UNIVERSIDADE ESTADUAL DE MARINGÁ
DEPARTAMENTO DE INFORMÁTICA
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

Flávio Luiz Schiavoni

FRADE – *Framework* para infra-estrutura de um Ambiente Distribuído de Desenvolvimento de *Software*

MARINGÁ
2007

Livros Grátis

http://www.livrosgratis.com.br

Milhares de livros grátis para download.

Flávio Luiz Schiavoni

FRADE – *Framework* para infra-estrutura de um Ambiente Distribuído de Desenvolvimento de *Software*

Dissertação apresentada à Universidade Estadual de Maringá, como requisito para a obtenção do grau de Mestre em Ciência da Computação.

Orientadora: Profa. Dra. Elisa Hatsue Moriya Huzita

MARINGÁ

2007

Dados Internacionais de Catalogação-na-Publicação (CIP)
(Biblioteca Central - UEM, Maringá – PR., Brasil)

Schiavoni, Flávio Luiz

S329f Frade - *framework* para infra-estrutura de um ambiente distribuído de desenvolvimento de *software* / Flávio Luiz Schiavoni. -- Maringá : [s.n.], 2007.

181 p. : figs.

Orientadora : Profª. Drª. Elisa Hatsue Moriya Huzita.

Dissertação (mestrado) - Universidade Estadual de Maringá. Programa de Pós-graduação em Ciência da Computação, 2007.

1. Engenharia de software. 2. Sistemas distribuidos. 3. Software - Desenvolvimento. 4. Ambiente de desenvolvimento de software. 5. Framework. 6. Software - Arquitetura. I. Universidade Estadual de Maringá. Programa de Pós-graduação em Ciência da Computação. II. Título.

CDD 22.ed. 005.12

FLÁVIO LUIZ SCHIAVONI

FRADE – FRAMEWORK PARA INFRA-ESTRUTURA DE UM AMBIENTE DISTRIBUÍDO DE DESENVOLVIMENTO DE SOFTWARE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Aprovado em 05/06/2007.

BANCA EXAMINADORA

Profa. Dra. Elisa Hatsue Moriya Huzita
Universidade Estadual de Maringá – DIN/UEM

Profa. Dra. Tania Fatima Calvi Tait
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Jorge Luis Nicolas Audy
Pontifícia Universidade Católica do Rio Grande do Sul – PRPPG/PUCRS

Aos Schiavoni's,
grandes guerreiros
que justificam qualquer esforço.

AGRADECIMENTOS

Espero não esquecer de ninguém nesta seção. Começo me preocupando com isto.

À minha namorada, noiva e esposa Eliete Maria de Carvalho Schiavoni, que durante este mestrado me fez seu noive e seu esposo.

Aos meus pais, Adélcio e Vildair, por toda a força.

Aos meus irmão Marcos e André, os cunhados e cunhadas, Patrícia, Luciana, Sérgio, Simone e Myrcea.

Aos meus sogros Sabino e Neyde.

A minha orientadora Professora Doutora Elisa Hatsue Moriya Huzita que me acompanhou e me aconselhou durante todo o meu mestrado.

À professora Tânia que acompanhou de perto este trabalho.

Ao pessoal da AOCP, empresa aonde trabalho, que além de me liberar quando precisei financiou inúmeras cópias deste documento. Valeu Chefes!

Ao povo da FAFIMAN, faculdade aonde leciono, professores do departamento, demais professores e meus alunos que são motivação para trabalhar.

À minha turma de mestrado, grandes colegas que ajudaram a PAA se tornar mais fácil.

Aos amigos de rock que sempre ajudam a refrescar a cabeça.

Bem, um bolo deste não se faz sozinho. Aos que efetivamente colocaram a mão na massa:

Adriana Herden, grande engenheira de *software*, que me ajudou com a camada de negócios logo no início;

Éderson Amorim que implementou o serviço de persistência;

Igor Steinmacher que também colaborou com a reimplementação do ambiente;

César que cuidou da seção de artefatos;

William que implementou toda a camada de comunicação e muito mais;

Lucia que cuidou da camada de negócios;

Igor Wiese, UML sênior, que garantiu a interoperabilidade;

Gustavo que implementou várias ferramentas;

Ana que está chegando para a PML;

Rogério Pozza (tiozão) que projetou o *workspace*.

Obrigado.

RESUMO

O Desenvolvimento de *Software*, proposto ao longo de anos necessita de várias ferramentas para o seu gerenciamento. No caso de ambientes de desenvolvimento distribuído de *software*, suas ferramentas de gerenciamento devem estar disponíveis para acesso remoto e devem dispor de mecanismos para apoiar a distribuição que estes se propõem a ter. Para disponibilizar estas ferramentas de apoio aos desenvolvedores há a necessidade de uma infra-estrutura que gerencie a comunicação entre os participantes do ambiente. Este trabalho propõe um *framework* para esta infra-estrutura e contém componentes que cuidam: do controle de versões de artefatos, gerenciamento da informação dos processos de desenvolvimento de *software* e gerenciamento das atividades de um projeto; do provimento de suporte para atividades cooperativas e suporte para a configuração dinâmica de um ambiente de desenvolvimento de *software*. A experimentação e implementação foi feita com o Ambiente de Engenharia de *Software* Distribuído – DiSEN (*Distributed Software Engineering Environment*). O DiSEN é um ambiente de desenvolvimento distribuído de *software* que visa disponibilizar, para equipes de desenvolvimento, ferramentas para apoiar o trabalho cooperativo e é desenvolvido pelo mesmo grupo de pesquisa no qual este trabalho está inserido.

Palavras-Chave: Ambiente Distribuído de Desenvolvimento de *Software*. *Framework*. DiSEN. DBC. Infra-estrutura. Gerenciadores.

ABSTRACT

The software development needs many management tools. To the software distributed development environment these tools must be available to remote access and must have features to stand the distribution. To dispose these stand tools to the developers there is a needing of an infrastructure that manages the communication among the environment workers. This dissertation presents a framework to this infrastructure that will have components that cares about the software configuration management, process information management, project activities management, cooperative tasks stands and dynamical configuration of the software environment. The experimentation and the implementation was done with DiSEN - *Distributed Software Engineering Environment*. DiSEN is a distributed PSEE that is developed by the same research group of this dissertation.

Key-words: Distributed PSEE. *Framework*. DiSEN. CBD. Infrastructure. Managers.

LISTA DE ILUSTRAÇÕES

FIGURA 1 – METODOLOGIA DO TRABALHO 21
FIGURA 2 – CAMADAS LÓGICAS DO *FRAMEWORK* 66
FIGURA 3 – CASO DE USO PARA OS SUPORTES LOCAIS 67
FIGURA 4 – ARQUITETURA INICIAL PROPOSTA PARA O DISEN 68
FIGURA 5 – POSICIONAMENTO DA CAMADA DE APLICAÇÃO 69
FIGURA 6 – INTERFACE DA CAMADA DE APLICAÇÃO 71
FIGURA 7 – ARQUIVO XML PARA CONFIGURAÇÃO DA CAMADA DE APLICAÇÃO 71
FIGURA 8 – INTERFACES DOS COMPONENTES DA CAMADA DE APLICAÇÃO 72
FIGURA 9 – POSICIONAMENTO DA CAMADA DE NEGÓCIOS 73
FIGURA 10– GERENCIADOR DE OBJETOS PROPOSTO POR PASCUTTI (2002) 74
FIGURA 11 – GERENCIADOR DE OBJETOS DO FRADE 75
FIGURA 12 – ITERFACE DO GERENCIADOR DE OBJETOS 75
FIGURA 13 – ARQUIVO DE CONFIGURAÇÃO DO GERENCIADOR DE OBJETOS 76
FIGURA 14 – INTERFACE DOS GERENCIADORES DO *FRAMEWORK* 76
FIGURA 15 – INTERFACE DOS OBJETOS DE NEGÓCIOS DOS GERENCIADORES 77
FIGURA 16 – INTERFACE DA CAMADA DE NEGÓCIOS 77
FIGURA 17 – DIAGRAMA DE PACOTES DO GERENCIADOR DE PROCESSOS 78
FIGURA 18 – CASOS DE USO DO GERENCIADOR DE PROCESSOS 79
FIGURA 19 – DEPENDÊNCIA DO GERENCIADOR DE PROCESSO 79
FIGURA 20 – ACESSO DO GERENCIADOR DE PROCESSO AOS SUPORTES 79
FIGURA 21 – MÉTODO INSERT NA CLASSE PROCESSOBO 80
FIGURA 22 – INTERFACE DO GERENCIADOR DE PROCESSO 80
FIGURA 23 – DIAGRAMA DE PACOTES DO GERENCIADOR DE PROJETOS 81
FIGURA 24 – CASOS DE USO DO GERENCIADOR DE PROJETOS 81
FIGURA 25 – DEPENDÊNCIAS DO GERENCIADOR DE PROJETOS 82
FIGURA 26 – ESTRUTURA DO GERENCIADOR DE PROJETOS 82
FIGURA 27 – ACESSO DO GERENCIADOR DE PROJETOS AOS SUPORTES 82
FIGURA 28 – INTERFACE DO GERENCIADOR DE PROJETOS 82
FIGURA 29 – DIAGRAMA DE PACOTES DO GERENCIADOR DE ARTEFATOS 84
FIGURA 30 – CASOS DE USO DO GERENCIADOR DE ARTEFATOS 84
FIGURA 31 – DEPENDÊNCIA DO GERENCIADOR DE ARTEFATOS 84
FIGURA 32 – ACESSO DO GERENCIADOR DE ARTEFATOS AOS SUPORTES 85
FIGURA 33 – IMPLEMENTAÇÃO DO ARTEFATOBO 85
FIGURA 34 – INTERFACE DO GERENCIADOR DE ARTEFATOS 85
FIGURA 35 – DIAGRAMA DE PACOTES DO GERENCIADOR DE RECURSOS 87
FIGURA 36 – CASOS DE USO DO GERENCIDOR DE RECURSOS 87
FIGURA 37 – DEPENDÊNCIA DO GERENCIADOR DE RECURSO 87
FIGURA 38 – ACESSO DO GERENCIADOR DE RECURSOS AO SUPORTE 88
FIGURA 39 – INTERFACE DO GERENCIADOR DE RECURSOS 88
FIGURA 40 – DIAGRAMA DE PACOTES DO GERENCIADOR DE LOCAIS 89
FIGURA 41 – DEPENDÊNCIA DO GERENCIADOR DE LOCAIS 90
FIGURA 42 – CASOS DE USO DO GERENCIADOR DE LOCAIS 90
FIGURA 43 – ACESSO DO GERENCIADOR DE LOCAIS AOS SUPORTES 91
FIGURA 44 – REGRAS DE NEGÓCIO PARA LOCAIS: LOCALBO 91
FIGURA 45 – INTERFACE DO GERENCIADOR DE LOCAIS 91
FIGURA 46 – DIAGRAMA DE PACOTES DO GERENCIADOR DE USUÁRIOS. 93
FIGURA 47 – DEPENDÊNCIAS DO GERENCIADOR DE USUÁRIOS 93

