ACL.

4.2 Protocolos de Comunicação

Os agentes de software precisam ser capazes de interagir e se comunicar com outros agentes e com o ambiente para que seja possível a cooperação e o compartilhamento de conhecimento. Para isso, é necessário que eles tenham uma linguagem de comunicação comum, ou seja, uma linguagem de comunicação de agente (ACL - *agent communication language*) com sintaxe, semântica e pragmática

precisamente definidas que servirão de base para comunicação entre agentes de software projetados e desenvolvidos independentemente.

Um dos protocolos de comunicação mais utilizados em sistemas multi-agentes é KQML, que foi desenvolvido como parte do projeto KSE (*American Knowledge Sharing Efforts*) na Universidade de *Maryland* [FIN 93] [LAB 97].

KQML é uma linguagem e um protocolo que dão suporte aos agentes para que estes identifiquem, estabeleçam conexão e troquem mensagens com outros agentes. O conteúdo semântico de uma mensagem não é especificado em maiores detalhes em KQML, pois como o padrão é aberto, várias linguagens podem ser usadas para trocar conhecimento e podem ser integradas a uma mensagem KQML [BRE 98].

Esta linguagem é baseada na teoria dos atos de fala [BOR 2001] e enfoca um extenso conjunto de mensagens pré-definidas, chamadas *performatives* que definem as possíveis operações que os agentes podem executar. Toda mensagem KQML tem a seguinte estrutura básica:

(<Performative>
:content<statement/speechact>
:sender<name>
:receiver<name>
:language<text>
:ontology<text>
)

Outro protocolo de comunicação é FIPA ACL. Esse protocolo especifica um padrão para as mensagens através do estabelecimento da codificação, semântica e pragmática das mensagens. Esse padrão especifica que as mensagens devem ser codificadas em forma textual, pois diferentes agentes podem executar em diferentes plataformas e diferentes tecnologias de rede. A sintaxe da ACL está muito perto da linguagem de comunicação KQML. Porém, apesar da similaridade sintática, há diferenças fundamentais entre KQML e ACL, e a mais evidente é a existência de uma semântica formal para ACL que deve eliminar qualquer ambigüidade e confusão no uso da linguagem [BEL 99].

Uma mensagem FIPA ACL contém um conjunto de um ou mais elementos de mensagem. O número de elementos necessários para se efetivar uma comunicação entre agentes varia de acordo com a situação, sendo que o único elemento obrigatório em todas as mensagens ACL é a *performative*, embora seja esperado que a maioria das mensagens também possua os elementos *sender*, *receiver* e *content*. A seguir são listados os elementos de mensagem FIPA ACL, porém sem considerar suas codificações específicas em uma implementação [FIP 2001c].

- ***Performative***: denota o tipo de ato comunicativo da mensagem ACL. A especificação Biblioteca de Atos Comunicativos FIPA [FIP 2002a] propõe uma biblioteca de atos comunicativos que podem ser utilizados pelos desenvolvedores.
- ***Sender***: denota a identidade de quem está enviando a mensagem, isto é, o nome do agente do ato comunicativo.
- ***Receiver***: denota a identidade do destinatário da mensagem que pode ser um agente ou conjunto de agentes, quando a mensagem é *multicast*.

- ***Reply-to***: este elemento indica que mensagens subseqüentes nesta conversação serão direcionadas para o agente nomeado no elemento *reply-to*, em vez do agente nomeado no elemento *sender*.
- ***Content***: denota o conteúdo da mensagem; equivalentemente denota o objeto da ação.
- ***Language***: denota a linguagem na qual o elemento *content* é expresso.
- ***Encoding***: denota a codificação da expressão da linguagem de conteúdo (*content*);
- ***Ontology***: denota a(s) ontologia(s) usada(s) para dar um significado aos símbolos na expressão do conteúdo.
- ***Protocol***: denota o protocolo de interação que o agente emissor está empregando com esta mensagem ACL.
- ***Conversation-id***: introduz uma expressão (um identificador de conversação) que é usado para identificar a seqüência de atos comunicativos que juntos formam a conversação.
- ***Reply-with***: apresenta uma expressão que será usada pelo agente que irá responder a esta mensagem.

  Exemplo: Se um agente X envia uma mensagem para o agente T que contém
  Reply-with <expr>
  O agente T responderá com uma mensagem contendo:
  In-reply-to <expr>
- ***In-Reply-to***: denota uma expressão que referencia uma mensagem anterior para a qual esta mensagem é uma resposta. Ver exemplo acima.
- ***Reply-by***: denota uma expressão de horário e/ou data que indica o último momento em que o agente emissor gostaria de receber uma resposta.

Em nossa proposta, a comunicação entre agentes será através de passagem de mensagem, onde FIPA ACL é a linguagem para representar mensagens de acordo com os padrões estabelecidos pela FIPA.

Segundo Brenner [BRE 98], os protocolos de cooperação mais utilizados em uma aplicação baseada em agentes são Planejamento Global Parcial e Sistema de Contrato de Rede. As especificações FIPA tratam Contrato de Rede como um protocolo de interação [FIP 2002b], porém, como este capítulo está sendo abordado conforme descrito em Brenner [BRE 98], optamos por manter o título da próxima seção que descreve esses protocolos como protocolos de cooperação.

4.3 Protocolos de Cooperação

Os agentes podem trabalhar de forma cooperativa quando o nível de complexidade de um problema a ser resolvido é alto. Esses agentes compartilham conhecimento utilizando a propriedade de comunicação e trabalham de forma paralela. Os protocolos mais importantes, segundo Brenner [BRE 98], são:

Planejamento Global Parcial

A mais importante característica do Planejamento Global Parcial é que, a todo agente de uma aplicação baseada em agentes é dada a capacidade para que ele adquira conhecimento sobre o estado atual de outros agentes, sendo que este conhecimento pode ser utilizado pelo agente para otimizar suas próprias tarefas [BRE 98].

A condição básica para o uso do planejamento global parcial é a exigência de que diversos agentes distribuídos trabalhem na solução de um problema global. Se este é o caso, um agente como parte do planejamento global parcial pode observar as ações e relacionamentos entre um grupo de outros agentes e então formar conclusões para seu próprio trabalho. Detalhes adicionais sobre planejamento global parcial podem ser encontrados em Brenner [BRE 98].

Contrato de Rede

O protocolo contrato de rede consiste em um número de nodos que são formados pelos agentes individuais e representa um conceito que pode ser usado para estabelecer mecanismos de coordenação eficientes entre os agentes em uma aplicação baseada em agentes.

A comunicação nesse tipo de sistema é feita através de mensagens de aceitação, baseadas no protocolo de contrato de rede. Os nós do sistema são formados pela base de dados, pelo processador de contrato, pela coordenação dos nós, pelo processador de tarefa, responsável pela resolução dos subproblemas e pelo processador de comunicação, o único que está ligado diretamente à rede e responsável pelo envio e recebimento das mensagens.

O Protocolo de Interação (IP – *Interaction Protocol*) Contrato de Rede FIPA [FIP 2002b] é uma modificação secundária do IP contrato de rede original ao qual é adicionado ato comunicativo de rejeição e confirmação. No IP contrato de rede, um agente tem o papel de gerente que deseja ter alguma tarefa executada por um ou mais agentes e adicionalmente deseja otimizar uma função que caracteriza a tarefa. Esta característica poderia ser um preço, ou o tempo para conclusão de uma tarefa, distribuição justa de tarefas, etc.

O gerente solicita propostas de outros agentes emitindo um ato *call for proposals* que especifica a tarefa, e quais condições o gerente está colocando para a execução da tarefa. Agentes que recebem o *call for proposals* são vistos como potenciais contratantes e são capazes de gerar propostas para executar a tarefa como atos *propose*. A proposta do contratante inclui as precondições que estão sendo estabelecidas para a tarefa, podendo ser o custo, o tempo para realização da tarefa, etc. Alternativamente, o contratante pode recusar (*refuse*) a proposta. Uma vez que o prazo estabelecido tenha esgotado, o gerente avalia as propostas recebidas e seleciona um ou mais agentes para executar a tarefa. Para os agentes das propostas selecionadas será enviado um ato *accept-proposal* e para os outros agentes um ato *reject-proposal*. Assim que o gerente aceita as propostas, as mesmas são ligadas ao contratante, sendo que este se compromete a executar a tarefa. Uma vez que o contratante tenha completado a tarefa, é enviada uma mensagem de conclusão para o gerente.

Este IP requer que o gerente identifique quando todas as respostas foram recebidas. No caso em que o contratante não responder com um ato *propose* ou um

refuse, o gerente pode ser deixado esperando indefinidamente. Por precaução contra esta situação, o *call for proposals* inclui um tempo máximo em que a resposta poderia ser recebida pelo gerente. Propostas recebidas depois do tempo máximo definido são automaticamente rejeitadas .

Porém, a FIPA não trata somente desse IP. Na verdade ela mantém uma biblioteca de IP (IPL – *Interaction Protocol Library*), dando ao desenvolvedor liberdade para escolher o IP mais apropriado para sua aplicação, pois o padrão FIPA não força o uso de um IP em particular.

Assim, caso seja de interesse, um desenvolvedor pode criar o seu próprio IP, submetê-lo à FIPA para aprovação e inclusão na IPL FIPA. Os critérios mínimos que devem ser satisfeitos para que um IP esteja em conformidade com o padrão FIPA são: (i) o IP deve ter uma representação clara e exata através de diagramas de protocolo AUML (*Agent Unified Modeling Language*), que é uma extensão da linguagem UML (*Unified Modeling Language*) para modelar agentes e sistemas baseados em agentes, proposta por Odell [ODE 2000]; (ii) o IP deve ter uma documentação clara e consistente, e (iii) a justificativa da utilidade/benefício de um novo IP deve ser clara.

4.4 Considerações Finais

Este capítulo apresentou os métodos de comunicação quadro-negro, passagem de mensagem e sistema federativo e também os protocolos KQML e FIPA ACL. No padrão FIPA, toda comunicação entre os agentes é executada através de passagem de mensagem, utilizando FIPA ACL como protocolo para representar as mensagens. Como o DiSEN está sendo proposto em conformidade com o padrão FIPA, será adotada também a passagem de mensagem como método de comunicação e FIPA ACL como protocolo.

Foi tratado também sobre protocolos de cooperação, com detalhes sobre o Protocolo Contrato de Rede. Cabe ressaltar que, para o DiSEN, não especificaremos, nesse momento, nenhum protocolo em particular. Como os detalhes de implementação estão fora do escopo desse trabalho e como a tecnologia está em constante evolução, deixaremos como incumbência para os desenvolvedores do ambiente decidirem o protocolo mais apropriado às características do DiSEN.

5 Simulação e Validação da Arquitetura Proposta

Neste capítulo será descrita a simulação do DiSEN. Para tanto, foi estabelecida a seguinte estratégia: elaborar diagramas de *use cases*, descrever os *use cases*, elaborar o diagrama de classe do ambiente e elaborar o diagrama de seqüência e descrevê-los. Os diagramas de *use case* possibilitarão descrever os principais *use case* do ambiente; o diagrama de classe, as possíveis classes que serão necessárias para a realização dos *use cases* identificados; os diagramas de seqüência, para visualizar e documentar os aspectos dinâmicos do sistema. Conforme descrito na seção 3.3.6 o processo a ser utilizado neste ambiente será a MDSODI, portanto, optou-se por utilizar a notação da MDSODI para simular e assim validar o DiSEN.

Diagrama de *Use Case*

O diagrama de *use case* descreve graficamente como ocorre uma interação típica entre um ator e um sistema. Cada um dos *use case* descritos abrange uma função a estar disponível ao usuário e define um objetivo do projeto [LAR 2000]. Os diagramas de *use cases* são importantes principalmente para a organização e modelagem dos comportamentos de um sistema [BOO 2000].

Considerando a notação da MDSODI, poderemos encontrar atores exclusivos, atores paralelos, atores distribuídos e atores paralelos e distribuídos. Em relação aos *use cases*, podemos encontrar *use cases* seqüenciais, *use cases* paralelos, *use cases* distribuídos e *use cases* paralelos e distribuídos .

Para facilitar o entendimento, os diagramas de *use case* foram divididos em: DiSEN, Gerenciar Processo de Desenvolvimento, Gerenciar Execução do Projeto e Definir Região de Agente. Tais diagramas foram especificados considerando os aspectos de sistemas distribuídos de paralelismo/concorrência e distribuição. A figura 5.1 ilustra o diagrama de *use case* do DiSEN e a descrição dos *use cases* encontra-se na tabela 5.1.

A figura 5.1 mostra o diagrama de *use case* do DiSEN, onde todos os *use cases* são paralelos e distribuídos, ou seja, podem estar em locais distintos do ADS e devem ser executados em paralelo. Em conseqüência disso os atores envolvidos com os *use cases* também são paralelos e distribuídos. O diagrama mostra também o relacionamento de cada *use case* em relação aos demais, representado por uma linha mais densa quando o relacionamento é paralelo e por uma linha mais fina quando é seqüencial.