FIGURA 48 – CASOS DE USO DO GERENCIADOR DE USUÁRIOS 94
FIGURA 49 – ACESSO DO GERENCIADOR DE USUÁRIOS AOS SUPORTES 94
FIGURA 50 – REGRAS DE NEGÓCIOS PARA USUARIOS: CLASSE USUARIOBO 95
FIGURA 51 – INTERFACE DO GERENCIADOR DE USUÁRIOS 95
FIGURA 52 – DIAGRAMA DE PACOTES DO GERENCIADOR DE FERRAMENTAS....97
FIGURA 53 – DEPENDÊNCIA DO GERENCIADOR DE FERRAMENTAS....97
FIGURA 54 – FERRAMENTAS COM SUPORTE A COMUNICAÇÃO 98
FIGURA 55 – CASOS DE USO DO GERENCIADOR DE FERRAMENTAS....98
FIGURA 56 – ACESSO DO GERENCIADOR DE FERRAMENTAS AOS SUPORTES....99
FIGURA 57 – REGRAS DE NEGÓCIOS PARA FERRAMENTAS: CLASSE FERRAMENTABO....99
FIGURA 58 – INTERFACE DO GERENCIADOR DE FERRAMENTAS 99
FIGURA 59 – CAMADA DE INFRA-ESTRUTURA....100
FIGURA 60 – MAPEAMENTO DOS GERENCIADORES DE NEGÓCIO PARA A CAMADA DE INFRA-ESTRUTURA 101
FIGURA 61 – INTERFACE DOS SERVIÇOS DO *FRAMEWORK* 101
FIGURA 62 – ATIVAÇÃO DE SERVIÇOS REMOTO PELO SUPORTE 102
FIGURA 63 – INTERFACE DOS SUPORTES DO *FRAMEWORK*....102
FIGURA 64 – INTERFACE DA CAMADA DE INFRA-ESTRUTURA 103
FIGURA 65 – ARQUIVO XML DE CONFIGURAÇÃO DA CAMADA DE INFRA-ESTRUTURA 103
FIGURA 66 – INTERFACE PARA SUPORTE A PERSISTÊNCIA UTILIZANDO DAOS 104
FIGURA 67 – CLASSE ENTIDADE: REPRESENTA OS OBJETOS PERSISTENTES DO *FRAMEWORK* 104
FIGURA 68 – INTERFACE DO SUPORTE A PERSISTÊNCIA....105
FIGURA 69 – INTERFACE DO SERVIÇO DE PERSISTÊNCIA 106
FIGURA 70 – SERVIÇO A PERSISTÊNCIA: CLASSE HIBERNATEDAO 107
FIGURA 71 – MÉTODO INSERT DA CLASSE HIBERNATEDAO 107
FIGURA 72 – MAPEAMENTO OBJETO-RELACIONAL DA CLASSE ARTEFATO 108
FIGURA 73 – INTERFACE DO SUPORTE A ARTEFATOS....109
FIGURA 74– INTERFACE DO SERVIÇO DE ARTEFATOS 113
FIGURA 75 – ARQUIVO DE CONFIGURAÇÃO DAS FERRAMENTAS INSTALADAS NO AMBIENTE 115
FIGURA 76 – CONTEÚDO DO PACOTE DE INSTALAÇÃO DOS MÓDULOS 116
FIGURA 77 – INTERFACE DO SUPORTE A ATUALIZAÇÃO....117
FIGURA 78 – ESTRUTURA DO REPOSITÓRIO DE ATUALIZAÇÕES....118
FIGURA 79 – INTERFACE DO SERVIÇO DE ATUALIZAÇÃO 118
FIGURA 80 – INTERFACE DO SUPORTE A ACESSO 120
FIGURA 81 – CLASSE QUE REPRESENTA O USUÁRIO DO SISTEMA: SYSTEMUSER....121
FIGURA 82 – INTERFACE DO SERVIÇO DE ACESSO 122
FIGURA 83 – INTERFACE DO SUPORTE À COMUNICAÇÃO....123
FIGURA 84 – INTERFACE DO SERVIÇO DE COMUNICAÇÃO 125
FIGURA 85 – FUNCIONAMENTO DA COMUNICAÇÃO COM OBJETOS REMOTOS 125
FIGURA 86 – INTERFACE DO SUPORTE A DISTRIBUIÇÃO 127
FIGURA 87 – INTERFACE DO SERVIÇO DE DISTRIBUIÇÃO....128
FIGURA 88 – CAMADA DE COMUNICAÇÃO 129
FIGURA 89 – ESTRUTURA DA CAMADA DE COMUNICAÇÃO 131
FIGURA 90 – INTERFACE DA CAMADA DE COMUNICAÇÃO....132
FIGURA 91 – CLASSE MENSAGEM 132
FIGURA 92 – CLASSES MENSAGENS PADRÃO PARA A COMUNICAÇÃO SÍNCRONA 133
FIGURA 93 – CLASSE COMPRESSÃO USADA PARA COMPACTAR AS MENSAGENS DA COMUNICAÇÃO....133
FIGURA 94 – DIAGRAMA DE CLASSE DO COMPONENTE DE TRANSPORTE 134
FIGURA 95 – COMPONENTES DA CAMADA DE APLICAÇÃO 134

FIGURA 96 – COMPONENTES DA CAMADA DE NEGÓCIOS 134
FIGURA 97 – DEPENDÊNCIA DOS GERENCIADORES DA CAMADA DE NEGÓCIO 135
FIGURA 98 – COMPONENTES DA CAMADA DE INFRA-ESTRUTURA 135
FIGURA 99 – DEPENDÊNCIA DOS GERENCIADORES E OS SUPORTES 135
FIGURA 100 – DEPENDÊNCIA ENTRE OS SUPORTES 136
FIGURA 101 – COMPONENTES DO *WORKSPACE* 136
FIGURA 102 – COMPONENTES DA CAMADA DE COMUNICAÇÃO..137
FIGURA 103 – DEPENDÊNCIAS DA CAMADA DE COMUNICAÇÃO 137
FIGURA 104 – FÁBRICA ABSTRATA PARA A INSTANCIAÇÃO DO *FRAMEWORK*...........................143
FIGURA 105 – CLASSE GLOBAL: REFERÊNCIA A TODAS AS CAMADAS INSTANCIADAS 143
FIGURA 106 – DIAGRAMA DE ATIVIDADES: GLOBAL 144
FIGURA 107 – DIAGRAMA DE ATIVIDADES: CONFIGURAR..145
FIGURA 108 – DIAGRAMA DE ATIVIDADES: AUTENTICAR..145
FIGURA 109 – DIAGRAMA DE ATIVIDADES: LOADMODULES 146
FIGURA 110 – DIAGRAMA DE ATIVIDADES: EXECUTAR TELAS 147

LISTA DE TABELAS

TABELA 1 – LINGUAGENS DE PROGRAMAÇÃO DOS PSEE ESTUDADOS 60
TABELA 2 – PROTOCOLO DE COMUNICAÇÃO 60
TABELA 3 – TIPOS DE COMUNICAÇÃO..61
TABELA 4 – SISTEMA DE ARMAZENAMENTO DE DADOS (META-DADOS / ARTEFATOS).............62
TABELA 5 – REQUISITOS DESEJÁVEIS PARA UM ADDS...63
TABELA 6 – MAPEAMENTO ENTRE OS MÉTODOS DO SERVIÇO DE ARTEFATOS E O REPOSITÓRIO CVS..114
TABELA 7 – CONFIGURAÇÃO E ATUALIZAÇÃO DO AMBIENTE E SEUS ARQUIVOS DE CONFIGURAÇÃO 116
TABELA 8 – AVALIAÇÃO DOS REQUISITOS DESEJÁVEIS PARA ADDS NO FRADE 138
TABELA 9 – RESUMO DOS EXPERIMENTOS REALIZADOS 149

LISTA DE ABREVIATURAS

ADDS	Ambiente Distribuído De Desenvolvimento De *Software*
ADS	Ambientes de Desenvolvimento de *Software*
ADSOD	Ambiente de Desenvolvimento de *Software* Orientado a Domínio
ADSOrg	Ambiente de Desenvolvimento de *Software* Orientado a Organização
API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
BD	Banco de Dados
BO	*Business Object*
CASE	*Computer-Aided Software Engineering*
CORBA	*Common Object Request Broker Architecture*
CSCW	*Computer Supported Cooperative Work*
CSE	*Cooperative Software Engineering*
CVS	*Control Version Service*
DAO	*Data Access Object*
DBC	Desenvolvimento Baseado em Componentes
DiSEN	*Distributed Software Engineering Environment*
ECMA	*European Computes Manufactures Association*
ESD	*Eletronic Software Distribution*
FTP	*File Transfer Protocol*
GCS	Gerência de Configuração de *Software*
GNU	*GNU Not Unix*
GP	Gerenciamento de Projeto
GUI	*Graphical User Interface*
HTTP	*HiperText Transfer Protocol*
ICS	Itens de Configuração de *Software*
IDL	*Interface Definition Language*
IP	*Internal Protocol*
ISO	*International Organization for Standardization*
J2EE	*Java to Enterprise Edition*
JEDI	*Java Event-Based Distributed Infrastructure*
JSP	*Java Server Pages*
LAN	*Local Area Network*
MVC	*Model-View-Control*

OMS	*Object Management System*
OO	Orientação a Objetos
OODB	*Object Oriented Data Base*
ORB	*Object Request Broker*
OSI	*Open System Interconnect*
OSSD	*OpenSource Software Development*
PCE	*Process Centred Environments*
PML	*Process Modeling Language*
PSEE	*Process-centered Software Engineering Environment*
PSS	*Process Support System*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
SCI	*Software Configuration Item*
SCM	*Software Configuration Manager*
SEE	*Software Engineering Environment*
SGBDOO	Sistema de Gerenciamento de banco de dados Orientado a objetos
SOAP	*Simple Object Access Protocol*
TCP	*Transmission Control Protocol*
UML	*Unified Modeling Language*
URL	*Universal Resource Locator*
VCS	*Version Control System*
WAN	*Wide Area Network*
WSDL	*Web Service Definition Language*
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO....17

1.1 CONTEXTO....18
1.2 MOTIVAÇÃO....19
1.3 OBJETIVOS....20
1.3.1 OBJETIVO GERAL....20
1.3.2 OBJETIVOS ESPECÍFICOS....20
1.4 METODOLOGIA....20
1.5 ESTRUTURA DO TRABALHO....22

2 CONCEITOS ENVOLVIDOS E TRABALHOS RELACIONADOS....23

2.1 CONCEITOS ENVOLVIDOS....23
2.1.1 DESENVOLVIMENTO GLOBAL DE *SOFTWARE*....23
2.1.2 AMBIENTE DE DESENVOLVIMENTO DE *SOFTWARE*....26
2.1.3 SISTEMAS DISTRIBUÍDOS....34
2.1.4 COMPONENTES....37
2.1.5 FRAMEWORK....39
2.1.6 ARQUITETURA EM CAMADAS....41
2.2 TRABALHOS RELACIONADOS....42
2.2.1 ADELE/TEMPO....42
2.2.2 CAGIS....43
2.2.3 ENDEAVORS....45
2.2.4 EPOS....46
2.2.5 EXPSEE....48
2.2.6 GENESIS....49
2.2.7 MARVEL....52
2.2.8 MILOS....54
2.2.9 ODYSSEY....55
2.2.10 OIKOS....56
2.2.11 PROSOFT....57
2.2.12 PROSYT....58
2.2.13 COMPARAÇÃO ENTRE OS AMBIENTES ESTUDADOS....59
2.2.14 REQUISITOS DESEJÁVEIS DE ADDS....62
2.3 CONCLUSÃO....64

3 O *FRAMEWORK* FRADE....65

3.1 VISÃO GERAL....65
3.2 CONTEXTO....68
3.3 CAMADA DE APLICAÇÃO....69
3.4 CAMADA DE NEGÓCIOS....72
3.4.1 GERENCIADOR DE PROCESSOS....77
3.4.2 GERENCIADOR DE PROJETOS....80
3.4.3 GERENCIADOR DE ARTEFATOS....83

3.4.4 GERENCIADOR DE RECURSOS 86
3.4.5 GERENCIADOR DE LOCAIS 88
3.4.6 GERENCIADOR DE USUÁRIOS 91
3.4.7 GERENCIADOR DE FERRAMENTAS 95
3.5 CAMADA DE INFRA-ESTRUTURA 99
3.5.1 SUPORTE À PERSISTÊNCIA 104
3.5.2 SERVIÇO DE PERSISTÊNCIA 105
3.5.3 SUPORTE A ARTEFATOS 108
3.5.4 SERVIÇO DE ARTEFATO 110
3.5.5 SUPORTE À ATUALIZAÇÃO 114
3.5.6 SERVIÇO DE ATUALIZAÇÃO 117
3.5.7 SUPORTE A ACESSO 118
3.5.8 SERVIÇO DE ACESSO 121
3.5.9 SUPORTE À COMUNICAÇÃO 123
3.5.10 SERVIÇO DE COMUNICAÇÃO 124
3.5.11 SUPORTE À DISTRIBUIÇÃO 126
3.5.12 SERVIÇO DE DISTRIBUIÇÃO 127
3.6 CAMADA DE COMUNICAÇÃO 128
3.7 CONCLUSÃO 134

4 AVALIAÇÃO DO *FRAMEWORK* 138

4.1 IMPLEMENTAÇÃO 139
4.2 EXTENSIBILIDADE 141
4.3 INSTANCIAÇÃO DO *FRAMEWORK* 142
4.4 EXPERIMENTOS REALIZADOS 147
4.5 AVALIAÇÃO 149
4.6 CONTRIBUIÇÃO 150
4.7 CONCLUSÃO 151

5 CONCLUSÕES 152

5.1 LIMITAÇÕES 153
5.2 TRABALHOS FUTUROS 153

REFERÊNCIAS 155

ANEXO I – OUTROS ADSS ENCONTRADOS NA LITERATURA 171

1 Introdução

Atualmente, as organizações estão se movendo mais efetivamente para explorarem sistemas de computação e, isto está potencializando um crescimento para uma nova classe de aplicações em larga escala. Estes sistemas de computação devem oferecer uma alta integração para usuários que estão fisicamente distribuídos e interagindo com uma multiplicidade de dispositivos computacionais (COULOURIS et al., 2000). Segundo SOMMERVILLE (2000), virtualmente, todos os grandes sistemas baseados em computadores são agora sistemas distribuídos.

Esta nova realidade estimulou a adoção de uma nova estratégia em desenvolvimento de *software* que se denomina desenvolvimento global de *software*. Nela as equipes se encontram em locais geograficamente distribuídos e interagem no desenvolvimento de um projeto em comum.

Equipes de desenvolvimento global de *software* necessitam de ferramentas para automatizar o processo de desenvolvimento de *software*. A automatização do processo de desenvolvimento de sofware e o estabelecimento de uma infra-estrutura adequada são atividades importantes durante o ciclo de vida de um projeto. Estas atividades incluem a seleção de ferramentas de forma a aumentar e manter a eficiência (ROYCE, 1998).

Dentre as características que os sistemas devem possuir, a flexibilidade recebe destaque. Sistemas devem ser flexíveis pois as decisões de projeto que parecem razoáveis podem depois se tornar inviáveis (TANENBAUM, 1994). Ainda, esta flexibilidade deve ser refletida não apenas no produto de *software* gerado por empresas de desenvolvimento mas também nas ferramentas utilizadas e na sua forma de desenvolver.

Dentre as ferramentas existentes para desenvolvimento de *software*, os Ambientes de desenvolvimento de *software* (ADS) se tornam peça fundamental para o controle de processos de desenvolvimento de sistemas. Uma definição estendida de ADS se encontra na sub-seção 2.1.2 deste trabalho.

Segundo PASCUTTI (2002) o DiSEN (*Distributed Software Engineering Environment*) é um ambiente que propõe o gerenciamento distribuído de desenvolvimento de *software* baseado em agentes. O ambiente possui arquitetura constituída por gerenciadores (objetos, agentes e *workspace*), suportes (persistência, nomeação e concorrência) e um supervisor de configuração dinâmica para prover infra-estrutura necessária na realização do desenvolvimento distribuído por equipes geograficamente dispersas.

Diante das propriedades desejáveis de flexibilidade em ferramentas de *software* e na automatização do desenvolvimento de *software*, a idéia de definir e usar componentes e *framework*s tem recebido grande atenção na comunidade de desenvolvimento de sistemas.

Esta motivação ocorre pela própria definição de componente como sendo uma unidade de *software* testada para fins específicos que seja útil, adaptável, portável e reutilizável. O desenvolvimento de *software* baseado em componentes (DBC) tem por objetivo permitir que os desenvolvedores usem mais de uma vez o código escrito em qualquer linguagem (PETER, PEDRYCZ, 2001). Cada componente pode ser encarado como uma entidade de execução independente. Além disto, a utilização de componentes aumenta a confiabilidade, reduz riscos no processo, permite uma utilização mais efetiva de especialistas e acelera o desenvolvimento de *software* (SOMMERVILLE, 2000).

Com DBC espera-se possibilitar uma maior reutilização dos componentes definidos, facilitar as modificações de maneira rápida e efetiva além de gerenciar a complexidade do desenvolvimento (GIMENES, HUZITA, 2005).

Assim, este paradigma se adapta às necessidades do DiSEN por não obrigar seus participantes a utilizarem uma única implementação de um determinado gerenciador mas permitir que haja a possibilidade de utilizar outras implementações sem contudo perder a integridade do ambiente como um todo.

Entre os fatores críticos listados por PRIKLADNICKI (2003), a infra-estrutura é uma aspecto crucial para o sucesso do desenvolvimento distribuído de software.

É diante deste cenário que este trabalho propôs construir um *framework* para ambientes distribuídos de desenvolvimento de *software*. Este *framework* poderá ser instanciado nas diversas estações que comporão o ambiente. Particularmente, o ambiente DiSEN está sendo considerado como estudo de caso. Para tanto, os suportes para a infra-estrutura e os gerenciadores do DiSEN apresentados em PASCUTTI (2002) , serão encarados como componentes de *software* individuais a serem integrados ao ambiente através do *framework* proposto no presente trabalho.

1.1 Contexto

O DiSEN é um ambiente distribuído de desenvolvimento de *software* desenvolvido por um grupo de pesquisa homônimo da Universidade Estadual de Maringá (HUZITA, et al, 2004). Os trabalhos relativos ao DiSEN e desenvolvidos até o momento (PASCUTTI, 2002), (BATISTA, 2003), (PEDRAS, 2003), (MORO, 2003), (LIMA, 2004), (POZZA, 2005), (ENAMI, 2006) e (WIESE, 2006) tratam de alguns gerenciadores e dos seus respectivos papéis dentro do ambiente. Dentre estes trabalhos, há alguns que se constituem em aplicações para o DiSEN (PEDRAS, 2003), (BATISTA, 2003), (MORO, 2003) e (ENAMI, 2006), enquanto que outros em definições de gerenciadores e de um *framework* que engloba os *workspaces* do ambiente (POZZA, 2005), em um mecanismo para apoio da tomada de decisões para a seleção de recursos humanos (LIMA, 2004) e em um modelo para interoperabildiade para ADS (WIESE, 2006). Contudo, até o momento, não houve uma preocupação explicita em definir mecanismos de infra-estrutura que ofereçam suporte à flexibilidade e a configuração dinâmica do ambiente.

É neste contexto que este trabalho se encaixa no mesmo grupo de pesquisa. Mais que propor um ambiente rígido, há a necessidade de uma adequação do ambiente conforme a necessidade de cada participante. Portanto, justifica-se a definição de um *framework* que disponibilize uma infra-estrutura, adequada, do DiSEN. Este *framework* permitiu que o ambiente seja configurado dinamicamente e que atenda aos vários participantes em várias fases de desenvolvimento dos diversos projetos.

A proposta inicial de arquitetura para o ambiente DiSEN (PASCUTTI, 2002) descreve vários gerenciadores ligados por um canal de comunicação. Além disto, a arquitetura inicial previa que o ambiente ofereceria suporte à persistência, nomeação e concorrência. A comunicação entre os gerenciadores e os suportes necessários no ambiente seria através de um canal de comunicação (*middleware* e *middleware-agent*).

Certo de que o canal de comunicação ainda não tinha sido escolhido e que o mesmo poderia não contar com o suporte a persistência e tarefa de nomeação, PEDRAS (2003) propõe que haja serviços de integração e configuração do DiSEN e também funções de controle de acesso de maneira a atender as necessidades de instalar, atualizar e remover ferramentas no ambiente de forma dinâmica. PEDRAS (2003) ainda separa o canal de

comunicação da camada de *Middleware* deixando os suportes que o ambiente deverá possuir independente do *middleware* escolhido.

As possibilidades de conexão entre os diversos gerenciadores do DiSEN propostas por MORO (2003) sugerem que o canal de comunicação seja um *Middleware* sem, no entanto, definir qual a tecnologia de comunicação seria utilizada para a integração deste *middleware*.

O trabalho de MORO (2003) também traz uma especificação de que o ambiente deverá possuir suporte à persistência, nomeação e transação conforme especificados em trabalhos anteriores.

As funções de controle, acesso e serviços de gerenciamento de configuração propostas por PASCUTTI (2002) se transformam em gerenciadores na forma de um gerenciador de versão e configuração no trabalho de MORO (2003) e um gerenciador de acesso de maneira a integrá-los aos gerenciadores do DiSEN.

Com base nos trabalhos aqui apresentados publicados pelo nosso grupo de pesquisa, a presente proposta de dissertação apresenta um ajuste do modelo original de arquitetura identificando os gerenciadores que constituirão os componentes do *framework* para a infra-estrutura.

O DiSEN é o ADDS aonde este *framework* será primeiramente aplicado e validado.

1.2 Motivação

A definição de um *framework* implica em desenvolver seus componentes, definir as interfaces e as suas funcionalidades. O desenvolvimento baseado em *frameworks* oferece uma flexibilidade que atende as questões relacionadas à transparência, escalabilidade e compartilhamento de recursos desejáveis em ambientes distribuídos de desenvolvimento de *software*.

O desenvolvimento de qualquer sistema a partir de um *framework* propõe que haja pontos de variabilidade e pontos não variáveis permitindo assim criar certos contratos (classes abstratas) entre componentes. Estes pontos de variabilidade possibilitarão as alterações necessárias de alguns componentes para melhor configuração do ambiente em cada estação de trabalho. Os contratos ou pontos não variáveis garantirão a integridade do ambiente. Estes pontos de variabilidade são chamados de *hot spots* do *framework* e os pontos não variáveis são chamados de *frozen spots*.