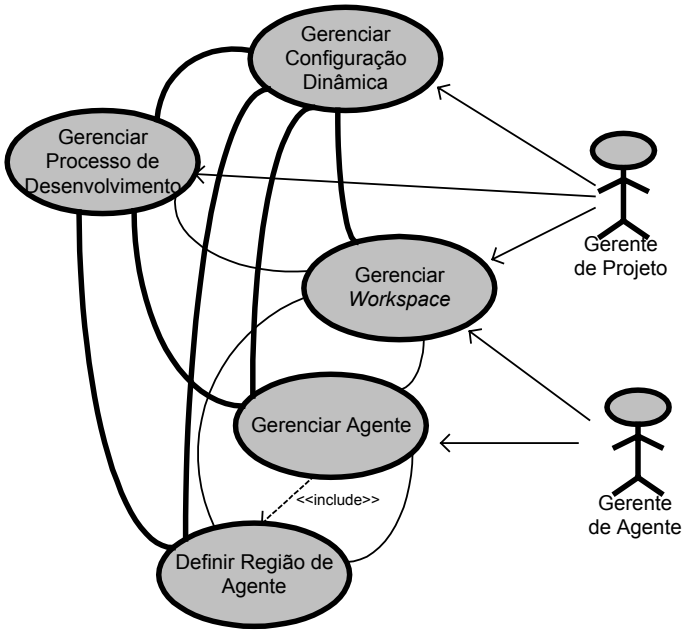


FIGURA 5.1 – Diagrama de *use case* do DiSEN

TABELA 5.1 – Descrição dos *use cases* do Diagrama do DiSEN

Use Case	**Quem inicia a ação**	**Descrição**
Gerenciar *workspace*	Gerente de Projeto e Gerente de Agente	Definir os *workspaces* de cada local, os atores que trabalharão em cada *workspace* bem como os agentes que farão parte do mesmo.
Gerenciar Processo de Desenvolvimento	Gerente de Projeto	Gerenciar todas as etapas de um processo de desenvolvimento de software. Esse é um *use case* base e a partir dele é originado o diagrama da figura 5.2.
Gerenciar Configuração Dinâmica	Gerente de Projeto	Gerenciar a (re)configuração do ambiente e inclusão/modificação/exclusão de serviços em tempo de execução.
Gerenciar Agentes	Gerente de Agente	Gerenciar todas as operações relativas a agentes.
Definir Região de Agente	Gerente de Agente	Inserir uma nova região de agente. Esse é um *use case* base, pois dá origem a outro diagrama (ver figura 5.4).

A fim de facilitar o entendimento do relacionamento entre os *use cases* de um diagrama, sugere-se construir uma matriz com cada *use case* em uma linha e uma coluna da matriz, a fim de relacionar cada *use case* em relação aos demais. Segundo a notação MDSODI, o relacionamento entre dois *use cases* pode ser paralelo ou seqüencial. Quando o relacionamento é paralelo a linha que liga os mesmos é mais densa e, para representar o seqüencial, a linha é mais fina. A tabela 5.2 mostra a definição dos relacionamentos entre os *use cases* do diagrama apresentado na figura 5.1.

TABELA 5.2 – Definição dos relacionamentos entre *use cases* do Diagrama do DiSEN

Use Case	Gerenciar Configuração Dinâmica	Gerenciar Processo de Desenvolvimento	Gerenciar *Workspace*	Gerenciar Agente	Definir Região de Agente
Gerenciar Configuração Dinâmica		Paralelo	Paralelo	Paralelo	Paralelo
Gerenciar Processo de Desenvolvimento	Paralelo		Seqüencial	Paralelo	Paralelo
Gerenciar *Workspace*	Paralelo	Seqüencial		Seqüencial	Seqüencial
Gerenciar Agente	Paralelo	Paralelo	Seqüencial		Seqüencial
Definir Região de Agente	Paralelo	Paralelo	Seqüencial	Seqüencial	

A figura 5.2 ilustra o diagrama de *use case* Gerenciar Processo de Desenvolvimento, sendo que, as tabelas 5.3 e 5.4, apresentam, respectivamente, a descrição dos *use cases* e a definição dos relacionamentos entre eles, ou seja, se é um relacionamento paralelo ou seqüencial, segundo a notação MDSODI.

O diagrama de *use case* apresentado na figura 5.2 mostra os *use cases* necessários para que o gerente de projeto possa gerenciar o processo de desenvolvimento. Todos os *use cases* do diagrama, bem como o ator, são paralelos e distribuídos, devendo ser executados de forma paralela e podendo estar em locais diferentes no ADS.

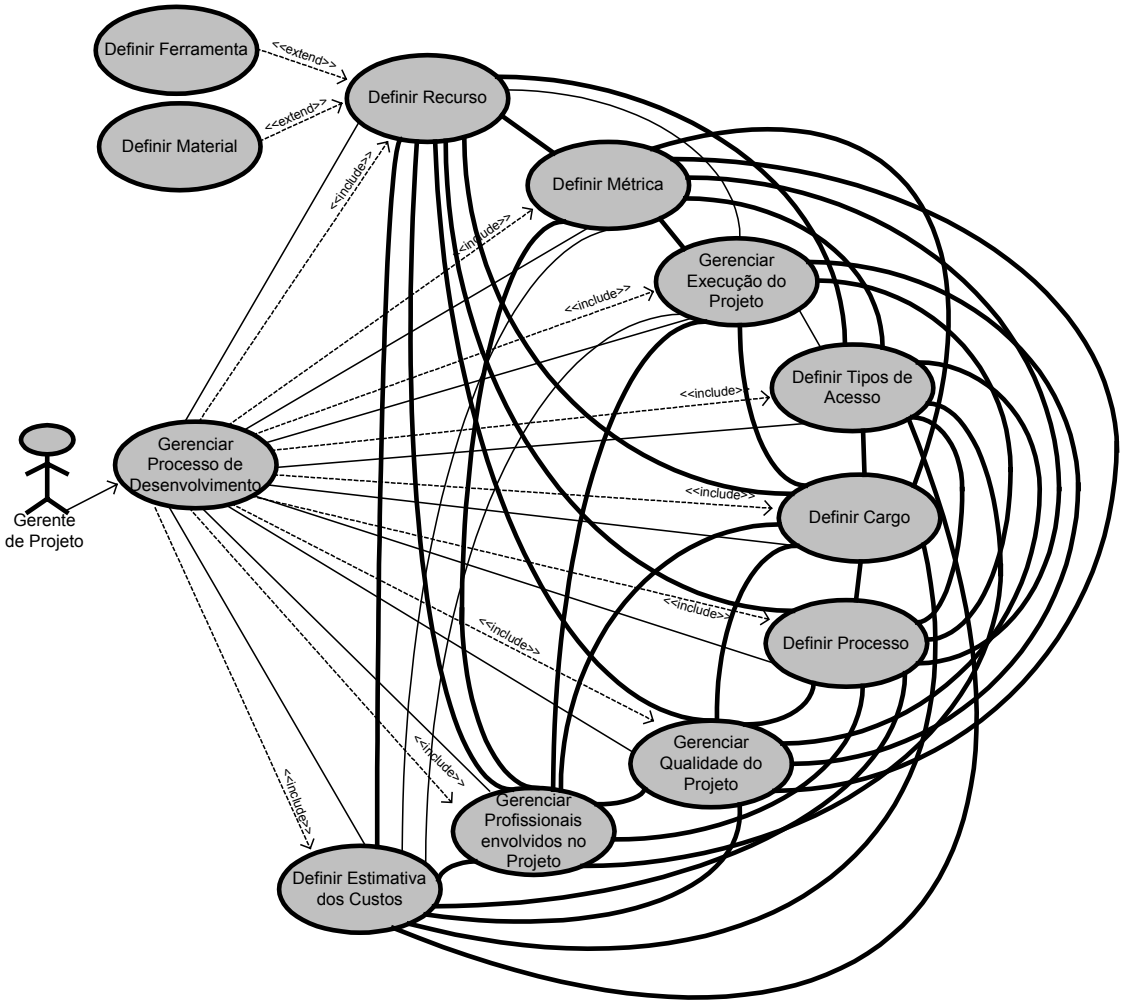


FIGURA 5.2 – Diagrama de *use case* Gerenciar Processo de Desenvolvimento

TABELA 5.3 – Descrição dos *use cases* do Diagrama Gerenciar Processo de Desenvolvimento

Use case	**Quem inicia a ação**	**Descrição**
Gerenciar Execução do Projeto	Gerente de Projeto	Gerenciar todas as tarefas necessárias para a realização de um projeto. Esse é um *use case* base, já que a partir dele foi originado outro diagrama de *use case*.
Definir Métrica	Gerente de Projeto	Inserir uma nova métrica
Definir Estimativa dos Custos	Gerente de Projeto	Inserir uma nova estimativa de custo.
Gerenciar Profissionais Envolvidos no Projeto	Gerente de Projeto	Gerenciar os passos necessários para tornar mais efetivo o uso dos recursos humanos envolvidos no projeto.
Gerenciar Qualidade do Projeto	Gerente de Projeto	Gerenciar os passos necessários para garantir que o projeto irá satisfazer as necessidades para as quais ele foi elaborado.
Definir Processo	Gerente de Projeto	Inserir um novo processo. Inicialmente a MDSODI será a única instância de processo, porém o ambiente já estará preparado para vários processos de desenvolvimento.
Definir Cargo	Gerente de Projeto	Inserir um novo cargo.
Definir Tipos de Acesso	Gerente de Projeto	Inserir um novo tipo de acesso.
Definir Recurso	Gerente de Projeto	Inserir um novo recurso que pode ser uma ferramenta ou um material.
Definir Ferramenta	Gerente de Projeto	Inserir uma nova ferramenta.
Definir Material	Gerente de Projeto	Inserir um novo material.

A figura 5.3 ilustra o diagrama *use case* Gerenciar Execução do Projeto, e a descrição dos *use cases* encontra-se na tabela 5.5. Também pode ser encontrada na tabela 5.6 a definição do relacionamento de um *use case* em relação aos demais, segundo a notação MDSODI.

No diagrama de *use case* da figura 5.3 são mostrados os *use cases* necessários para que o gerente de projeto possa gerenciar a execução de um projeto como, por exemplo, definir atividade e definir agenda do ator. Como nos diagramas anteriores, todos os *use cases*, assim como o ator, são paralelos e distribuídos, devendo ser executados de forma paralela e podendo estar em locais diferentes no ADS.

TABELA 5.4 – Definição dos relacionamentos entre *use cases* do Diagrama Gerenciar Processo de Desenvolvimento

Use Case	Gerenciar Processo de Desenvolvi-mento	Gerenciar profissionais envolvidos no projeto	Gerenciar qualidade do projeto	Definir processo	Definir cargo	Definir tipos de acesso	Definir recurso	Gerenciar execução do projeto	Definir métrica	Definir estimativa dos custos
Gerenciar Processo de Desenvolvi-mento		Seqüencial	Seqüencial	Seqüencial	Seqüencial	Seqüencial	Seqüencial	Seqüencial	Seqüencial	Seqüencial
Gerenciar profissionais envolvidos no projeto	Seqüencial		Paralelo	Paralelo	Paralelo	Paralelo	Paralelo	Paralelo	Paralelo	Paralelo
Gerenciar qualidade do projeto	Seqüencial	Paralelo		Paralelo	Paralelo	Paralelo	Paralelo	Paralelo	Paralelo	Paralelo
Definir processo	Seqüencial	Paralelo	Paralelo		Paralelo	Paralelo	Paralelo	Seqüencial	Paralelo	Paralelo
Definir cargo	Seqüencial	Paralelo	Paralelo	Paralelo		Paralelo	Paralelo	Paralelo	Paralelo	Paralelo
Definir tipos de acesso	Seqüencial	Paralelo	Paralelo	Paralelo	Paralelo		Paralelo	Seqüencial	Paralelo	Paralelo
Definir recurso	Seqüencial	Paralelo	Paralelo	Paralelo	Paralelo	Paralelo		Seqüencial	Paralelo	Paralelo
Gerenciar execução do projeto	Seqüencial	Paralelo	Paralelo	Seqüencial	Paralelo	Seqüencial	Seqüencial		Seqüencial	Seqüencial
Definir métrica	Seqüencial	Paralelo	Paralelo	Paralelo	Paralelo	Paralelo	Paralelo	Seqüencial		Paralelo
Definir estimativa dos custos	Seqüencial	Paralelo	Paralelo	Paralelo	Paralelo	Paralelo	Paralelo	Seqüencial	Paralelo	