O *framework* apresentado neste trabalho traz uma divisão de seus componentes em camadas. As camadas do *framework* possuem distintos níveis lógicos de abstração. A separação de um ambiente em níveis lógicos permite isolar cada funcionalidade/característica do ambiente e tratar cada uma delas de maneira independente.

Desta maneira, além de propor um *framework* de aplicação, é objetivo deste trabalho definir as funcionalidades de um ADS, separar seus níveis de abstração, propor a separação das camadas do ambiente e definir a interface dos componentes de cada camada.

Mais que atender as necessidades de ambiente de Desenvolvimento de *Software*, este *framework* se propõe a atender necessidades de um sistema distribuído. Os mecanismos de distribuição devem permitir a transparência do ambiente tanto para a localização de serviços quanto de usuários. É através da distribuição que este trabalho propõe facilidades de colaboração / cooperação entre desenvolvedores em um ADDS.

1.3 Objetivos

Os objetivos deste trabalho se dividem em objetivo geral e objetivos específicos.

1.3.1 Objetivo Geral

O objetivo deste trabalho é compor um *framework* para infra-estrutura do DiSEN que possibilite:

- Prover suporte às atividades de desenvolvimento distribuído de *software* como o controle de versões de artefatos, gerenciamento da informação dos processos e gerenciamento de projeto;
- Prover suporte para atividades cooperativas;
- Prover suporte para a configuração dinâmica de um ambiente de desenvolvimento de *software*.

1.3.2 Objetivos Específicos

- Aprofundar o entendimento sobre ADSs e, especificamente, o DiSEN;
- Estudar os gerenciadores de ADDSs, seus requisitos e a necessidade de comunicação entre os mesmos;
- Estudar padrões de projetos, *Frameworks* e componentes para encontrar os possíveis padrões de projeto que serão utilizados neste trabalho;
- Estudar comunicação distribuída para melhor definir o canal de comunicação do ambiente.

1.4 Metodologia

Para o desenvolvimento deste trabalho foi feito um levantamento bibliográfico e um estudo dos trabalhos anteriormente desenvolvidos sobre o ambiente DiSEN. O ambiente DiSEN é o estudo de caso deste trabalho por ser desenvolvido por nosso grupo de pesquisa. A partir dos gerenciadores e suportes para infra-estrutura propostos em (PASCUTTI, 2002) foi feito um levantamento e um mapeamento de cada gerenciador, suporte ou supervisor para os componentes de *software* do *framework*. Feito o levantamento dos gerenciadores e de sua distribuição entre as camadas do ambiente foram analisadas as diferentes funcionalidades de cada um dos gerenciadores e também o detalhamento de suas necessidades em relação à infra-estrutura.

Foi realizada também uma revisão bibliográfica de outros ADS e o mapeamento dos requisitos comuns a estes ambientes. A partir deste ponto houve uma transposição das necessidades recorrentes do domínio de ADDSs para a arquitetura em camadas do FRADE.

Na divisão das funcionalidades entre as camadas do *framework* houve a preocupação em definir de maneira explícita as responsabilidades de cada camada.

Os requisitos mapeados para o domínio de Ambientes de Desenvolvimento de *software* foram agrupados de maneira a criar um domínio de aplicação. Partindo do domínio modelado e de suas classes de negócios foi realizado um mapeamento do que seria necessário criar para que a integração dos requisitos funcionais do modelo da aplicação pudesse ser realizada com os requisitos não-funcionais como segurança, persistência, confiabilidade, distribuição e colaboração.

Uma vez mapeado um requisito, houve a necessidade de decidir quem seria responsável por fornecê-lo ao ambiente, aonde o mesmo se encontraria na divisão de camadas e como este requisito seria integrado ao *framework*.

A partir desta divisão de camadas e responsabilidades partiu-se para a definição das interfaces das classes abstratas para a comunicação e troca de mensagens entre os componentes de infra-estrutura que foram mapeados. A definição dos métodos das interfaces para cada requisito em cada uma das camadas e a comunicação entre as diversas camadas foi feita com o objetivo de definir os componentes do *framework*.

A partir da definição das interfaces partiu-se para a implementação de alguns componentes para a validação do *framework* e, por conseguinte, para uma construção baseada em padrões de projetos de maneira a simplificar a comunicação entre os componentes.

A Figura 1 ilustra a metodologia utilizada para o desenvolvimento deste trabalho:

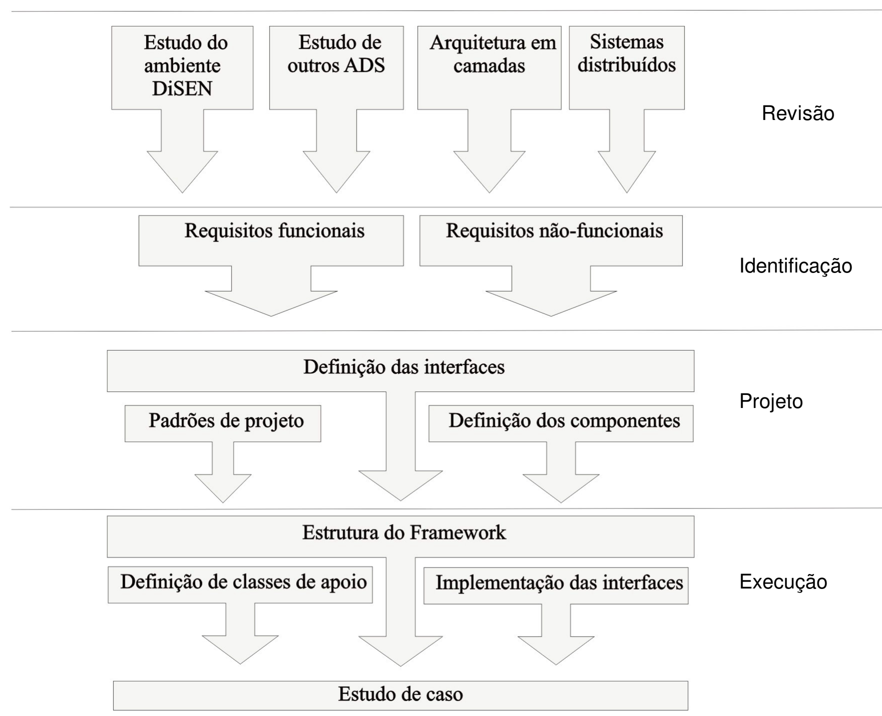

Figura 1 – Metodologia do trabalho

A prototipação do *framework* iniciou-se pelo canal de comunicação e suportes definidos como necessários para oferecer transparência no ambiente. Alguns gerenciadores como: gerenciador de ferramentas, gerenciador de locais e gerenciador de usuários foram identificados e atuam na camada de negócios provendo mecanismos para a transparência na distribuição.

1.5 Estrutura do trabalho

Este trabalho está estruturado da seguinte forma:

No capítulo 2 serão apresentados os conceitos que fundamentam esta dissertação e os trabalhos relacionados a ambientes de desenvolvimento de *software*.

No capítulo 3, adentrando propriamente no trabalho desenvolvido nesta dissertação, serão definidas as camadas que compõem este *framework*, bem como, o seu conteúdo, as responsabilidades de cada uma das camadas, seus componentes e gerenciadores. Da camada de infra-estrutura serão apresentados os serviços e suportes que comporão o *framework*. Da camada de aplicação, os componentes de apoio ao desenvolvimento de aplicações e da camada de comunicação apresentar-se-á a estrutura do canal de comunicação do *framework*.

O capítulo 4 versará sobre a avaliação do *framework*.

Por fim, no capítulo 5 serão desenvolvidos a conclusão e as coordenadas para trabalhos futuros.

2 Conceitos Envolvidos e Trabalhos relacionados

Neste capítulo, serão apresentados os principais conceitos correlacionados a este trabalho. Referidos conceitos são: desenvolvimento global de *software*, ambientes de desenvolvimento de *software*, sistemas distribuídos, componentes, *frameworks* e arquitetura em camadas. Da mesma maneira, será apresentado o estudo de trabalhos relacionados conforme proposto na metodologia deste trabalho.

2.1 Conceitos envolvidos

Para o desenvolvimento desta dissertação, alguns conceitos necessários para o seu embasamento serão explicados a seguir.

2.1.1 Desenvolvimento global de *software*

Os projetos de desenvolvimento de *software* contemporâneos têm, progressivamente, aumentado de tamanho e complexidade, sendo cada vez mais comum sua realização por equipes de médio porte (entre dez e vinte desenvolvedores) e grande porte (acima de vinte desenvolvedores). As facilidades de comunicação proporcionadas pela Internet, a necessidade de experiência em diversas áreas de conhecimento e a pressão por cronogramas mais restritos, possibilitaram que alguns projetos sejam realizados por diferentes equipes trabalhando concorrentemente. Ainda mais, as dificuldades de reunir os especialistas necessários em um mesmo local físico e a delegação do desenvolvimento de determinados componentes para outras empresas, são exemplos de fatores que podem exigir que as equipes participantes de um projeto estejam geograficamente distribuídas (TEIXEIRA et al., 2001-a) (TEIXEIRA et al., 2001-b).

Ao longo dos anos o *software* se tornou um componente vital da maioria dos negócios. O crescimento do sucesso depende da utilização do *software* como uma arma competitiva. Na década de 80, muitas organizações começaram a experimentar o desenvolvimento de *software* remoto como uma facilidade a mais (HERBSLEB, MOITRA, 2001).

A crescente globalização do ambiente de negócios e da economia têm afetado, diretamente, o mercado de desenvolvimento de *software*. Os engenheiros de *software* vêm reconhecendo, há algum tempo, a profunda influência do desenvolvimento global de *software* na globalização dos negócios e, através de reações alarmistas estão se movimentando para encontrar um modelo de negócio que possa atender a este mercado. Recentemente, a atenção está se voltando para o entendimento dos fatores, que permitem as multinacionais e as corporações virtuais a operar com sucesso ultrapassando as fronteiras geográficas, e, culturais em busca de vantagens competitivas como baixos custos, produtividade ou qualidade na área de desenvolvimento de sistemas (HERBSLEB, MOITRA, 2001) (FREITAS, MAIA, NUNES, 2004).

As novas formas de competição e cooperação, que vão além das fronteiras dos países, estão provocando um impacto profundo não apenas na comercialização e distribuição, mas também, na maneira em que os produtos são criados, desenvolvidos, testados e entregues aos consumidores (HERBSLEB, MOITRA, 2001).

Este fenômeno é alimentado por fatores tais como o acesso a uma grande quantidade de mão de obra especializada, redução nos custos do desenvolvimento, presença global e proximidade ao consumidor. Apesar do sucesso de várias equipes globais, as pesquisas revelam que a distância contribui para aumentar a complexidade nos processos organizacionais. Primeiramente, os processos de comunicação e coordenação são afetados pela distância, com conseqüências diretas na definição, construção, testes e entrega do *software* ao cliente final, assim como no gerenciamento do desenvolvimento (LANUBILE, DAMIAN, OPPENHEIMER, 2003). Os projetos de *software open source* (*OSSD - Open source software development*) são casos típicos de desenvolvimento global de *software* (FUJIEDA, OCHIMIZU, 2003).

Tanto na literatura quanto em ambientes empíricos as condições associadas ao crescimento do desenvolvimento global de *software* baseiam-se no fato de que a maioria das garantias que há em um projeto pequeno não se aplica em um projeto constituído por uma rede grande de colaboradores. Dessa maneira, as práticas que garantem o desenvolvimento de projetos com uma grande rede de colaboradoes podem ser aplicadas a qualquer projeto disperso, independente de quão disperso este projeto possa vir a ser (PYYSIAINEN, 2003).

As vantagens do desenvolvimento global são permitir aos participantes (BRAUN, DUTOIT, BRUEGGE, 2003):

- Encontrar *stakeholders* por meio da busca por esforços relacionados, discussões ou materiais;
- Buscar rapidamente artefatos por versão, *stakeholder*, tópico, localização, ou qualquer outra característica presente no espaço do conhecimento;
- Entender e aprender pelo histórico do projeto ou formar projetos por meio da busca por entradas relacionadas que incluem argumentação, alternativas e decisões.

Desta forma, o desenvolvimento global de *software* vem se tornando, rapidamente, uma tendência para as empresas de tecnologia. As pesquisas de (HERBSLEB et al., 2001) demonstram que o desenvolvimento distribuído pode melhorar o tempo no ciclo de desenvolvimento. Demonstram também que no desenvolvimento distribuído há poucas maneiras de um colega de trabalho colaborar com outro caso este esteja sobrecarregado de tarefas no projeto.

Vários fatores aceleraram a distribuição geográfica do desenvolvimento de *software* (HERBSLEB, MOITRA, 2001), entre elas:

- A necessidade de capitalizar recursos globais para obter sucesso e a competitividade que requer recursos escassos, não importando onde estes estejam alocados;
- A vantagem da proximidade do negócio com o mercado, incluindo o conhecimento dos consumidores e as condições locais;
- A rápida formação de corporações virtuais e equipes virtuais que aproveitam as oportunidades do mercado;
- A pressão constante quanto ao prazo da entrega com a equipe dispersa em diferentes fusos horários e um desenvolvimento distante;
- A necessidade por flexibilidade em capitalizar, agregar e adquirir oportunidades de negócio não importa aonde elas se encontrem.

Como resultado, o desenvolvimento do *software* está crescendo em um universo multicultural, global e distribuído. Gerentes, engenheiros e executivos encontram desafios em vários níveis, desde o técnico ao social e cultural (HERBSLEB, MOITRA, 2001).

Para a efetividade do desenvolvimento global de *software* há algumas preocupações que devem ser consideradas (BIANCHI et al., 2003):

- A decisão de como dividir as tarefas entre os locais de maneira que todos estejam aptos para trabalhar de maneira tão independente quando possível;
- A questão cultural quando a equipe é constituída por pessoas de diferentes culturas;
- A comunicação inadequada causada pelo fato da distribuição geográfica da equipe aumentar o custo com a comunicação formal;
- A dificuldade em propiciar a troca de experiência entre os membros da equipe;
- O gerenciamento do conhecimento que se torna mais complexo em um ambiente distribuído com compartilhamento da informação. A dificuldade em gerenciar este conhecimento torna mais lenta as oportunidades de reuso;
- O gerenciamento de projeto e processo que torna necessário sincronizar o trabalho de vários locais;
- As questões técnicas que têm impacto na rede de comunicação entre as pessoas.

A distância entre membros da equipe aumenta a complexidade do processo organizacional, uma vez que os processos de comunicação, coordenação e controle são afetados por ela. Por esta razão, é necessário que haja, não apenas um conjunto de ferramentas para habilitar o processo de *software,* mas uma pesquisa multidisciplinar que permita visualizar as dificuldades por vários ângulos distintos e fornecer uma visão maior quanto às dificuldades que isto pode trazer (DAMIAN, LANUBILE, OPPENHEIMER, 2003).

Outro fator preponderante que pode causar falhas na comunicação são as diferenças culturais, técnicas e sociais da equipe. Independente da localização dos membros da equipe, o gerenciamento de um projeto necessita de mecanismo para garantir a atualização do projeto, das informações do projeto[1] e também garantir o compartilhamento de informações e soluções que podem acelerar o desenvolvimento (HERBSLEB, MOITRA, 2001).

As diferenças culturais são possivelmente a característica mais confusa e interessante das equipes globais. Membros com diversas atitudes assumindo o gerenciamento e o risco de trabalhar em equipes conjuntas leva o projeto a rumos distintos e até inesperados tornando o seu gerenciamento um desafio (LANUBILE, DAMIAN, OPPENHEIMER, 2003).

O trabalho à distância possui a dificuldade inerente de comunicação. Porém, o trabalho distribuído geograficamente possui vantagens pelo mesmo motivo. Equipes distribuídas podem aproveitar fusos horários e uma empresa distribuída pode possuir funcionários trabalhando no mesmo projeto 24 horas por dia utilizando as variações do relógio global. Com isto, um dos grandes problemas associado ao desenvolvimento de sistemas, o prazo de entrega, encontra um forte aliado sem cansar a equipe ou necessitar de trabalhadores trocando de turnos (HERBSLEB, MOITRA, 2001).

[1] As atualizações do projeto refletem nos artefatos gerados em um processo de desenvolvimento de *software* como diagramas e códigos fonte. As informações do projeto significam as informações geradas pelo gerenciamentodo projeto como atas de reuniões, documentos de conclusão de etapas do processo e cronogramas.

Um exemplo disso é relatado por KIEL (2003) em um projeto que possuía uma equipe na Alemanha e outra no Canadá. Com 8 horas de diferença em seus fusos horários o projeto alcançava um período de 16 horas ininterruptas de desenvolvimento. Neste projeto, apesar da vantagem temporal, a separação temporal, diferença de línguas e diferenças culturais acabaram por contribuir para o desafio de criar uma atmosfera de confiança, respeito e cooperação que caracterizam um projeto de *software* coeso.

Em contrapartida, tem-se notado que a distância, que pode causar problemas de comunicação, pode ser representada pelos vários andares de um prédio ou mesmo pessoas na mesma sala com divisórias. As soluções de comunicação adotadas por uma equipe de desenvolvimento podem ser as mesmas para a distribuição entre os andares de um prédio e para uma distância que atravessa os limites de um país (HERBSLEB, MOITRA, 2001).

Outro problema que pode ser causado pelo desenvolvimento geograficamente distribuído é a falha na sincronização do projeto. A sincronização requer que os marcos de tempo e de tarefas de um projeto sejam claros, e possuam entradas e saídas bem definidas em seu fluxo de trabalho. Este problema não ocorre apenas em equipes distribuídas mas em várias equipes de desenvolvimento que não possuem clareza em seu processo de desenvolvimento (HERBSLEB, MOITRA, 2001).

Estes são apenas alguns dos fatores que tornam um desafio gerenciar projetos de *software* desenvolvidos em estruturas geograficamente distribuídas. Há uma necessidade de ferramentas e técnicas que não apenas auxiliem o processo de desenvolvimento mas, também, orientem as características sociais e organizacionais no desenvolvimento global (LANUBILE, DAMIAN, OPPENHEIMER, 2003).

Ferramentas e ambientes foram desenvolvidos nos últimos anos, como será tratado na próxima seção, para ajudar no controle e coordenação de times de desenvolvimento trabalhando em ambientes distribuídos geograficamente. Muitas destas ferramentas são focadas nos procedimentos de suporte para comunicação formal como elaboração automatizada de documentos, processos e outros canais de comunicação não interativos (PRIKLADNICKI, AUDY, 2003). Exemplos destas ferramentas são os ambientes de desenvolvimento de *software*.

2.1.2 Ambiente de desenvolvimento de *software*

Desde o advento dos ambientes de apoio a programação, como a *UNIX Programmer´s Workbench* na década de 70, há uma tendência, na indústria de *software,* em desenvolver sistemas para dar suporte às suas atividades através do ciclo de vida de seus produtos, que vão do desenvolvimento à manutenção (BOLDYREFF et al., 2003) (TAYLOR et al., 1988).

Os primeiros ambientes de desenvolvimento de *software* (ADS) começaram como ambientes de programação: coleções de ferramentas desenvolvidas para acelerar o desenvolvimento de código para um sistema (POPOVICH, 1992). Estes ambientes de *software* eram um pouco mais inteligentes do que editores de texto e, tinham como objetivo, dar suporte apenas à programação (TAYLOR et al., 1988). Estes ambientes contavam com ferramentas como compiladores, montadores e depuradores (LIMA REIS, 1998).

Na década de 80 surgiu a necessidade de ferramentas de apoio a outras fases do desenvolvimento de *software*, como análise e projeto, visando a integração de projetos de *software*. Estas ferramentas têm como característica principal o suporte a determinados

métodos de desenvolvimento e ficaram conhecidas como ferramentas CASE (LIMA REIS, 1998) (BOLDYREFF et al., 2003).

Devido à necessidade de outras ferramentas de apoio no desenvolvimento de sistemas, as ferramentas CASE evoluíram para CASE integrados (I-CASE - Integrated CASE), os quais são compostos de várias ferramentas capazes de transferir informações para outras ferramentas (LIMA REIS, 1998).

Com a evolução do desenvolvimento de sistemas, a engenharia de *software* vem demonstrando maior interesse na área de tecnologia de processos de *software*. Mais que criar a definição de processos de *software*, esta área de pesquisa inclui a construção de ferramentas e ambientes para a modelagem, evolução dos processos de desenvolvimento de *software*, simulação e execução (DERNIAME, KABA, WASTELL, 1999).

A evolução das ferramentas CASE em ADS se deu pela necessidade de ferramentas que forneçam um apoio mais amplo e efetivo aos desenvolvedores de *software*, de forma que metas como aumento da produtividade, melhoria da qualidade, diminuição de custos e diminuição do tempo para introdução no mercado possam ser alcançadas (VILLELA et al., 2003).

Um ADS deve proporcionar alto nível de integração entre suas ferramentas e ser capaz de executar um modelo de processo de *software* por meio da coordenação dos desenvolvedores na execução de suas tarefas, gerência de alocação de recursos, coleta de métricas, execução automática de algumas atividades e adequações do processo durante sua execução. O objetivo principal de um ADS é prover suporte ao desenvolvimento de *software* cooperativo e automatizar partes do processo que são possíveis de mecanização (CUGOLA, GHEZZI, 1998).

Há vários tipos de ADS atualmente em pesquisa como: centrados em processos, orientados a domínio, orientados a organização, distribuídos e federações de ADS.

Ambientes de Desenvolvimento de *Software* Centrados em Processos

Desde o começo da década de 90 há um aumento na preocupação pela qualidade na maioria dos setores industriais. Por esta razão, a importância de processos de produção também tem crescido. Os processos são importantes pois há um relacionamento claro entre a qualidade do produto e a qualidade do processo utilizado para criar este produto. A uniformidade de ambiente para diferentes projetos pode otimizar a produtividade de diferentes produtos e, também, melhorar o tempo para o mercado e diminuir o custo de produção (CUGOLA, GHEZZI, 1998).

A importância do processo de *software* está diretamente ligada à idéia comumente aceita de que a qualidade do produto é resultado da qualidade do processo (BARTHELMESS, 2003). A idéia é que, por meio da melhoria do processo é possível melhorar a qualidade do produto final produzido (CUGOLA, GHEZZI, 1998).