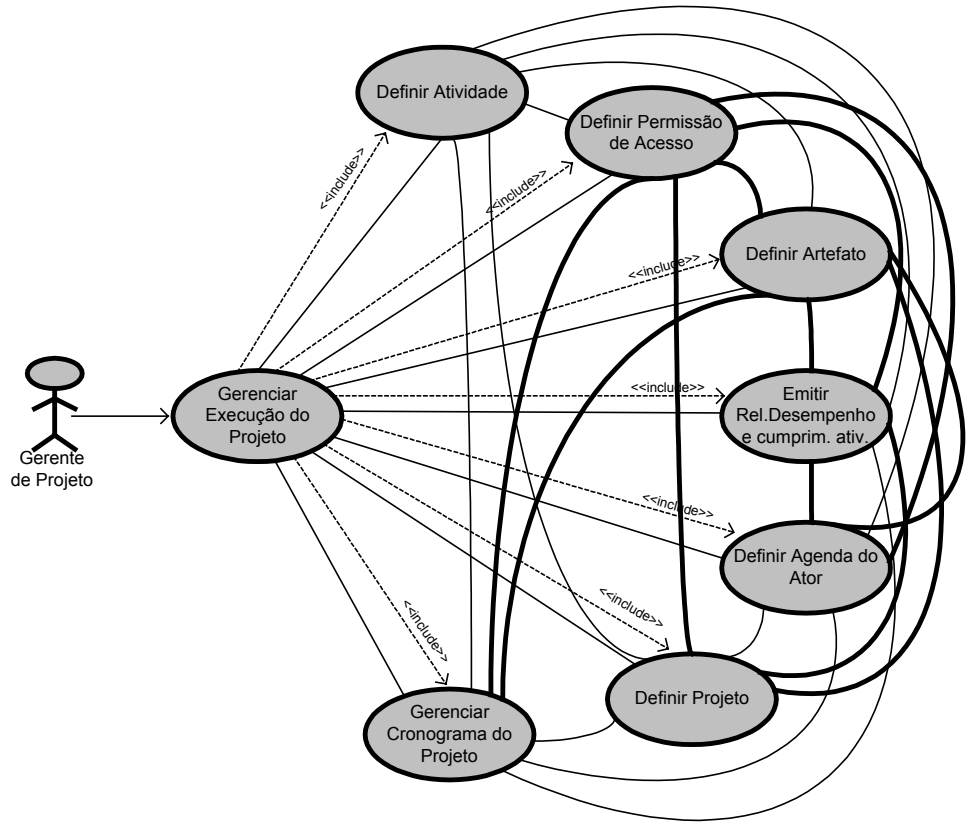


FIGURA 5.3 – Diagrama de *use case* Gerenciar Execução do Projeto

TABELA 5.5 – Descrição dos *use cases* do Diagrama Gerenciar Execução do Projeto

Use case	**Quem inicia a ação**	**Descrição**
Definir Projeto	Gerente de Projeto	Inserir um novo projeto.
Definir Atividade	Gerente de Projeto	Inserir uma nova atividade.
Definir Permissão de Acesso	Gerente de Projeto	Inserir uma nova permissão de acesso.
Definir Artefato	Gerente de Projeto	Inserir um novo artefato.
Definir Agenda do Ator	Gerente de Projeto	Agendar uma atividade para um determinado ator.
Elaborar Relatórios de Desempenho e Cumprimento das Atividades	Gerente de Projeto	Elaborar relatórios que descrevam a situação atual do projeto e o que a equipe já realizou.
Gerenciar Cronograma do Projeto	Gerente de Projeto	Definir data de início prevista e data de término prevista para cada atividade relacionada ao projeto. Envolve também o controle das mudanças no cronograma do projeto.

TABELA 5.6 – Definição dos relacionamentos entre *use cases* do Diagrama Gerenciar Execução do Projeto

Use Case	Gerenciar Execução do Projeto	Definir Atividade	Definir Permissão de Acesso	Definir Artefato	Emitir relatório desempenho e cumprimento de atividade	Definir agenda do ator	Definir projeto	Gerenciar cronograma do projeto
Gerenciar Execução do Projeto		Seqüencial	Seqüencial	Seqüencial	Seqüencial	Seqüencial	Seqüencial	Seqüencial
Definir Atividade	Seqüencial		Seqüencial	Seqüencial	Seqüencial	Seqüencial	Seqüencial	Seqüencial
Definir Permissão de Acesso	Seqüencial	Seqüencial		Paralelo	Paralelo	Paralelo	Paralelo	Paralelo
Definir Artefato	Seqüencial	Seqüencial	Paralelo		Paralelo	Paralelo	Paralelo	Paralelo
Emitir relatório desempenho e cumprimento de atividade	Seqüencial	Seqüencial	Paralelo	Paralelo		Paralelo	Paralelo	Seqüencial
Definir agenda do ator	Seqüencial	Seqüencial	Paralelo	Paralelo	Paralelo		Seqüencial	Seqüencial
Definir projeto	Seqüencial	Seqüencial	Paralelo	Paralelo	Paralelo	Seqüencial		Seqüencial
Gerenciar cronograma do projeto	Seqüencial	Seqüencial	Paralelo	Paralelo	Seqüencial	Seqüencial	Seqüencial	

A figura 5.4 ilustra o diagrama de *use case* Definir Região de Agente, e a descrição dos *use cases* encontra-se na tabela 5.7

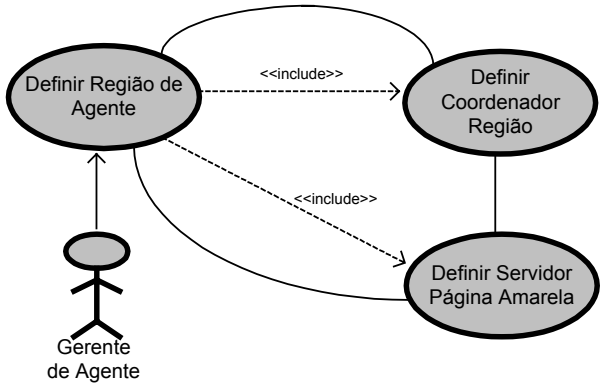


FIGURA 5.4 – Diagrama de *use case* Definir Região de Agente

TABELA 5.7 – Descrição dos *use cases* do Diagrama Definir Região de Agente

Use Case	**Quem inicia a ação**	**Descrição**
Definir Coordenador de Região	Gerente de Agente	Definir o agente que será o coordenador de cada região de agente.
Definir Servidor Página Amarela	Gerente de Agente	Definir o agente que será o servidor de página amarela de cada região de agente.

Diagrama de Classes

A partir dos casos de uso apresentados nas figuras 5.1, 5.2, 5.3 e 5.4, foi elaborado o diagrama de classes do ambiente conforme mostra a figura 5.5. Devido ao diagrama envolver muitas classes, optou-se por definir primeiramente o diagrama com todas as classes, porém sem a especificação dos seus atributos e métodos. A figura 5.5 mostra o diagrama com todas as classes identificadas até o momento para o DiSEN.

A figura 5.6 mostra os atributos e as classes relacionadas ao gerenciador de objetos, identificadas até o momento. Já a figura 5.7 mostra os atributos e as classes para o gerenciador de agentes.

Para a elaboração do diagrama de classes, segundo a notação MDSODI, é necessário a identificação dos métodos a fim de definir os relacionamentos entre as classes, pois esses relacionamentos, além de representar a ligação entre as mesmas, representa também o tipo de relacionamento que as classes possuem quanto à forma de execução de seus métodos (seqüencialmente, parcialmente paralelo ou totalmente paralelo). Além disso, a representação diferenciada das próprias classes ilustra a forma de execução dos métodos internos (seqüencialmente, parcialmente paralelo ou totalmente paralelo). Assim, somente o diagrama da figura 5.8 foi elaborado segundo a notação MDSODI.

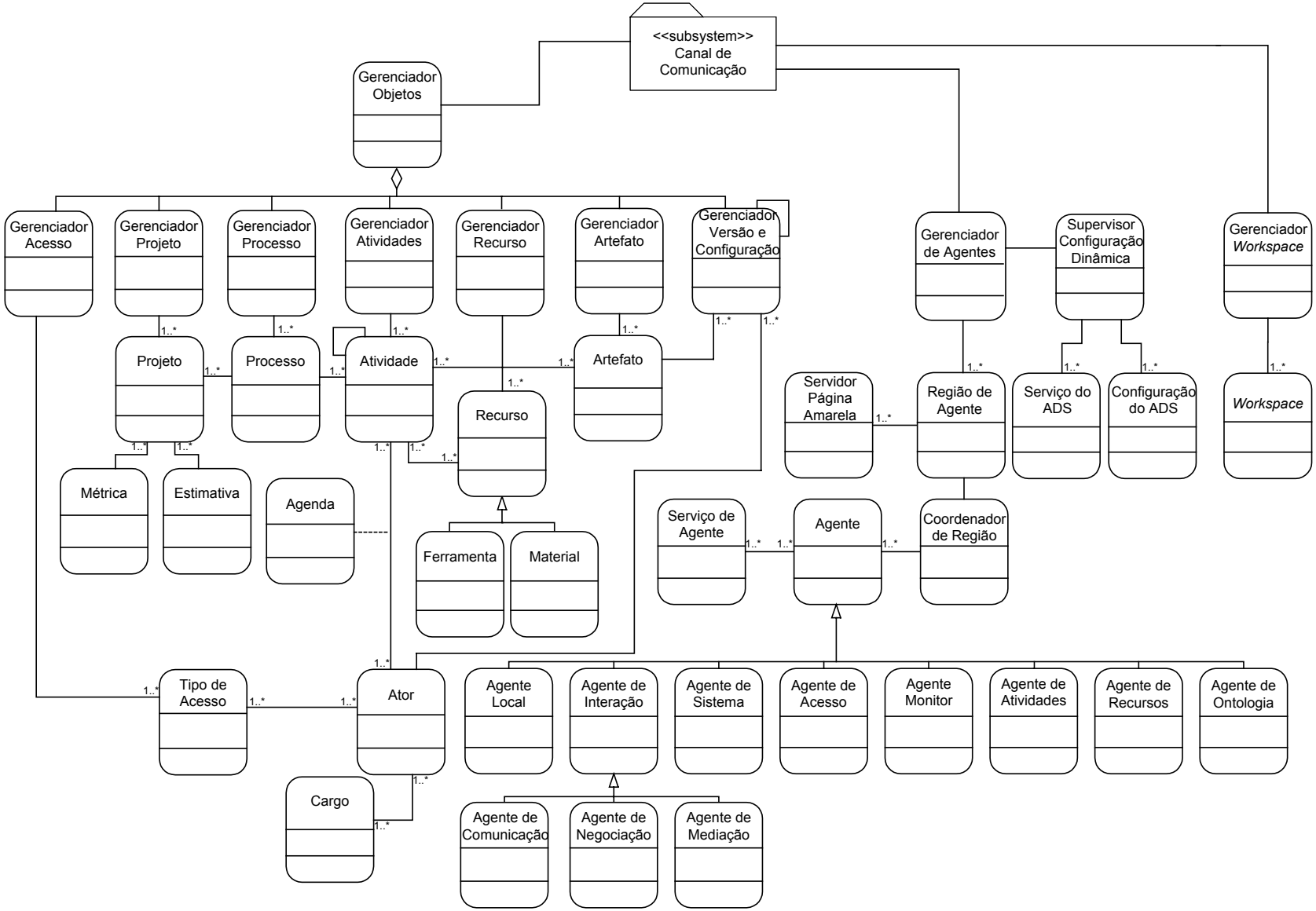


FIGURA 5.5 – Diagrama de classes do ambiente

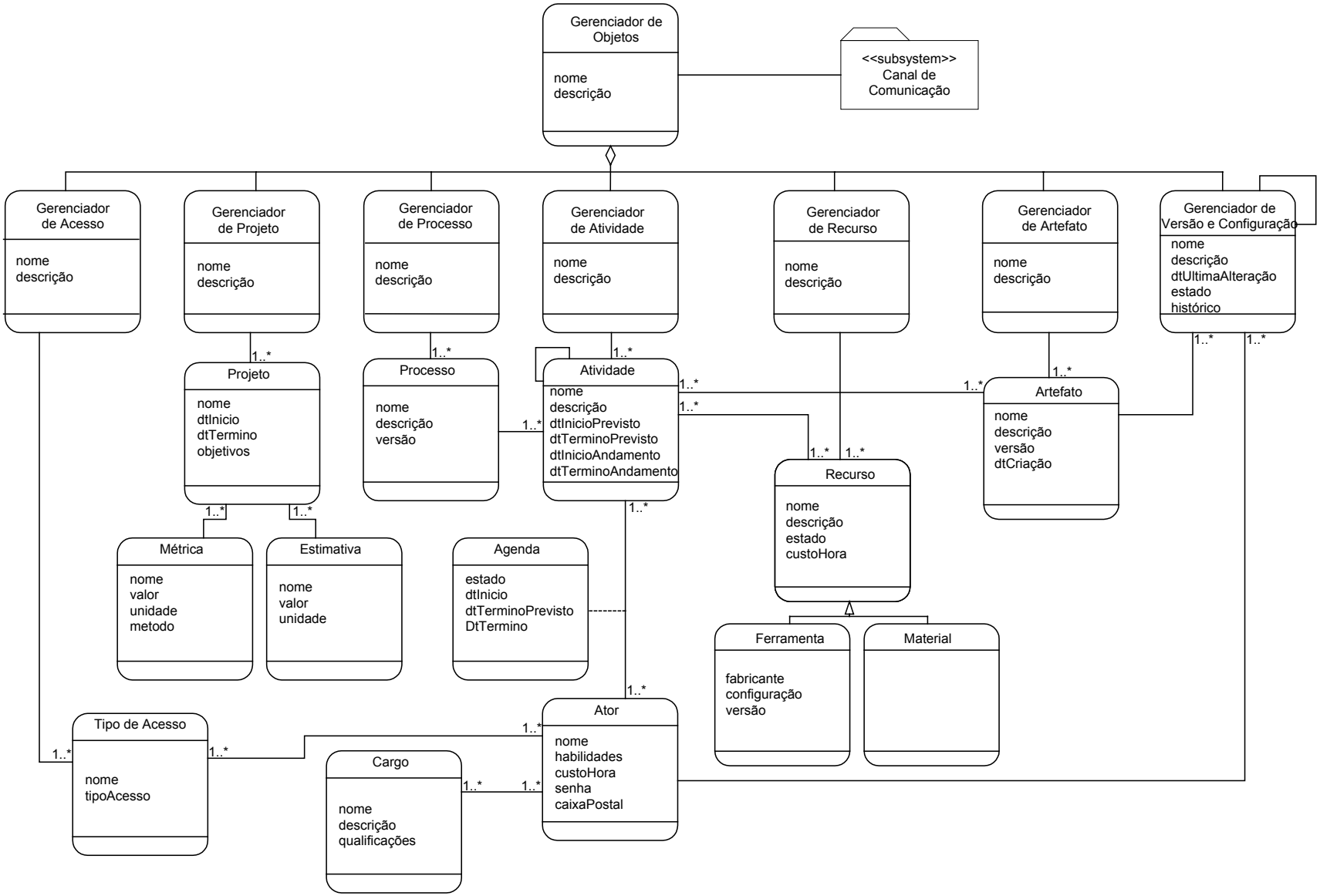


FIGURA 5.6 – Diagrama de classes e atributos definidos para o gerenciador de objetos

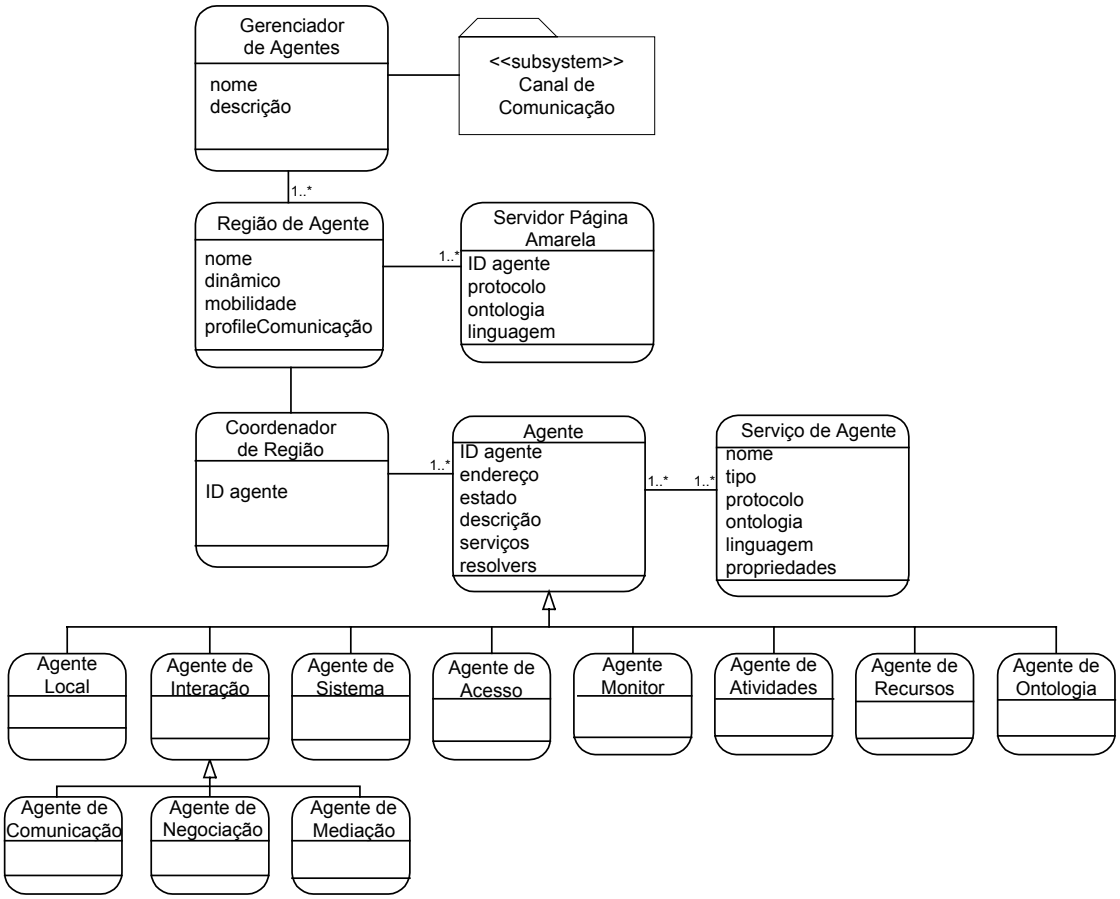


FIGURA 5.7 – Diagrama de classes e atributos definidos para o gerenciador de agentes

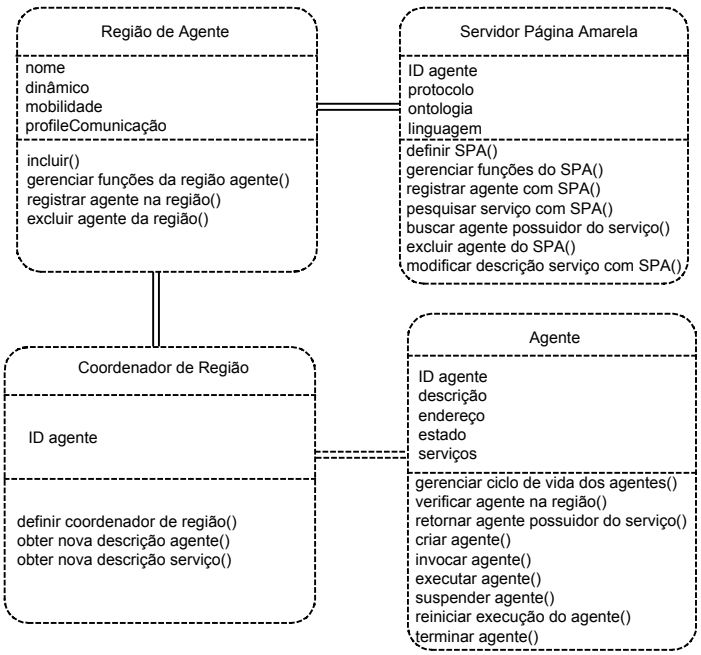


FIGURA 5.8 – Diagrama de classes parcial referente ao gerenciador de agentes (conforme a notação MDSODI)

Diagrama de Seqüência

Um diagrama de seqüência dá ênfase à ordenação temporal das mensagens. Para cada *use case* identificado deve ser elaborado um diagrama de seqüência [BOO 2000]. Nesta seção iremos mostrar os diagramas de seqüência mais relevantes para o ambiente.

Diagrama de Seqüência Definir Região de Agente

O diagrama de seqüência definir região de agente, conforme mostrado na figura 5.9, possui o ator gerente de agente que inicia a ação com o evento *solicita incluir região* na classe região de agente. A inclusão de uma nova região envolve definir qual será o agente coordenador da região e qual será o agente supervisor de página amarela da região. Isso é feito através dos métodos *definir coordenador de região* e *definir servidor de página amarela*, respectivamente.

Uma vez que o CR está definido, este será responsável por gerenciar as funções da região de agente, gerenciar as funções do servidor de página amarela e gerenciar o ciclo de vida dos agentes na região. Para evitar a complexidade do diagrama de seqüência, optou-se por fazer três diagramas separados, um para cada responsabilidade citada acima, porém vinculados ao diagrama definir região de agente.

Assim, o diagrama de seqüência gerenciar funções da região de agente, mostrado na figura 5.10, representa o registro de um agente numa determinada região. Esse registro é necessário para que o agente possa realizar suas funcionalidades. Além disso, cada agente tem um ciclo de vida que é gerenciado pela Região de Agente. Para a efetivação desse registro, o agente deve solicitar o registro ao CR, que é responsável por gerenciar as operações da região.

Uma vez que o agente está registrado numa região, ele pode modificar sua descrição, desde que o CR autorize. A vida de um agente na RA termina com a exclusão do seu registro no CR.

Além dessas operações, um agente pode pesquisar a descrição de um outro agente na região. Podem-se observar os passos envolvidos nessa pesquisa que começa com o agente solicitando ao CR a descrição de um outro agente. O CR verifica se esse outro agente está registrado na região. Em caso positivo, retorna uma mensagem contendo a descrição do agente pesquisado.

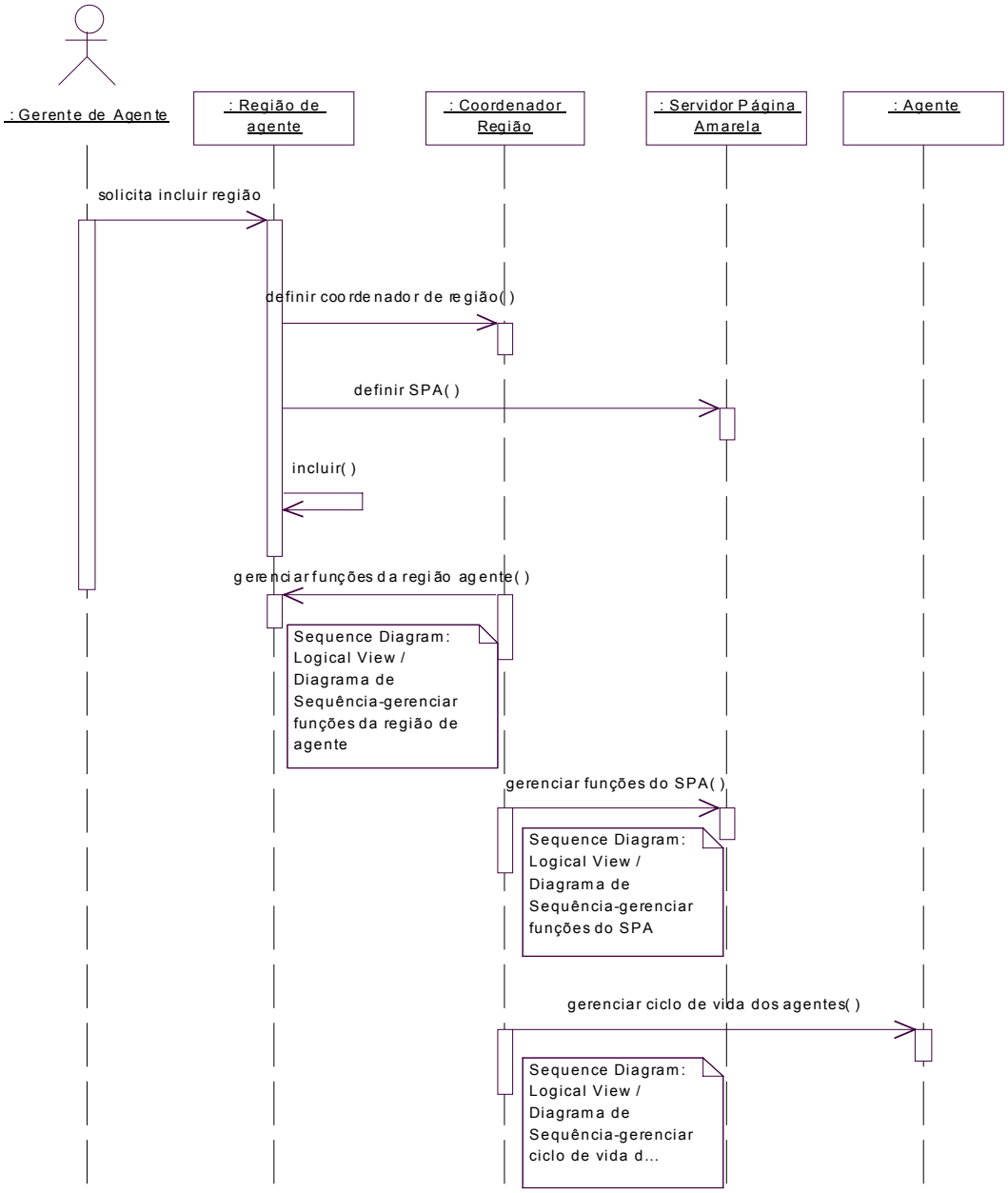


FIGURA 5.9 – Diagrama de seqüência Definir Região de Agente

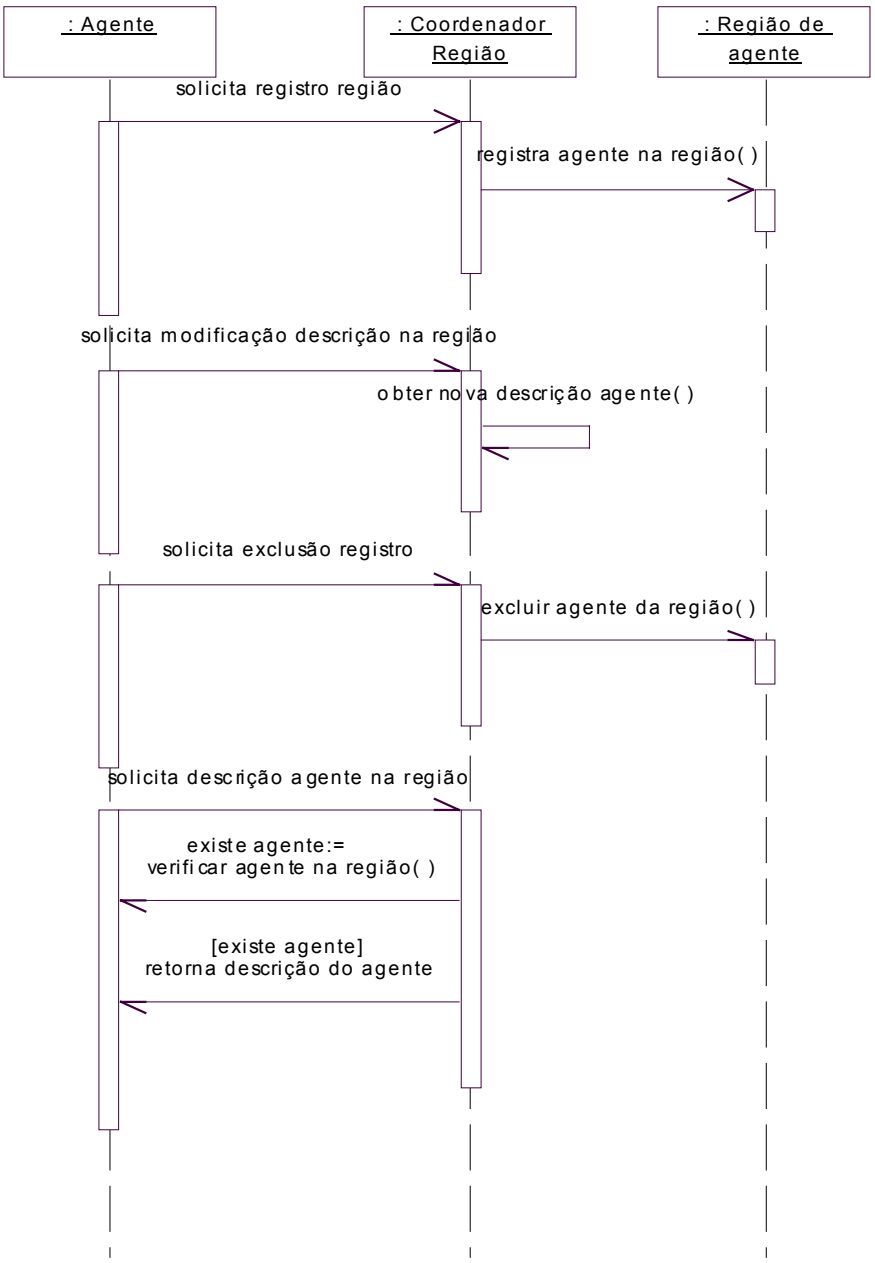


FIGURA 5.10 – Diagrama de seqüência Gerenciar Funções da Região de Agente

Todo agente que deseja tornar público seus serviços para outros agentes deve requerer o seu registro com o SPA da região. Assim, o SPA é responsável por armazenar e tornar disponíveis informações sobre serviços anunciados pelos agentes numa mesma região, possibilitando a colaboração e delegação de tarefas.