Os ambientes de engenharia de *software* centrados em processo (*PSEEs -Process-centered Software Engineering Environments*) formaram uma geração de ambientes de suporte às atividades de desenvolvimento de *software*. Eles exploram a representação explícita de processos que especificam como controlar as atividades de desenvolvimento de *software*, os papéis e atividades dos desenvolvedores e como utilizar e controlar as ferramentas de desenvolvimento (AMBRIOLA, CONRADI, FUGGETTA, 1997) (FREITAS,

MAIA, NUNES, 2004) (REIS, LIMA REIS, NUNES, 2001). Nestes ambientes, um modelo de processo é interpretado por um mecanismo de execução, o qual interage com os desenvolvedores e com as ferramentas do ambiente a fim de controlar o processo de *software* (LIMA REIS, 1998).

Processos de *software* envolvem o desenvolvimento de atividades complexas por pessoas diferentes com perfis técnicos distintos. O controle automático da execução das atividades é possível pela utilização de um modelo de processo. Um modelo de processo é uma descrição formal de um processo de *software*. Muitos tipos de informações devem ser integradas neste modelo de maneira a identificar quem, quando, como e porque os passos tomados durante o desenvolvimento de *software* serão tomados (REIS, LIMA REIS, NUNES, 2001).

A notação formal de um modelo de processo é feita por meio de uma linguagem de modelagem do processo. Antes da execução, o modelo é instanciado através da definição do cronograma, dos desenvolvedores que atuarão no processo e dos recursos a serem alocados. Falhas na modelagem, inconsistências nos prazos e na alocação de recursos somente serão detectados quando o processo estiver em andamento. Desta forma, a execução de processos de *software* pode levar a altos custos em decorrência da falta de uma prévia validação do modelo (REIS et al., 1999).

Os Ambientes de Desenvolvimento de Software (PSEE – *Process-centered Softtware Engineering Environment*) se estabelecem no contexto de suporte à definição rigorosa de processos fornecendo serviços para analisar, simular, habilitar, executar e acompanhar um projeto de desenvolvimento e, ainda, reutilizar definições de processos em outros processos (REIS, LIMA REIS, NUNES, 2001).

O acompanhamento permite o controle da execução das atividades, considerando a hierarquia e o estado destas (ARAÚJO, 1998) (MIAN, NATALI, FALBO, 2001). Uma mesma atividade em diferentes estados pode utilizar diferentes ferramentas. Desta maneira o processo é utilizado como um conjunto de regras que governam também a utilização destas ferramentas (POPOVICH, 1992)

A utilização de processos pode trazer benefícios como (FREITAS, MAIA, NUNES, 2004) (WILLIAMS, 1988) (AVERSANO et al., 2004):

- Melhor comunicação entre as pessoas envolvidas no desenvolvimento de *software*;
- Automatização para determinadas tarefas;
- Controle do fluxo de informação;
- Disponibilidade de informações sobre o andamento do processo;
- Possibilidade de reutilização de processo de *software*; e coleta automática de métricas;
- Rastreabilidade de determinadas características no gerenciamento de processos de negócios que contribuem para o aumento do desempenho e da qualidade do serviço;
- Padronização dos procedimentos.

As características que distinguem ADS orientados a processo (e produtos de workflow) de outros ADS são a focalização explícita em mecanismos de processo, que resulta na necessidade de definição do processo e linguagens de execução, além da sua ênfase em

comunicação e integração de pessoas e suas ações, ao invés da comunicação e integração de ferramentas. A diferença de um ADS em relação ao ADS orientado a processo é sutil e depende do grau de suporte ao processo que o ambiente provê (LIMA REIS, 1998).

O uso de ambientes orientados a processos força a definição rigorosa da execução do processo. O treinamento de novos usuários é facilitado pois um desenvolvedor novato pode estudar o processo definido e ter uma visão de como é o funcionamento da organização (LIMA REIS, REIS, NUNES, 1998). (LIMA REIS, 1998).

Ambientes de Desenvolvimento de *Software* Orientados a Domínios

O desenvolvimento de *software* em um domínio em que os desenvolvedores não têm experiência alguma não é uma tarefa fácil. Além da própria complexidade da atividade de desenvolvimento de um sistema, os desenvolvedores têm que lidar com a necessidade de conhecer os conceitos do domínio para facilitar a interação com seus usuários e permitir a construção do sistema (OLIVEIRA et al., 2000).

Neste contexto surgem os Ambientes de Desenvolvimento de *Software* Orientados a Domínios (ADSOD), que podem ser vistos como uma evolução dos PSEEs. Os ADSODs se propõem a prover apoio para o desenvolvimento, reparo e melhorias em *software* e para o gerenciamento e controle das atividades de desenvolvimento (BROWN, 1990).

A idéia de um ADSOD é a mesma aplicada a ADSs: automatizar várias tarefas do processo de desenvolvimento de *software* e tornar fácil o seu controle fornecendo apoio aos engenheiros de *software* em domínios específicos por meio do uso do conhecimento deste domínio durante todo o processo de desenvolvimento. Para permitir o uso do conhecimento durante o desenvolvimento de *software*, um ADSOD possui os seguintes requisitos mínimos (OLIVEIRA et al., 2000) (DIAS, PIETROBOM, ALVES, 2004):

1. Apoiar a investigação do domínio durante o desenvolvimento através de algum nível de formalização,
2. Permitir ao desenvolvedor trabalhar diretamente com o domínio do problema,
3. Apoiar a identificação, acesso e uso de objetos do conhecimento do domínio,
4. Apoiar o trabalho colaborativo entre usuários do domínio (ou especialistas do domínio) e desenvolvedores dado que, com o conhecimento do domínio integrado no ambiente, o trabalho com os usuários pode ser mais fácil por utilizar um vocabulário comum e particular do domínio em questão

Ambientes de desenvolvimento de *software* Orientados a Organização

Em Organizações de *Software*, a disponibilidade de conhecimento sobre os clientes e sobre os seus respectivos domínios representa uma vantagem estratégica na competição por novos projetos. Conhecimento sobre a organização cliente e sobre o seu domínio de negócio é fundamental nas atividades iniciais de um projeto de *software*. Estas atividades também exigem uma visão global da organização cliente e experiência de desenvolvimento de *software* no seu domínio de negócio (VILLELA, ROCHA, TRAVASSOS, 2004).

Diante desta realidade é apresentado o conceito de Ambientes de Desenvolvimento de *Software* Orientados a Organização (ADSOrg), que estende o conceito de ADSOD, uma vez que conhecimento sobre o domínio de aplicação é apenas um dos tipos de conhecimento requeridos no desenvolvimento e na manutenção de *software* (VILLELA, ROCHA, TRAVASSOS, 2004).

A definição de ADSOrg é a de ser ADS centrados em processo que apóiam a Gerência do Conhecimento ao longo dos processos de desenvolvimento e manutenção de *software*. Os objetivos de tais ambientes são (VILLELA, ROCHA, TRAVASSOS, 2004):

- Apoiar os desenvolvedores de *software* na execução de suas atividades, fornecendo todo o conhecimento que tenha sido capturado e acumulado pela organização por sua importância para o desenvolvimento e a manutenção de *software*;
- Apoiar o aprendizado organizacional a partir do aprendizado dos desenvolvedores nos projetos de *software*.

Estendendo o conceito de ADS, os ADSOrg também possuem definições de processos. Diferentes processos de *software* podem ser especializados de acordo com as características de cada organização, podendo ser incluídas novas atividades, porém mantendo os elementos básicos definidos no Processo Padrão. De forma semelhante, pode haver a necessidade de incluir atividades específicas para o tipo de *software* e/ou de especializar atividades já previstas (VILLELA et al., 2004).

Durante a etapa de especialização são produzidos os processos especializados. Estes processos são importantes para apoiar o desenvolvimento e a manutenção de *software* na organização, refletindo o conhecimento e a experiência obtida sobre o uso do paradigma e dos métodos. As atividades do processo padrão são detalhadas de acordo com o paradigma e os métodos de desenvolvimento escolhidos de acordo com as características do desenvolvimento de *software* na organização, tais como experiência dos desenvolvedores e tipos de *software* desenvolvidos (VILLELA et al., 2004).

Ambientes Distribuídos de Desenvolvimento de *Software*

A crescente complexidade do *software* desenvolvido atualmente teve como conseqüência o aumento do número de profissionais envolvidos no processo de *software*. Devido à globalização da economia e ao rápido crescimento da Internet, cada vez mais processos de *software* se expandem por múltiplas organizações (FREITAS, MAIA, NUNES, 2004-b).

A distribuição geográfica e interorganizacional dos projetos criaram a necessidade de ferramentas e técnicas para a coordenação e cooperação de equipes nas tarefas que são realizadas cooperativamente (AVERSANO et al., 2004) (LIMA REIS, REIS, NUNES, 1998).

Com o intuito de suprir esta nova necessidade surgiram os Ambientes Distribuídos de Desenvolvimento de *Software* (ADDS) avançando o conceito de ADS na tentativa de simplificar a colaboração e a cooperação entre desenvolvedores. O principal objetivo de um ADDS envolve o uso concorrente de objetos de *software*, permitindo que um mesmo objeto seja manipulado por vários desenvolvedores (LIMA REIS, REIS, NUNES, 1998) (LIMA REIS, REIS, NUNES, 1998).

Três importantes fatores têm dirigido o desenvolvimento de sistemas para uma engenharia colaborativa (BOLDYREFF, DEWAR, SMITH, WEISS, NUTTER, WILCOX, RANK, 2003):

1. O crescimento da globalização na indústria de *software* pelos projetos multi-organizacionais se tornou uma realidade tanto nas indústrias de *software* como nas comunidades de desenvolvimento *open-source*;
2. O desenvolvimento baseado em Componentes permite que empresas reutilizem componentes desenvolvidos por outras empresas, o que força a colaboração entre estas;
3. O longo tempo de vida útil de grandes sistemas. Pelo desenvolvimento destes sistemas já passaram vários times de desenvolvedores. Estes sistemas necessitam de preservação de artefatos de *software* importantes, documentação e registros formais e informais o que leva as equipes a um alto nível de colaboração.

Os ambientes direcionados ao desenvolvimento distribuído de *software* possuem os requisitos comuns de um ADS como integração, gerência dos dados e processo. Entretanto, deve ser destacada a necessidade de permitir cooperação e distribuição de forma eficiente (REIS, 1998). Há diversas abordagens exploradas na literatura, para configurar ADDS explorando coordenação e cooperação (REIS, 1998):

- **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 ADDS, 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 dos ADDS atendido por meio 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.

Entretanto, nem sempre estes componentes são tratados formalmente e raras são as vezes que estão presentes no mesmo ambiente. A maioria apresenta apenas uma descrição informal do mecanismo de execução e do mecanismo que propicia a cooperação (LIMA REIS, REIS, NUNES, 1998). É necessário ainda um maior esforço dos projetos em ADDS para formalizar os componentes que ofereça suporte adequado à distribuição, cooperação, colaboração e coordenação de projetos e processos de *software*.

Federação de Ambientes de desenvolvimento de *software* centrados em processo

Segundo CUGOLA e GHEZZI (1998), as equipes de *software* são normalmente compostas de pessoas trabalhando para uma mesma organização. O aumento da complexidade do processo de desenvolvimento de *software*, todavia, requer a adoção de novos paradigmas de desenvolvimento para prover suporte à cooperação de equipes diferentes, possivelmente pertencendo a organizações distintas. Cada equipe deve estar apta a usar suas próprias ferramentas, procedimentos e dados levando em conta que estas equipes autônomas precisam colaborar para o desenvolvimento do produto.

Quando o desenvolvimento de *software* envolve organizações autônomas, é inviável utilizar um único modelo de processo para refletir todo o escopo do processo de *software* (FREITAS, MAIA, NUNES, 2004-b). Porém, a integração destas organizações para alcançar um objetivo comum leva à necessidade de meios de integrar ADS distintos em projetos comuns.

As federações de PSEEs permitem que cada equipe especifique o seu próprio modelo de processo e as interações existentes com outras equipes. Uma federação é composta de instâncias locais, que executam o modelo de processo de cada equipe, e de um componente responsável pela comunicação entre os processos locais, denominado fundação. Todos os PSEEs podem interagir acoplando atividades comuns como testes de integração de componentes de *software* desenvolvidos independentemente por diferentes organizações. A comunicação entre os processos locais é guiada pela definição das relações existentes entre os processos de cada equipe e pelas políticas interorganizacionais (FREITAS, 2005) (BASILE et al., 1996).

Integração

A integração tem sido considerada o elemento central para prover um suporte efetivo ao desenvolvimento de *software*, pois é ela quem define todas as regras e diretrizes que governarão o uso das ferramentas. A integração é responsável pela combinação de ferramentas de forma que elas trabalhem em harmonia em todas as etapas do processo (MIAN, NATALI, FALBO, 2001).

Existem vários níveis de integração de ferramentas, diferindo no suporte fornecido. Dentre eles, podem-se citar (TRAVASSOS, 1994) (MIAN, NATALI, FALBO, 2001):

- **Integração de Dados**: A chave para integrar ferramentas é a habilidade de compartilhar informação de projeto.
- **Integração de Apresentação**: A integração de apresentação tem como objetivo tornar as interfaces do ADS consistentes, permitindo que o usuário utilize as ferramentas, alternando de uma para a outra, sem sofrer um impacto na interface. Os componentes utilizados são os mesmos, com as mesmas funcionalidades, não havendo a necessidade de aprender como utilizar cada ferramenta. A partir do momento que se aprende uma, consegue-se trabalhar mais facilmente com as outras.
- **Integração de Controle**: Para integrar um conjunto de ferramentas é preciso ter um foco forte no gerenciamento do processo. Engenheiros de *software*

tornaram-se mais orientados a processos, usando métodos, técnicas e ferramentas para controlar e guiar seu trabalho.

- **Integração de Conhecimento**: Com o aumento da complexidade dos processos de *software* faz-se necessário considerar informações de natureza semântica, sem as quais a integração não é plenamente obtida. O conhecimento, assim como os dados, deve estar disponível e representado de forma única no ambiente, de forma a poder ser acessado por todas as ferramentas que dele necessitarem, evitando redundância e inconsistências.

Arquitetura de ADS

Um Ambiente de Desenvolvimento de *Software* deve proporcionar um alto nível de integração entre suas ferramentas e ser capaz de executar um modelo de processo de *software*, através da coordenação dos desenvolvedores na execução de suas tarefas, permitindo coleta de métricas, execução automática de algumas atividades e mudança do processo durante sua execução (LIMA REIS, REIS, NUNES, 1998).

O modelo de referência ECMA (*European Computer Manufacturers Association*), descreve 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, não devendo ser utilizado como uma especificação de implementação (LIMA REIS, 1998).

O modelo ECMA segue uma visão orientada a serviços abstraindo detalhes de implementação e concentrando-se nas capacidades do ambiente. Em ADS orientados a processo, a escolha do processo específico utilizado em um projeto de desenvolvimento é livre e, a sua definição e execução é apoiada por um conjunto de serviços de gerência do processo: desenvolvimento, execução, visibilidade, monitoração, transação e serviços de recursos (LIMA REIS, 1998).

A arquitetura típica de um PSEEs consiste, no mínimo, de três componentes (BARGOUTHI et al., 1996):

- Uma interface multi-usuário;
- Um componente de apoio ao processo;
- Um componente de gerenciamento de informação.

Alguns requisitos são considerados indispensáveis para o apoio ao desenvolvimento de *software*. São eles (LIMA REIS, REIS, NUNES, 1998) (LIMA REIS, 1998):

- **Suporte a múltiplos usuários**: mecanismos para atender vários usuários requisitando, ao mesmo tempo, serviços ao ambiente;
- **Gerência de objetos**: controle do acesso e da evolução de objetos compartilhados. As versões dos documentos e produtos 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 produtos que estão manipulando;
- **Gerência de Processo**: o suporte à modelagem e execução do processo de *software*. Dentro deste requisito, encontra-se a necessidade de um formalismo de modelagem de processo e uma máquina de execução das definições de processo;
- **Extensibilidade**: permitir 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**: todos os níveis de integração devem estar disponíveis para viabilizar os outros requisitos.

2.1.3 Sistemas distribuídos

Um sistema distribuído é uma coleção de máquinas autônomas que estão conectadas através de uma rede de computadores. Cada máquina executa componentes e opera sobre a camada de distribuição (EMMERICH, 2000).

Uma definição mais abrangente é que um sistema distribuído constitui-se de um conjunto de processadores autônomos, conectados através de um subsistema de comunicação, que cooperam através da trocas de mensagens (CESETTI, AKAMATU, KIRNER, 1993).

Por estarem conectados através de uma rede, um sistema distribuído pode ser visto pelo conceito da arquitetura cliente/servidor. Este conceito de arquitetura é melhor entendido na indústria de computadores. Ele tem sido utilizado no desenvolvimento de *hardware* e *software*. Ele consiste na divisão de funções em dois grupos: os que requisitam as funções (cliente) e os que realizam as funções requisitadas (servidores) (COLS, 1993).

Juntos eles formam um sistema computacional completo com uma divisão distinta de responsabilidades. A arquitetura cliente/servidor tira vantagens das redes de computação aumentando a capacidade de processamento e compartilhamento de recursos. É aceitável ainda, que estações cliente assumem o papel de servidores na rede, e que alguns servidores sejam clientes de outros servidores, tornando esta divisão mais heterogênea (LEWANDOWSKI, 1998).

O proprietário do recurso ou serviço é chamado servidor, enquanto o usuário, é chamado de cliente. Cliente e servidor podem existir fisicamente no mesmo computador como servidores de bases de dados instalados localmente (FEINSTEIN, 2000).

De uma forma geral, um servidor é uma entidade que aceita requisições de clientes, realiza seu processamento e, envia uma resposta com o resultado (LEWANDOWSKI, 1998). Servidores de arquivos, servidores de banco de dados e servidores de impressão são exemplos típicos de processos, que residiriam em processadores servidores (CESETTI, AKAMATU, KIRNER, 1993).

Os computadores clientes comandam a execução da aplicação, acessando, quando necessário, os computadores servidores, a fim de que esses realizem algumas funções específicas de servidores (CESETTI, AKAMATU, KIRNER, 1993). Também é considerada a possibilidade de os servidores procurarem alterações que tenham ocorrido em clientes para tomar uma ação apropriada (LEWANDOWSKI, 1998).

Os clientes sempre aparecem localmente para a aplicação e eles escondem a localização e a conexão lógica necessárias para comunicar com as funções do servidor (COLS, 1993).

O que difere a arquitetura cliente/servidor da arquitetura distribuída são as características que se pretende alcançar em um sistema distribuído. Segundo COULOURIS, DOLLIMORE e KINDBERG (2000), um sistema distribuído deve suportar replicação, transação de dados compartilhados, controle de concorrência, transações distribuídas, recuperação e tolerância à falhas, e segurança. Estas características também são citadas por TANENBUAM (1994) que adiciona ainda: transparência, flexibilidade, confiabilidade, desempenho e escalabilidade. Outras características como abertura, heterogeneidade, acesso, e compartilhamento de recursos são reforçadas por EMERICH (2000). Este trabalho, por se propôr um *framework* para Ambientes Distribuídos, pretende focar as seguintes características de sistemas distribuídos:

Compartilhamento de recursos

A arquitetura cliente/servidor é um modelo comum para a distribuição de recursos. Muitos sistemas cliente/servidor executam os processos cliente e servidor em computadores separados. Na visão do cliente a obtenção de um serviço requer apenas o envio de uma requisição ao servidor e receber sua resposta (FEINSTEIN, 2000).

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, compiladores e outros objetos de dados (SOMMERVILLE, 2000). 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. É possível o compartilhamento em sistemas multi-usuários, mas este compartilhamento, muitas vezes deve ser fornecido e gerenciado por um computador central (SOMMERVILLE, 2000).

Em uma rede aonde há compartilhamento de recursos, estes estão fisicamente em um 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 (PASCUTTI, 2002) (SOMMERVILLE, 2000).

Escalabilidade

Um sistema distribuído é capaz de funcionar, eficientemente, em diversas escalas e possuir a capacidade de operar independente do seu crescimento. Este crescimento costuma ocorrer por uma demanda maior do sistema como a adição de novos recursos. Basicamente, um sistema distribuído 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 (TANENBAUM, 1994) (PASCUTTI, 2002) (SOMMERVILLE, 2000).

Transparência

Uma das características mais importantes de sistemas distribuídos é a forma como estes são apresentados aos usuários. Um sistema distribuído deverá ser visto pelo usuário como um sistema único e conexo (TANENBAUM, STEEN, 2002). A transparência permite ao usuário enxergar um sistema distribuído como uma única máquina e não como uma coleção de componentes independentes. A transparência pode ser dividida entre (PASCUTTI, 2002) (TANENBAUM, 1994):

- **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.

Além disto, há dois níveis distintos de transparência que um sistema distribuído pode possuir: esconder a distribuição do usuário ou esconder a distribuição dos programas. A primeira transparência é mais simples de ser conseguida pois implica em implementar programas que ofereçam transparência. A transparência para os programas é mais complexa pois necessita que hajam mecanismos que ofereçam esta transparência ao programa como ferramentas de infra-estrutura ou sistemas operacionais distribuídos (TANENBAUM, 1994) (CESETTI, AKAMATU, KIRNER, 1993).

O objetivo da transparência é permitir a opacidade da comunicação, escondendo do cliente qual servidor deve ser acessado, qual o formato da mensagem a ser utilizado, qual o protocolo de comunicação. Esta abordagem liberta o sistema para a mobilidade dos serviços e recursos através da rede aumentando sua escalabilidade (COLS, 1993).

Tomando, novamente, como base, a arquitetura cliente servidor, nota-se que esta possui dois modelos básicos dependendo de qual lado a carga de trabalho é assumida. No modelo cliente-magro e servidor-gordo o computador cliente não é responsável por mais que gerenciar a apresentação dos dados enquanto o servidor é responsável pela grande maioria das funcionalidades e processamento do sistema (FEINSTEIN, 2000) (LEWANDOWSKI, 1998).

Esta abordagem é baseada em clientes simples e de baixo custo de desenvolvimento para que sua implementação seja possível em várias plataformas. Isto implica, no fato de que o servidor possa ser mais complexo e o seu desenvolvimento mais caro, pois suas funcionalidades serão preservadas e utilizadas por vários clientes, em várias plataformas (COLS, 1993).

O cliente-gordo e servidor-magro é o oposto. O servidor é responsável apenas por gerenciar a base de dados enquanto o cliente faz praticamente todo o resto (FEINSTEIN, 2000). A responsabilidade do processamento e controle é do cliente, deixando a cargo do servidor a persistência dos dados (LEWANDOWSKI, 1998).