As funções suportadas pelo SPA estão descritas no diagrama de seqüência gerenciar funções do SPA, mostrado na figura 5.11. Para que o agente possa registrar seus serviços com o SPA, ele deve, primeiramente, solicitar o registro ao CR, que por sua vez, efetua o registro com o SPA.

Um agente pode pesquisar serviços fornecidos por outros agentes através do SPA. Para isso, o agente deve enviar uma mensagem de solicitação para o CR, este por

sua vez efetua a pesquisa do serviço com o SPA. Caso exista um agente que ofereça o serviço que está sendo solicitado, o SPA retorna o identificador desse agente.

Além das operações de registro de serviço e pesquisa de serviço, um agente que tenha se registrado com o SPA pode solicitar a sua exclusão ou pode solicitar a modificação da descrição do serviço que havia sido registrada. Em ambas as situações, a solicitação sempre será feita ao CR.

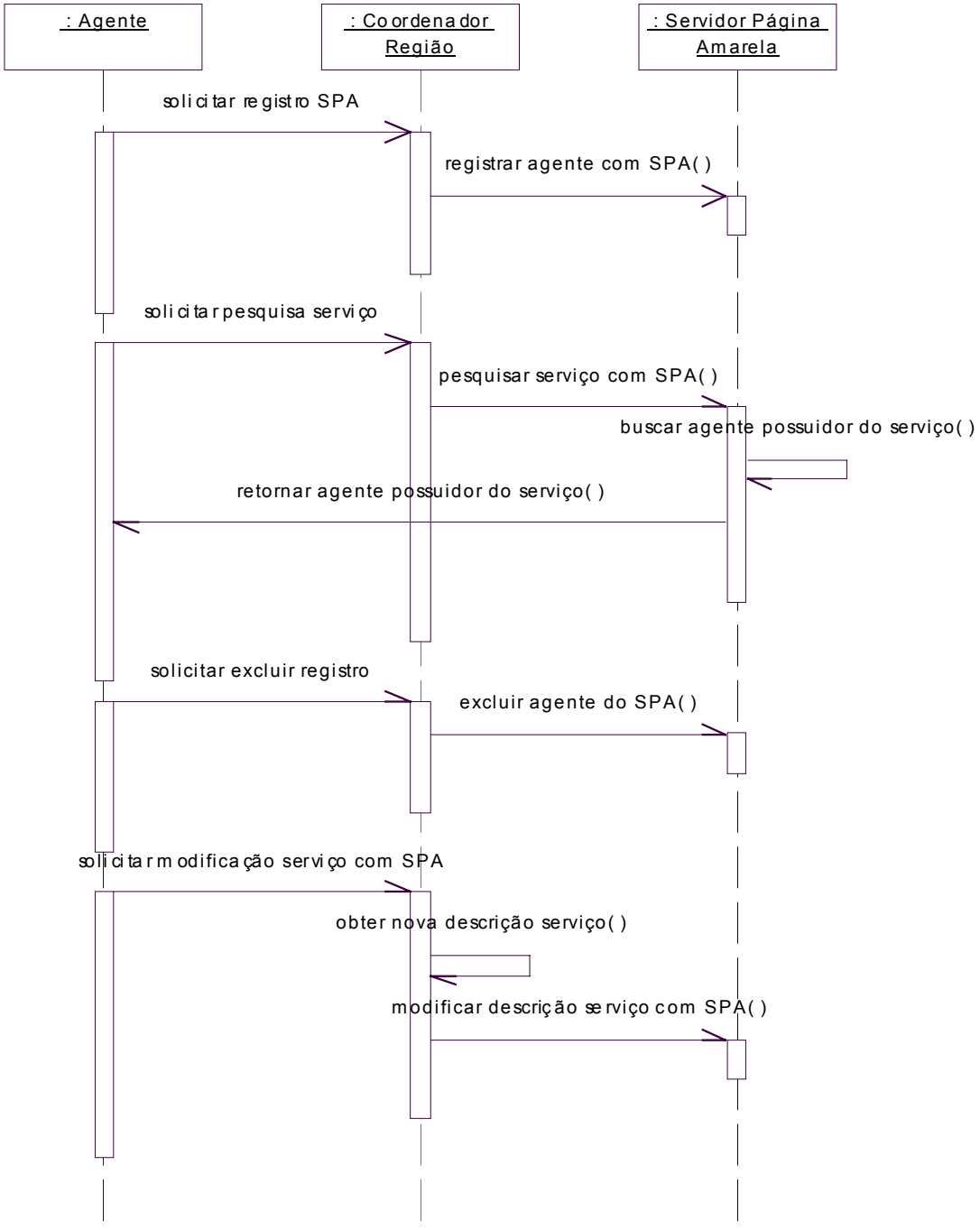


FIGURA 5.11 – Diagrama de seqüência Gerenciar Funções do Servidor de Página Amarela

A última responsabilidade do CR é gerenciar o ciclo de vida dos agentes que residem na região. Um agente pode ser criado, invocado, executado, suspenso, reiniciado ou terminado pelo CR, conforme mostra o diagrama de seqüência gerenciar ciclo de vida dos agentes da figura 5.12.

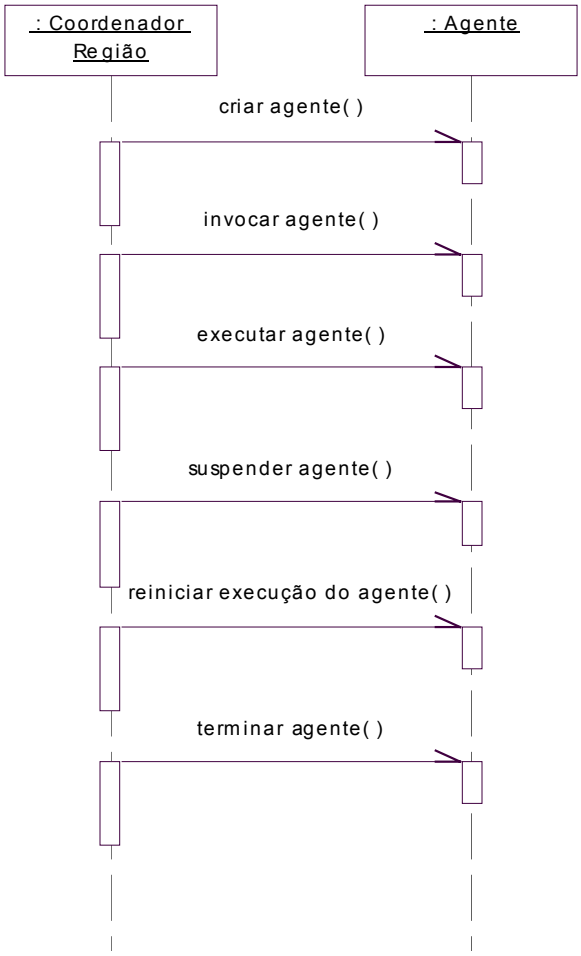


FIGURA 5.12 – Diagrama de seqüência Gerenciar Ciclo de Vida dos Agentes

A fim de ilustrar a troca de mensagens entre os elementos constituintes do DiSEN, serão definidos alguns cenários:

Cenário 1: Acesso ao DiSEN

Para que um ator possa iniciar o uso do DiSEN é preciso primeiro que ele tenha permissão para acessar o ambiente. Assim, quando um ator deseja acessar o ambiente, através do *workspace*, o agente de comunicação envia uma mensagem para o CR através do canal de comunicação a fim de verificar se existe um agente de acesso naquela região. Se existir um agente de acesso naquela região, ele vai verificar se o ator possui permissão de acesso e retorna uma mensagem para o CR que, por sua vez, retorna para o agente de comunicação. Caso não exista um agente de acesso na região, o CR deve comunicar-se com as outras regiões até encontrar o agente de acesso. Caso o ator não tenha permissão de acesso, o mesmo não poderá utilizar nenhuma funcionalidade do ambiente.

O diagrama de seqüência da figura 5.13 está representando a troca de mensagens entre os elementos do DiSEN necessários para que o ator possa ter acesso ao ambiente. Conforme descrito em Fowler [FOW 2000], em um diagrama de seqüência pode existir a condição e a iteração. A condição indica quando uma mensagem é enviada, sendo que o envio é feito somente se a condição for verdadeira. Porém, no diagrama da figura 5.13, quando a condição [existeAgente] for falsa deverá ser enviada uma mensagem para a RA e se a condição for verdadeira deverá ser enviada uma mensagem para o agente de acesso. Já a iteração foi utilizada na mensagem *procura agente acesso* que está sendo enviada do CR para a RA, indicando que a busca deve ser feita até que seja encontrado um agente de acesso em alguma região.

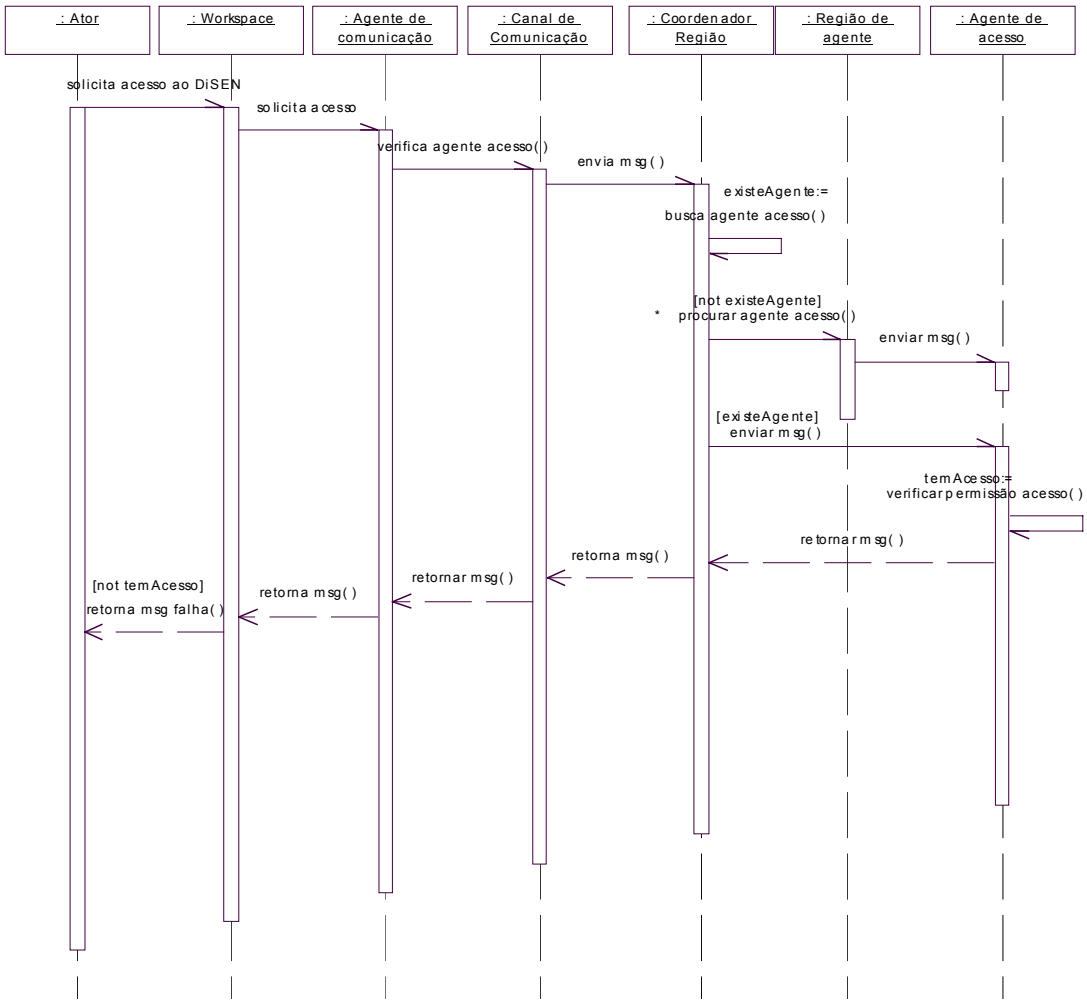


FIGURA 5.13 – Diagrama de seqüência do cenário Acesso ao DiSEN

Cenário 2: Execução de atividade

Para que uma atividade seja realizada ela pode ser iniciada pelo ator ou automaticamente. No primeiro caso, o ator estará trabalhando no *workspace* e para que ele possa realizar uma atividade é necessário um recurso, como uma ferramenta, por exemplo. Automaticamente o agente de comunicação é acionado. Este envia uma mensagem para o CR (da região onde ele está) através do canal de comunicação. O CR verifica se há um agente de recursos naquela região. Em caso positivo, o agente de recursos é consultado sobre a disponibilidade do recurso solicitado. Caso o estado do recurso esteja LIVRE, ele é alocado para a atividade que está sendo realizada pelo ator

que o solicitou, e o seu estado deve ser atualizado para EM USO. Caso o recurso esteja EM USO por outra atividade, o agente de negociação vai tentar resolver o conflito através de uma estratégia de negociação específica para o problema. Caso o agente de negociação não consiga alcançar entendimento, o agente de mediação será usado para ajudar no processo de negociação. Caso não exista um agente de recursos naquela região, o CR comunica-se com outra região até conseguir encontrar o agente de recursos, sendo que, se essa região estiver no mesmo local, a comunicação será via canal de comunicação, caso contrário, será via Internet.

Quando a atividade é automática, irão acontecer os mesmos passos descritos anteriormente, porém não haverá a interação do ator e sim do agente de atividades. Deve ser verificado se todas as atividades anteriores já foram terminadas.

A cada atividade realizada, quer tenha sido de forma automática ou pelo ator, o agente de atividades deve, quando a mesma foi iniciada, ter atualizado a data de início de andamento e também o estado da atividade na agenda do ator para EM ANDAMENTO; quando a atividade for encerrada, atualizar a data de término de andamento e o estado da atividade na agenda do ator para TERMINADA e solicitar ao agente de recursos a atualização do estado do recurso para LIVRE.

O cenário descrito acima está representado no diagrama de seqüência da figura 5.14, onde está sendo mostrada a troca de mensagens entre os elementos do DiSEN necessários para que o ator possa executar uma atividade. Neste diagrama também ocorrem a condição e a iteração. Quando a condição [existeAgente] for falsa, o CR deverá enviar uma mensagem para a RA e se a condição for verdadeira a mensagem será enviada para o agente de acesso. Já a iteração foi utilizada na mensagem *procura agente recurso* que está sendo enviada do CR para a RA, indicando que a busca deve ser feita até que seja encontrado um agente de recurso em alguma região.

Outras situações que podem ocorrer no DiSEN: um agente de atividades, por exemplo, da região 1 não quer mais residir nessa região. Ele pode solicitar a exclusão do seu registro do SPA através do CR. Para isso ele envia diretamente uma mensagem para o CR solicitando a exclusão; para a execução de alguma tarefa, o agente pode necessitar saber os serviços que são fornecidos por outro agente. Nesse caso, o agente envia uma mensagem para o CR, que por sua vez envia uma mensagem para o SPA a fim de descobrir se existe um agente com o serviço que está sendo solicitado. Estas situações já foram descritas nos diagramas de seqüência Gerenciar Funções da Região de Agente e Gerenciar Funções do Servidor de Página Amarela, representados nas figuras 5.10 e 5.11, respectivamente.

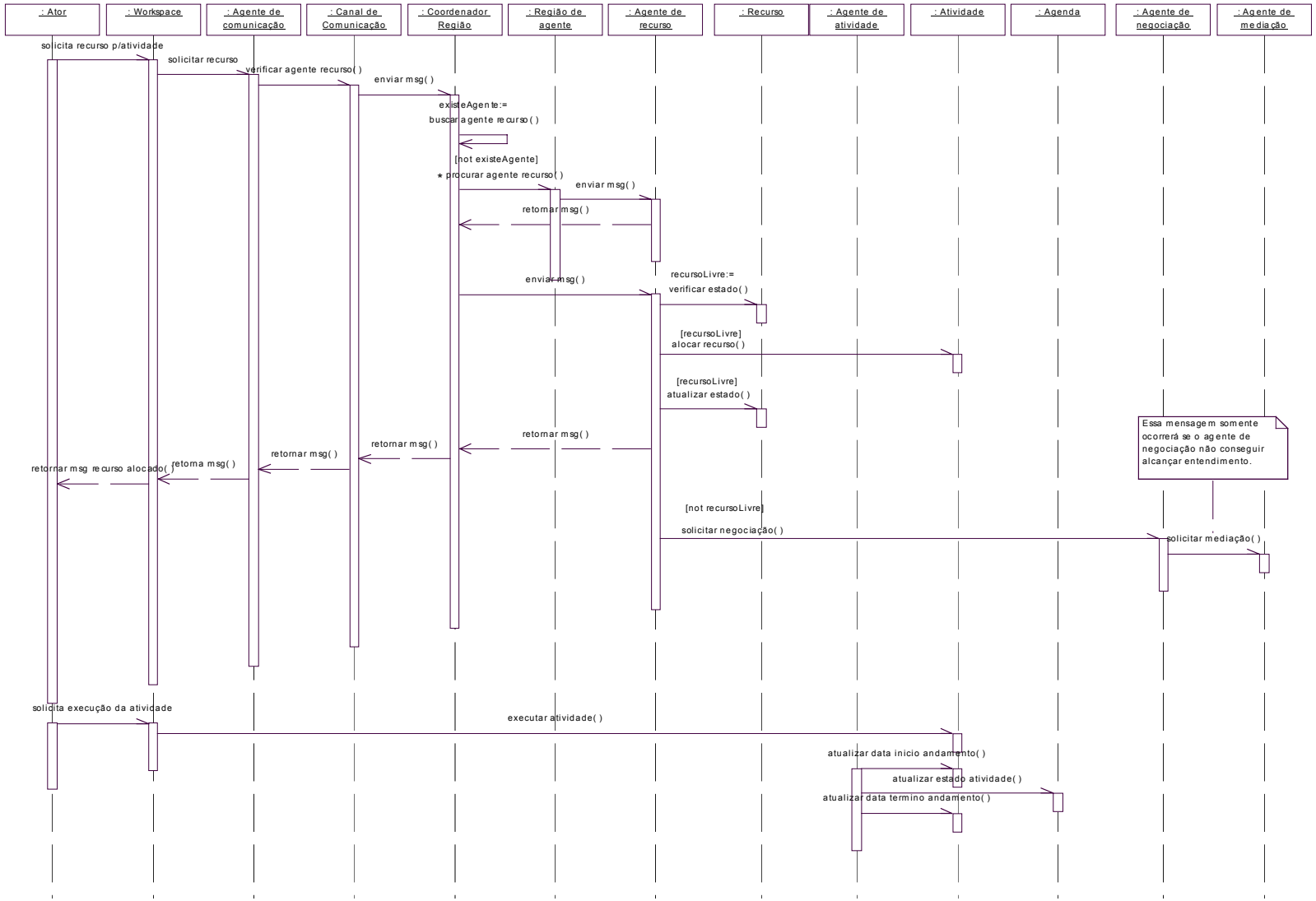


FIGURA 5.14 – Diagrama de seqüência do cenário Execução de Atividade

6 Conclusão

Esta dissertação apresenta a especificação de uma arquitetura de um ambiente de desenvolvimento de software distribuído. A arquitetura é baseada no estilo camadas, sendo composta pelas camadas dinâmica, de aplicação e da infra-estrutura. A comunicação entre as camadas de aplicação e da infra-estrutura se dá através do canal de comunicação, ou seja, este canal é responsável pela comunicação entre os elementos da arquitetura. Foi utilizada a notação MDSODI para especificar os *use case* e o diagrama de classes do ambiente e, para a simulação do ambiente, foram elaborados vários diagramas de seqüência que possibilitam visualizar a comunicação entre seus diferentes elementos, conforme apresentado no capítulo 5.

A larga utilização da Internet aliada ao uso cada vez maior das redes de computadores tem estimulado os estudos relativos ao desenvolvimento de softwares distribuídos. Esses softwares, quando comparados aos tradicionais, normalmente apresentam maior eficiência, devido às suas características diferenciadas, como por exemplo, tolerância à falhas, compartilhamento de recursos, distribuição, escalabilidade. Isto dificulta o desenvolvimento desse tipo de software com as tecnologias tradicionais de software, por causa dos limites destas em lidar com essas características; a tecnologia de agentes tem se mostrado uma boa alternativa para modelar e construir software distribuído complexo.

Portanto, é necessário também que a arquitetura dos ADS evolua para suportar novas metodologias de desenvolvimento que ofereçam o suporte necessário à construção de softwares complexos, podendo estar integrada a outras tecnologias, como a de agentes, e que permita a integração de profissionais distribuídos geograficamente.

Assim, uma vez expostas as características desejáveis para um ADS, para que se possa desenvolver software complexos, o DiSEN está sendo definido para fornecer suporte ao desenvolvimento distribuído de software, podendo estar em locais distintos com os desenvolvedores trabalhando de forma cooperativa.

A principal contribuição deste trabalho é a especificação da arquitetura do DiSEN, que possui como principais características:

- Pode estar distribuído geograficamente;
- Incorpora a MDSODI;
- Inclui um elemento que provê apoio ao trabalho cooperativo, ou seja, permite a interação de profissionais geograficamente dispersos, através da criação de regiões de agentes e *workspaces* em cada um destes locais, havendo a possibilidade de comunicação através do canal de comunicação.
- Possui atividades executadas por agentes. O desenvolvimento desses agentes deve ser em conformidade com as especificações propostas pela FIPA.

Os seguintes trabalhos encontram-se em andamento:

- A construção de uma ferramenta, denominada MDSODI – Requisitos, que dará apoio a MDSODI no que se refere à definição de requisitos [BAT 2002]. Esta ferramenta integrará o DiSEN e possuirá as seguintes características: (i) utilizar-se-á de técnicas para dar suporte à determinação dos requisitos funcionais e não-

funcionais; (ii) possibilitará desenvolvimento distribuído; (iii) irá prover apoio ao trabalho cooperativo e (iv) produzirá artefatos, tais como: modelo de negócio ou de domínio, diagrama de *use cases*, diagrama de colaboração e visão arquitetural do modelo de *use cases*.

Com a utilização dessa ferramenta espera-se benefícios, tais como: (i) transferência da informação (modelos, programas, documentos e dados) de uma ferramenta para outra; (ii) disponibilização das informações sobre os artefatos de uma fase para outra; (iii) comunicação melhorada entre as pessoas que estão trabalhando sobre grandes projetos de software.

- Definição e elaboração de uma ferramenta que dará apoio ao gerenciamento de desenvolvimento de software em sistemas distribuídos, tendo como objetivos: acompanhar o desenvolvimento do projeto através da comparação entre o planejado e o executado; verificar a situação de cada atividade assim como das equipes envolvidas; controlar as versões de cada atividade e todos os aspectos inerentes ao desenvolvimento de sistemas distribuídos. Essa ferramenta, denominada DIMANAGER – ***Distributed Software Manager*****,** oferecerá suporte à metodologia MDSODI e também será integrada ao DiSEN [PED 2002].
- Elaboração de uma proposta de solução de um repositório de metadados para ambiente de desenvolvimento de software distribuído [MOR 2002]. Esse repositório tem por objetivo fornecer suporte à integração e interoperabilidade das ferramentas de desenvolvimento de software ligadas à MDSODI, portanto, deve ser compatível com a arquitetura proposta nesta dissertação. Para que esse objetivo seja alcançado é necessário que: (i) o repositório de metadados esteja disponível na rede possibilitando que as diferentes ferramentas de desenvolvimento de software localizadas em nós distintos da rede possam utilizá-lo para o armazenamento e recuperação de seus artefatos; (ii) a solução detecte falhas na disponibilidade do suporte ao armazenamento e execute medidas de recuperação; (iii) a solução forneça o suporte para que ferramentas de desenvolvimento possam descobrir e utilizar os recursos de armazenamento de artefatos em tempo de execução.

Como trabalhos futuros sugere-se a abordagem dos seguintes aspectos:

- Como o DiSEN possui diversos *workspaces*, onde é permitida a edição cooperativa de artefatos, é necessário que esses *workspaces* sejam sincronizados e dêem suporte a *awareness*, pois é necessário levar em consideração um conjunto de questões, como por exemplo, quais mudanças os participantes estão fazendo?, onde as mudanças estão sendo feitas?, o que os participantes podem fazer?, quais objetos os participantes estão usando?.
- Implementar a arquitetura proposta. Porém, com certeza, isto será alcançado com a realização de vários trabalhos, pois a arquitetura envolve muitos elementos e será necessário um estudo minucioso e cuidadoso sobre a tecnologia disponível para a implementação de cada um desses elementos.
- Definir o servidor de ontologias, de acordo com a especificação Serviço de Ontologia FIPA [FIP 2001d].
- Estudar técnicas de inteligência artificial, tais como algoritmo genético, redes neurais, entre outras, a fim de tornar os agentes inteligentes, com capacidade de raciocínio e aprendizagem.

Anexo 1 MDSODI – Metodologia para Desenvolvimento de Software Distribuído

A definição desta metodologia faz parte do projeto de pesquisa "Uma Metodologia para o Desenvolvimento Baseado em Objetos Distribuídos Inteligentes" em desenvolvimento na Universidade Estadual de Maringá [HUZ 99].

A MDSODI [GRA 2000] é uma metodologia para desenvolvimento de software que leva em consideração algumas características identificadas em sistemas distribuídos, tais como: paralelismo/concorrência, comunicação, sincronização e distribuição, já nas fases iniciais do projeto.

Mantém as principais características do *Unified Process* [JAC 99]: dirigida a *use-case*, centrada na arquitetura e de desenvolvimento iterativo e incremental. Utiliza também algumas características propostas na MOOPP, como por exemplo, a representação gráfica para os possíveis tipos de objetos e classes, tendo como base o paralelismo.

A MOOPP [HUZ 95] é uma metodologia orientada a objetos para desenvolvimento de software para processamento paralelo que trata dos aspectos estáticos e dinâmicos do software, e que trata o aspecto paralelismo em todo o ciclo de desenvolvimento. Com esta metodologia é possível a identificação do paralelismo inter-objeto, inter-método e intra-método. O paralelismo inter-objeto se refere ao paralelismo que ocorre entre diferentes objetos. O paralelismo inter-métodos se refere ao paralelismo que ocorre entre diferentes métodos de um mesmo objeto. O paralelismo intra-método se refere ao paralelismo interno a um método específico.

1.1 Fases

O processo de desenvolvimento de software ocorre seguindo algumas fases, sendo estas: requisitos, análise, projeto, implementação e testes. A seguir serão descritas cada uma das fases propostas, bem como seus objetivos.

Requisitos

Esta fase tem como objetivo principal identificar as funcionalidades necessárias para o desenvolvimento do sistema de uma forma adequada e eficiente a partir das necessidades do usuário. Isto pode ser feito, por exemplo, através de entrevistas com o solicitante do produto.

A partir dessas informações, deve-se elaborar uma primeira versão do diagrama de *use case*, não considerando ainda requisitos da implementação. Deve-se, também, identificar, através de descrição textual, os requisitos não funcionais do sistema. Como, por exemplo, aspectos relacionados a características de sistemas distribuídos que tenham sido identificados: paralelismo/concorrência, distribuição. Procede-se à identificação dos tipos de atores e relacionamentos entre *use cases* considerando aspectos funcionais e não funcionais. Os *use cases* e atores poderão ser dos seguintes tipos:

***Use cases* seqüenciais**: *use cases* que agrupam um conjunto de funcionalidades que devem ser executadas seqüencialmente;

***Use cases* paralelos**: *use cases* que agrupam um conjunto de funcionalidades que devem ser executadas em paralelo.

***Use cases* distribuídos**: *use cases* que podem estar em diferentes locais no sistema;

***Use cases* paralelos e distribuídos**: podem estar em diferentes locais do sistema e devem ser executados em paralelo.

Atores exclusivos: atores que estão envolvidos em *use cases* seqüenciais;

Atores paralelos: atores que estão envolvidos em *use cases* paralelos;

Atores distribuídos: atores que se encontram localizados em diferentes nós dentro do sistema;

Atores paralelos e distribuídos: atores que são ao mesmo tempo paralelos e distribuídos;

Relacionamento entre *use cases*: seqüenciais – *use cases* que são executados seqüencialmente uns com os outros; paralelos: *use cases* que são executados paralelamente uns com os outros.

Análise

Esta fase consiste em analisar os requisitos descritos na fase anterior, refinar e estrutura para adquirir um entendimento mais preciso da descrição dos requisitos para ajudar a estruturar o sistema como um todo.

Com base nos requisitos identificados na fase anterior, assim como no diagrama de *use case* elaborado, definem-se as classes e objetos do sistema, identificando-se aspectos de concorrência/paralelismo, distribuição e comunicação entre as classes.

Deve-se também identificar os aspectos de paralelismo, distribuição e sincronização entre pacotes através de descrição textual e elaborar o diagrama de pacotes considerando estes aspectos.

Projeto

Nesta fase o sistema deve ser moldado levando-se em conta os requisitos não funcionais e outras considerações não identificadas nas fases anteriores, como por exemplo: linguagem de programação, interface com o usuário, reuso de componentes, entre outras. Identificam-se também os subsistemas.

O diagrama de classes deve ser analisado levando-se em consideração a concorrência e localização de métodos, classes e objetos.

Implementação

Esta fase tem como objetivo principal a construção/implementação do sistema, tendo como base aspectos identificados nas fases de requisitos, análise e projeto.

Nesta fase, devem ser definidas as interfaces entre os subsistemas identificados na fase de projeto. Devem-se também detalhar e implementar os métodos das classes já identificadas anteriormente. Os mecanismos para controle de sincronização e os que

tratam do balanceamento de carga entre os nós do processamento, considerando os requisitos de distribuição, paralelismo, sincronização e comunicação devem ser identificados.

Testes

Esta fase será melhor especificada em trabalhos futuros.

A subseção seguinte apresenta as representações propostas pela metodologia MDSODI para a especificação do processo de desenvolvimento de software.

1.2 Representações

Além das fases de um processo de desenvolvimento de software, a MDSODI propõe também as representações adequadas para a especificação do mesmo. A tabela A.1 ilustra os tipos de *use-cases* existentes com suas respectivas representações.

TABELA A.1 - Tabela dos tipos de *use cases*

Tipos de *use cases*	
conjunto de funcionalidades que devem ser executadas seqüencialmente.	
***Use cases* paralelos**: *use cases* que agrupam um conjunto de funcionalidades que devem ser executadas em paralelo.	
***Use cases* distribuídos**: *use cases* que podem estar em diferentes locais no sistema.	
***Use cases* paralelos e distribuídos**: podem estar em diferentes locais do sistema e devem ser executados em paralelo.	

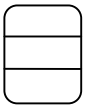
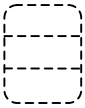
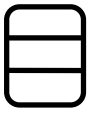
A MDSODI propõe também tipos de atores necessários para tratar os aspectos de sistemas distribuídos. A tabela A.2 mostra os tipos de atores e sua representação gráfica.

A metodologia para desenvolvimento de software distribuído considera também tipos de classes e objetos para tratar das características dos sistemas distribuídos. A tabela A.3 apresenta estes tipos de classes/objetos bem como sua representação gráfica.

TABELA A.2 - Tabela dos tipos de atores

Tipos de atores	Representação
Atores exclusivos: atores envolvidos com *use-cases* seqüenciais.	
Atores paralelos: atores envolvidos em *use-cases* paralelos.	
Atores distribuídos: atores que se encontram localizados em diferentes nós no sistema.	
Atores paralelos e distribuídos: atores que são, ao mesmo tempo, paralelos e distribuídos.	

TABELA A.3 - Tabela dos tipos de classes/objetos

Tipos de classes/objetos	Representação
Classes e Objetos exclusivos: todos os seus métodos são executados seqüencialmente.	
Classes e Objetos parcialmente paralelos: alguns métodos são executados seqüencialmente enquanto outros, concorrentemente.	
Classes e Objetos totalmente paralelos: todos os seus métodos são executados concorrentemente.	
Classes e Objetos distribuídos: localizados em diferentes nós dentro do sistema.	

A tabela A.4, apresentada a seguir, mostra os tipos de relacionamentos. Não estão apresentados os relacionamentos convencionais da orientação a objetos, como, por exemplo, agregação e generalização. Porém, os mesmos também poderão ser utilizados.

TABELA A.4 – Tabela dos tipos de relacionamentos

Tipos de relacionamentos	Representação
Relacionamento entre *use cases* exclusivos: *use cases* que são executados seqüencialmente. (Requisitos)	
Relacionamento entre *use cases* paralelos: *use-cases* que são executados concorrentemente com outros *use-cases*. (Requisitos)	
Relacionamento entre pacotes exclusivos: pacotes que são executados seqüencialmente. (Análise)	
Relacionamento entre pacotes parcialmente paralelos: pacotes que têm algumas classes executadas de forma concorrente com outras classes de outro pacote.(Análise)	
Relacionamento entre pacotes totalmente paralelos: pacotes em que todas as classes são executadas de forma concorrente com classes de outro pacote. (Análise)	
Relacionamento entre pacotes distribuídos: pacotes localizados em diferentes nós do sistema. (Análise)	
Relacionamento entre módulos/subsistemas exclusivos: subsistemas/Módulos que são executados seqüencialmente. (Projeto)	
Relacionamento entre módulos/subsistemas parcialmente paralelos: subsistemas/módulos que têm algumas classes executadas de forma concorrente com outras classes de outro subsistema. (Projeto)	
Relacionamento entre módulos/subsistemas totalmente paralelos: subsistemas/módulos em que todas as classes são executadas de forma concorrente com classes de outro subsistema.(Análise)	
Relacionamento entre subsistemas/módulos distribuídos: subsistemas/módulos localizados em diferentes nós dentro do sistema. (Projeto)	
Relacionamento entre objetos exclusivos: objetos que têm seus métodos executados seqüencialmente.	
Relacionamento entre objetos parcialmente paralelos: objetos que têm alguns métodos executados concorrentemente.	
Relacionamento entre objetos totalmente paralelos: objetos que têm todos os seus métodos executados concorrentemente.	
Relacionamento entre objetos distribuídos: objetos que se encontram localizados em diferentes nós do sistema.	

Bibliografia

[ALT 99] ALTMANN, J.; POMBERGER, G. Cooperative Software Development: Concepts, Model and Tools. In: TOOLS, 30. 1999, Santa Barbara. **Proceedings…** Santa Barbara: IEEE Society Press, 1999.

[ARA 97] ARAUJO, R. M.; DIAS, M. S.; BORGES, M. R. S. Suporte por Computador ao Desenvolvimento Cooperativo de Software: Classificação e Propostas. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 11., 1997, Fortaleza. **Anais...** Fortaleza: SBC, 1997. p. 299-314.

[ASK 2001] ASKLUND, U.; BENDIX, L. Configuration Management for Open Source Software. In: WORKSHOP ON OPEN SOURCE SOFTWARE ENGINEERING, 2001, Toronto. **Proceedings...** Toronto, Canada: [s.n.], 2001.

[BAN 92] BANDINELLI, S.; FUGGETTA, A.; GHEZZI, C. Process Enactment in SPADE. In: EUROPEAN WORKSHOP ON SOFTWARE PROCESS TECHNOLOGY, EWSPT, 2., 1992, Trondheim, Norway. **Software Process Technology**: proceedings. Berlin: Springer-Verlag, 1992.

[BAT 2002] BATISTA, S. M. **Proposta de uma Ferramenta de *Groupware* para Apoio à Definição de Requisitos para a MDSODI**. 2002. Exame de Qualificação (Mestrado) - DIN-UEM/UFPR, Maringá.

[BEL 99] BELLIFEMINE, F.; POGGI, A.; RIMASSA, G. JADE – A FIPA-Compliant Agent Framework. In: INTERNATIONAL CONFERENCE AND EXHIBITION ON THE PRACTICAL APPLICATION OF INTELLIGENT AGENTS AND MULTI-AGENTS, 4., 1999, London. **Proceedings...** London: [s.n.], 1999.

[BEL 2000] BELLIFEMINE, F.; POGGI, A.; RIMASSA, G. An Object-Oriented Framework to Realize Agent Systems. In: WOA WORKSHOP, 2000, Parma. **Proceedings…** Parma: [s.n.], 2000.

[BOO 2000] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML:** guia do usuário. Rio de Janeiro: Campus, 2000.

[BOR 2001] BORDINI, R. H. Inteligência Artificial Distribuída (Uma Introdução aos Sistemas Multiagentes). In: **IX ESCOLA DE INFORMÁTICA DA SBC SUL (ERI2001)**, Porto Alegre, Instituto de Informática – UFRGS, 2001, v.1, p. 1 – 18.

[BOR 95] BORGES, M. R. S.; CAVALCANTI, M. C. R.; CAMPOS, M. L. M. **Suporte por Computador ao Trabalho Cooperativo**. Canela-RS: Instituto de Informática da UFRGS, 1995. Trabalho apresentado na 14. Jornada de Atualização em Informática, 1994, Canela.

[BRE 98] BRENNER, W.; ZARNEKOW, R.; WITTIG, H. **Intelligent Software Agents:** foundations and applications. Berlin: Springer-Verlag, 1998.

[BRO 92] BROWN, A.; EARL, A.; MCDERMID, J. **Software Engineering Environments:** automated support for software engineering. London: Mc-Graw Hill, 1992.

[CHE 2001] CHEESMAN, J.; DANIELS, J. **UML Components, A Simple Process for Specifying Component-Based Software.** [S.l.]: Addison-Wesley, 2001.

[CON 94] CONRADI, R. et al. EPOS: Object-Oriented Cooperative Process Modeling. In: FINKELSTEIN, A. et al. (Ed.). **Software Process Modeling and Technology**. Taunton: Research Studies Press, 1994. p. 33-70.

[COU 2001] COULOURIS, G. et al. **Distributed Systems**. England: Pearson Education, 2001.

[CUG 99] CUGOLA, G.; GHEZZI, C. Design and Implementation of PROSYT: a Distributed Process Support System. In: INTERNATIONAL WORKSHOP ON ENABLING TECHNOLOGIES: INFRASTRUCTURE FOR COLLABORATIVE ENTERPRISES, 1999. **Proceedings…** Disponível em: <http://www.elet.polimi.it/Users/DEI/Sections/CompEng/Gianpaolo.Cugola/Papers/wetice99-final.pdf>. Acesso em: 28 fev. 2002.

[D'SO 99] D'SOUZA, D.; WILLS, A. **Objects, Components and Frameworks with UML – The Catalysis Approach.** [S.l.]: Addison-Wesley Publishing Company, 1999. 816 p.

[FER 95] FERNANDES, A. **Gerência de Software através de Métricas:** garantindo a qualidade do projeto, processo e produto. São Paulo: Atlas, 1995.

[FIN 93] FININ, T. et al. **Draft Specification of the KQML Agent-Communication Language**. The DARPA Knowledge Sharing Initiative External Interfaces Working Group. 1993. Disponível em: <http://citeseer.nj.nec.com/finin93draft.html>. Acesso em: 27 fev. 2002.

[FIP 2001] FIPA. **Agent Management Specification (FIPA00023)**. 2001. Disponível em: <http://www.fipa.org/specs>. Acesso em: 01 dez. 2001.

[FIP 2001a] FIPA. **Agent Message Transport Service Specification (FIPA00067)**. 2001. Disponível em: <http://www.fipa.org/specs>. Acesso em: 11 jan. 2002.

[FIP 2001b] FIPA. **Abstract Architecture Specification (FIPA00001)**. 2001. Disponível em: <http://www.fipa.org/specs>. Acesso em: 05 dez. 2001.

[FIP 2001c] FIPA. **ACL Message Structure Specification (FIPA00002)**. 2001. Disponível em: <http://www.fipa.org/specs>. Acesso em: 05 dez. 2001.

[FIP 2001d] FIPA. **Ontology Service Specification (FIPA00086)**. 2001. Disponível em: <http://www.fipa.org/specs>. Acesso em: 11 jan. 2002.

[FIP 2002] FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. Disponível em: <http://www.fipa.org>. Acesso em: 03 jan. 2002.

[FIP 2002a] FIPA. **Communicative Act Library Specification (FIPA00037)**. 2002. Disponível em: <http://www.fipa.org/specs>. Acesso em: 08 jun. 2002.

[FIP 2002b] FIPA. **Contract Net Interaction Protocol Specification (FIPA00029)**. 2002. Disponível em: <http://www.fipa.org/specs>. Acesso em: 20 jun. 2002.

[FLO 99] FLORES-MENDEZ, R. A. Towards the Standardization of Multi-Agent System Architectures: an overview. **Communications of the ACM Crossroads, Special Issue on Intelligent Agents, Association for Computer Machinery (ACM**), Issue 5.4, pp. 18-24, Summer 1999.

[FOW 2000] FOWLER, M. **UML Essencial:** um breve guia para a linguagem-padrão de modelagem de objetos. 2. ed. Porto Alegre: Bookman, 2000.

[GAR 2001] GARCIA, A. et al. An Aspect-Based Approach for Developing Multi-Agent Object-Oriented Systems. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 15., 2001, Rio de Janeiro. **Anais...** Rio de Janeiro: SBC, 2001. p. 177-192.

[GIM 94] GIMENES. I. M. S. **Uma Introdução ao Processo de Engenharia de Software**: Ambientes e Formalismos. Caxambu-MG: SBC, 1994. 42f. Trabalho apresentado na 13. Jornada de Atualização em Informática, 1994.

[GRA 2000] GRAVENA, J. P. **Aspectos Importantes de uma Metodologia para Desenvolvimento de Software com Objetos Distribuídos**. Trabalho de Graduação, Universidade Estadual de Maringá, Departamento de Informática, 2000.

[GUT 96] GUTWIN, C.; GREENBERG, S. Workspace Awareness for Groupware. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEM, CHI, 1996, Vancouver, Canada. **Proceedings…** Vancouver, Canada: ACM Press, 1996. p. 208-209.

[GUT 96a] GUTWIN, Carl. **Workspace Awareness Research.** Feb. 1996. Disponível em: <http://www.cpsc.ucalgary.ca/projects/grouplab/people/carl/research/awareness.html>. Acesso em: 09 jan. 2002.

[HUZ 95] HUZITA, E. H. M., **MOOPP – Uma Metodologia para Auxiliar o Desenvolvimento de Aplicações para Processamento Paralelo.** 1995. 213 p. Tese (Doutorado) - Escola Politécnica, USP, São Paulo.

[HUZ 99] HUZITA, E. H. M. **Uma Metodologia para o Desenvolvimento Baseado em Objetos Distribuídos Inteligentes**. Projeto de pesquisa em andamento, Universidade Estadual de Maringá. Departamento de Informática, 1999.

[IKV 2002] IKV++ TECHNOLOGIES. **Grasshopper**. Disponível em: <http://www.ikv.de>. Acesso em: 01 jul. 2002.

[JAC 97] JACOBSON, I.; GRISS M.; JONSSON P. **Software Reuse**. [S.l.]: Addison Wesley, 1997.

[JAC 99] JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **The Unified Software Development Process.** [S.l.]: Addison-Wesley, 1999.

[JEN 98] JENNINGS, N. R.; WOOLDRIDGE, M. **Agent Technology:** foundations, applications and markets. Berlin: Springer Verlag, 1998.

[JEN 2001] JENNINGS, N. An Agent-Based Approach for Building Complex Software Systems. **Communications of the ACM**, New York, v. 44, n.4, p. 35-41, Apr. 2001.

[LAB 97] LABROU, Y.; FININ, T. **A Proposal for a new KQML Specification**. Baltimore: University of Maryland, Computer Science and Electrical Engineering Department, 1997. (Technical Report TR CS-97-03).

[LAR 2000] LARMAN, Craig. **Utilizando UML e padrões:** uma introdução à análise e ao projeto orientados a objetos. Porto Alegre: Bookman, 2000.

[LIM 98] LIMA, C. A. **Um Gerenciador de Processos de Software para o Ambiente PROSOFT**. 1998. 197 p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[MIC 2002] MICROSOFT CORPORATION. **Distributed Component Object Model**. Disponível em: <http://www.microsoft.com/com/tech/dcom.asp>. Acesso em: 05 abr. 2002.

[MON 94] MONTANGERO, C.; AMBRIOLA, V. OIKOS: Constructing Process-Centered SDEs. In: **Software Process Modeling and Technology**, Research Studies Press, London, U. K., 1994.

[MOR 2002] MORO, C. F. **Proposta de um Repositório de Metadados para Ambiente de Desenvolvimento de Software Distribuído**. 2002. Exame de Qualificação (Mestrado) - DIN-UEM/UFPR, Maringá.

[MOU 92] MOURA, L. M. V.; ROCHA, A. R. C. **Ambientes de Desenvolvimento de Software**. Rio de Janeiro: COPPE/UFRJ, 1992. (ES-271/92).

[NUI 2001] NUNES, I. D. **Componentes de Percepção para o Ambiente PROSOFT Cooperativo**. 2001. 136 p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[NUN 94] NUNES, D. J. **PROSOFT**. Porto Alegre: CPGCC da UFRGS, 1994. Relatório de pesquisa interno.

[NWA 96] NWANA, H. Software Agents: an overview. **Knowledge Engineering Review**, [S.l.], v. 11, n.3, p. 1-40, Sept. 1996.

[NWA 99] NWANA, et al. ZEUS: A Toolkit for Building Distributed Multi-Agent Systems. **Applied Artificial Intelligence Journal**, Hingham, v. 13, n. 1, p., 1999.

[ODE 2000] ODELL, J.; PARUNAK, H. Van Dyke; BAUER, B. Extending UML for Agents. In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, AAAI, 17., 2000, Austin. **Proceedings…** Menlo Parle: AAAI Press, 2000.

[OMG 2000] OMG - Object Management Group – Agent Platform Special Interest Group. **Agent Technology – Green Paper**. Version 1.0, September 2000.

[OMG 2002] **OMG – Object Management Group**. Disponível em <http://omg.org>. Acesso em: 05 abr. 2002.

[ORF 96] ORFALI, H.E. J. **The Essential Distributed Objects Survival Guide.** New York: John Wiley&Sons, 1996.

[ORF 99] ORFALI, R. et al. **Client/Server Survival Guide**. New York: John Wiley&Sons, 1999.

[PAS 2001] PASCUTTI, M. C. D. **Fundamentos para uma Proposta de Definição de Arquitetura de Software Usando uma Metodologia de Desenvolvimento de Software Distribuído Baseada em Agentes**. 2001. 60 p. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[PAT 92] PATIL, R.S.; et al. The DARPA Knowledge Sharing Effort: Progress Report. In: THE ANNUAL INTERNATIONAL CONFERENCE ON KNOWLEDGE ACQUISITION, 1992, Cambridge MA. **Proceedings…** Cambridge MA: [s.n.], 1992.

[PED 2002] PEDRAS, M. E. V. **Uma Ferramenta de Apoio ao Gerenciamento de Desenvolvimento de Software Distribuído**. 2002. Exame de Qualificação (Mestrado) - DIN-UEM/UFPR, Maringá.

[PIN 2001] PINHEIRO, M. K.; LIMA, J. V.; BORGES, M. R. S. *Awareness* em Sistemas de *Groupware*. In: JORNADAS IBEROAMERICANAS DE INGENIERIA DE REQUISITOS Y AMBIENTES DE SOFTWARE, IDEAS, 4., Santo Domingo, Costa Rica. **Memorias...** San Jose, Costa Rica: CIT, 2001. p. 323-335.

[REI 98] REIS, C.; REIS, R.; NUNES, D. Gerenciamento do Processo de Desenvolvimento Cooperativo de Software no Ambiente PROSOFT. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 12., 1998, Maringá. **Anais...** Maringá: SBC, 1998. p. 221-236

[REI 98a] REIS, R. Q. **Uma Proposta de Suporte ao Desenvolvimento Cooperativo de Software no Ambiente PROSOFT**. 1998. 177 p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[REI 2000] REIS, C. A.; REIS, R. Q.; NUNES, D. J. Evolução do Ambiente PROSOFT para apoiar aspectos de Distribuição, Cooperação e Automação do Processo de Desenvolvimento de Software. In: JORNADAS IBEROAMERICANAS DE INGENIERÍA DE REQUISITOS Y AMBIENTES SOFTWARE, IDEAS, 3., 2000, Cancún, México. **Proceedings...** Cuernavaca: Centro Nacional de Investigación y Desarollo Tecnológico, 2000.

[REI 2000a] REIS, C. A. L. **Ambientes de Desenvolvimento de Software e seus Mecanismos de Execução de Processos de Software**. 2000, 114 p. Exame de Qualificação (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[SAM 98] SAMETINGER, J. **Software Engineering with Reusable Components**. New York: Springer-Verlag, 1998.

[SIL 2001] SILVA, L. F. da; PAULA, V. C. C. de. Um Meta-Modelo para Especificação de Arquiteturas de Software em Camadas. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 15., 2001, Rio de Janeiro. **Anais...** Rio de Janeiro: SBC, 2001. p.132-144.

[SOH 98] SOHLENKAMP, M. **Supporting group awareness in multi-user**

environment through perceptualization. 1998. Dissertation. Fachbereich Mathematik-Informatik der Universität – Gesamthochschule – Paderborn.

[SUN 2002] SUN MICROSYSTEMS. **The Source for Java™ Technology**. Disponível em: <http://www.java.sun.com>. Acesso em: 05 abr. 2002.

[SZY 99] SZYPERSKI, C. **Component Software:** beyond object-oriented programming. [S.l.]: Addison-Wesley, 1999.

[TRA 94] TRAVASSOS, G. H. **O Modelo de Integração de Ferramentas da Estação TABA**. 1994. Tese (D. Sc.) - COPPE/UFRJ, Rio de Janeiro.

[VES 95] VESSEY, I.; SRAVANAPUDI, A.J. CASE Tools as Collaborative Support Technologies. **Communications of the ACM**, New York, v. 38, n.1, p. 83-95, Jan. 1995.

[WAN 2000] WANG, A. I. Using Software Agents to Support Evolution of Distributed Workflow Models. In: INTERNATIONAL ICSC SYMPOSIUM ON INTERACTIVE AND COLLABORATIVE COMPUTING, ICC, 2000, Wollongon, Australia. **Proceedings…** Wollongong, Australia: [s.n.], 2000.

[WAN 2000a] WANG, A. I. Experience paper: Implementing a Multi-Agent Architecture for Cooperative Software Engineering. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, SEKE, 12., 2000, Chicago, USA. **Proceedings…** Chicago: [s.n.], 2000.

[WAN 2001] WANG, A. I. **Using a Mobile, Agent-based Environment to support Cooperative Software Processes**. 2001. 431 p. Thesis – Dept. of Computer and Information Science, Norwegian University for Science and Technology, Trondheim, Norway.

[WOO 95] WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent Agents: Theory and Practice. **Knowledge Engineering Review**, 1995. Disponível em: <http://www.ecs.soton.ac.uk/~nrj/pubs.html#1995>. Acesso em: 05 mar. 2001.

[WOO 2000] WOOLDRIDGE, M.; JENNINGS, N. R. The Gaia Methodology for Agent-Oriented Analysis and Design. **Journal of Autonomous Agent and Multi-Agents Systems**. [S.l.], v.3, n. 3, p. 285-312, 2000.

[YOU 90] YOURDON, E. **Análise Estruturada Moderna**. Rio de Janeiro: Campus, 1990.