Segundo PRESSMAN (2001), um modelo de cliente gordo tende a ser distribuído pois as linhas guias de distribuição de aplicações são localizar a apresentação e os dados estáticos na máquina cliente e o banco de dados no servidor.

Objetos distribuídos

Há uma terceira abordagem que permite cliente e servidores gordos. Neste caso, há em ambos os lados o mesmo nível de flexibilidade e força. Esta abordagem é chamada de modelo de objetos distribuídos. Os objetos regulares residem em um programa simples e não existem como entidades separadas até que o programa seja compilado. Os objetos distribuídos são

objetos estendidos que podem residir em qualquer lugar em uma rede e continuam a existir como entidades físicas enquanto estiverem sendo acessados por objetos remotos. Sistemas de objetos distribuídos robustos permitem que estes objetos sejam escritos em linguagens de programação distintas e se comuniquem através de um protocolo de mensagens padronizado (LEWANDOWSKI, 1998).

Utilizando objetos distribuídos uma aplicação pode ser dividida em várias partes, podendo, cada uma destas, ser executada em um computador diferente em uma rede. Uma aplicação distribuída requer a interação entre partes que estão remotas (HEUSER, 1994). Para que esta interação seja possível, é necessário que a máquina cliente e a máquina servidor estejam, fisicamente, conectadas através da rede. Para realizar esta comunicação é utilizada uma camada de abstração chamada *middleware*.

Em um sistema distribuído o *middleware* é uma camada de *software* que fornece um significado transparente para o acesso a informação entre clientes e servidores (FEINSTEIN, 2000) (LEWANDOWSKI, 1998).

A tecnologia de *middleware* ocupa um papel central na arquitetura distribuída cliente/servidor. Com base na arquitetura básica cliente/servidor há as tecnologias distribuídas como CORBA (*Common Object Request Broker Architecture*), RMI (R*emote Method Invocation*), e DCOM (*Distributed Component Object Model*) (FEINSTEIN, 2000).

Tomando o CORBA como exemplo, cujo protocolo de comunicação é o TCP/IP, verifica-se que o ORB (*Object Request Broker*) é o *middleware*, o qual precisa ser instanciado tanto no lado cliente quanto no lado servidor. O ORB é responsável por conectar a aplicação cliente com o objeto que a mesma deseja utilizar e o cliente precisa conhecer apenas o nome do objeto a ser acessado (FEINSTEIN, 2000).

No caso de objetos distribuídos o objeto remoto acessado funciona como um serviço. A interface do serviço é definida em IDL (*Interface Definition Language*) no nível de implementação da aplicação. Esta interface apresenta ao cliente os serviços que o servidor fornece. A compilação do arquivo IDL gera um *skeleton* no servidor e um *stub* no cliente. A comunicação entre o cliente e o servidor se dá através do *skeleton* e do *stub* (FEINSTEIN, 2000).

2.1.4 Componentes

No início da programação, quando as máquinas eram programadas por fios e cada byte de armazenamento era precioso, as sub-rotinas foram inventadas para economizar memória. A função das sub-rotinas era permitir ao programador executar segmentos de código mais de uma vez, em diferentes circunstâncias (parametrizado), sem a necessidade de duplicar o código fisicamente em todos os trechos de programa que o mesmo fosse necessário. Todavia, esta era uma visão diferente do reuso que se tem atualmente, pois a reutilização era utilizada para poupar recursos físicos da máquina. Atualmente pensa-se em reutilização com a finalidade precípua de poupar recursos humanos (CLEMENTS, 2001).

Logo os programadores observaram que eles podiam inserir sub-rotinas extraídas de programas feitos anteriormente ou mesmo escritos por outros programadores e ter a funcionalidade sem ter que se preocupar com os detalhes do código. Geralmente estas rotinas úteis para reaproveitamento foram agrupadas em bibliotecas (CLEMENTS, 2001).

Este fenômeno representou um paradigma poderoso e fundamental para o desenvolvimento de *software* devido à visualização destas sub-rotinas como algo atômico. Atualmente a engenharia de *software* está empenhada em explorar e desenvolver a aplicação deste paradigma. O reuso em *software*, seus métodos e técnicas para aumentar a reusabilidade do *software*, inclui o gerenciamento de repositório de componentes (CLEMENTS, 2001).

Por componente entende-se uma unidade de *software,* relativamente auto-contida, que executa determinada tarefa sem a intenção de ser um sistema completo (CHAMBERS, LANG, 1999). Os componentes são a menor parte, independente e autogerenciável, de um sistema (LEWANDOWSKI, 1998).

Os componentes de *software* podem ser constituídos por componentes mais simples, assim o processo de construção de componentes pode ser considerado como recursivo. Nesse contexto, um componente pode ser não apenas código pronto para reutilização, mas também, unidades de projeto que têm significado próprio (GIMENES, BARROCA, HUZITA, CARNIELLO, 2000).

Para especificar totalmente um componente é necessário defini-lo em termos das operações que ele oferece e das operações que ele requer. Estas operações definem a comunicação do componente (GOMAA, MENASCÉ, SHIN, 2001).

A especificação da comunicação do componente é chamada de Interface do componente. Um componente oferece interfaces bem definidas para o meio externo. Estas interfaces separam sua especificação da implementação e não permitem que os usuários conheçam os detalhes de implementação do componente. A especificação de um componente é, normalmente, publicada separadamente de seu código fonte por meio da especificação das interfaces oferecidas por ele. Por meio destas interfaces, um componente pode se unir a outros componentes e dar origem aos sistemas baseados em componentes (GIMENES, BARROCA, HUZITA, CARNIELLO, 2000).

Após a criação de componentes e interfaces deve-se iniciar a atividade, pelo projetista, de agrupamento de componentes. Esta atividade visa sistematizar o grande volume de componentes, possivelmente gerado na atividade anterior, agrupando-os por critérios de acoplamento e coesão (BLOIS, BECKER, WERNER, 2004).

A especificação dos componentes é complementada através de características de componente que favorecem a comunicação de decisões de projeto, a catalogação e a recuperação dos componentes descritos. A descrição interna dos componentes é o nível de detalhes mais próximo da implementação disponível. De uma maneira geral, a especificação seria suficiente para apoiar a decisão sobre a adoção de um componente particular numa aplicação (MANGAN, WERNER, BORGES, 2002).

O DBC é o paradigma líder em reutilização que permite o trabalho de equipes dispersas ao longo do tempo e fisicamente distantes (WERNER et al., 2003). Isto porque os componentes prometem rápido desenvolvimento de aplicações e um alto nível de customização de maneira reduzir o custo de desenvolvimento (LEWANDOWSKI, 1998). Além disto, a utilização de componentes pode simplificar a manutenção do *software*.

Um *software* está sujeito a modificações em qualquer etapa de seu ciclo de vida. Para evitar re-trabalho e perda de informações, entre outros problemas, estas modificações devem ser controladas (LOPES, MURTA, WERNER, 2005). Isto requer uma arquitetura composta de componentes reutilizaveis de modo que os componentes possam ser substituídos individualmente por novas versões (BROWN, 1988).

A reutilização é um princípio importante uma vez que permite a construção de *software* por meio de unidades bem especificadas e testadas, disponibilizando aos usuários os aspectos variáveis e invariáveis de um componente através de sua interface (BLOIS, BECKER, WERNER, 2004).

É possível pensar, também, na reutilização em nível de artefatos resultantes do processo de desenvolvimento de *software* (processo de *software*), como idéias, conceitos, requisitos e projetos adquiridos ou construídos (GIMENES et al., 2000) (WERNER et al., 2003).

O DBC visa fornecer um conjunto de procedimentos, ferramentas e notações que possibilitem que, ao longo do processo de *software*, ocorra tanto a produção de novos componentes quanto à reutilização de componentes existentes (GIMENES, BARROCA, HUZITA, CARNIELLO, 2000).

As vantagens podenciais do DBC incluem (CLEMENTS, 2001):

- **Diminuição do tempo de desenvolvimento**: É uma grande economia de tempo comprar um componente ao invés de desenvolvê-lo, codificá-lo, testá-lo e documenta-lo;
- **Aumento de confiabilidade de sistemas**: Um componente comprado possivelmente já foi utilizado em vários outros sistemas e, provavelmente, seus erros potenciais já foram corrigidos;
- **Aumento de flexibilidade**: Desenvolver um sistema para que o mesmo utilize componentes significa que este foi construído imune aos detalhes de implementação destes componentes.

2.1.5 Framework

O paradigma orientado a objetos surgiu como um poderoso instrumento para o desenvolvimento de sistemas, principalmente em relação ao conceito de reutilização. Existem algumas técnicas aceitas na área de orientação a objetos para possibilitar a reutilização, dentre as quais pode-se citar o uso de padrões, componentes e *frameworks* para domínios específicos que poderiam ser instanciados para produzir novos produtos de domínio (DIAS, PIETROBOM, ALVES, 2004).

Em contraste com a abordagem tradicional de reuso de *software*, que é o paradigma de um conjunto de bibliotecas contendo várias funcionalidades, os *frameworks* orientados a objetos permitem um nível de abstração mais alto entre um número de sistemas similares a ser capturado em termos de seus conceitos gerais e estrutura. O resultado é um projeto genérico que pode ser instanciado para cada projeto a ser construído (LEWANDOWSKI, 1998).

Um *framework* pode ser descrito como uma arquitetura de *software* semi-definida consistente em um conjunto de unidades individuais e de interconexões entre elas, de tal forma a criar uma infra-estrutura de apoio pré-fabricada para o desenvolvimento de aplicações de um ou mais domínios específicos (GIMENES et al., 2000). Dentro de um domínio específico, um *framework* é uma ferramenta que ajuda o programador a alcançar seus objetivos (LEWANDOWSKI, 1998).

Desta maneira, um *framework* é, mais do que uma hierarquia de classes, uma base genérica para aplicações de um determinado domínio com estruturas estáticas e dinâmicas e

que pode ser reutilizada por muitas outras aplicações. Utilizando a tecnologia de orientação a objetos, o desenvolvedor pode especializar, instanciar, ou reutilizar as classes abstratas do *framework*, respeitando os relacionamentos já definidos (GIMENES et al., 2000).

O conceito de *framework* está intimamente ligado ao de Componentes no que tange à reutilização de *software*. Um *framework* pode ser decomposto em um conjunto de objetos ou componentes (GIMENES, HUZITA, 2005) pois é uma combinação destes componentes que simplifica a construção de aplicações e que pode ser conectada a uma aplicação (PETER, PEDRYCZ, 2001). *Frameworks* são projetos de sub-sistemas feitos com uma coleção de classes abstratas e concretas e as interfaces entre elas, seus relacionamentos e colaboração. Um *framework* não é uma aplicação. Uma aplicação é construída através da integração de um ou mais *frameworks* (SOMMERVILLE, 2000).

Freqüentemente *frameworks* sugerem padrões de colaboração entre objetos que constituem o *framework*. O benefício disto é um sistema de *software* cuja manutenção se torna mais simples (LEWANDOWSKI, 1998). Além disto, a utilização de um *framework* pode ajudar a garantir que o produto de *software* irá funcionar corretamente pois o mesmo possui em sua estrutura o conhecimento sobre o domínio aonde a aplicação irá atuar (LEWANDOWSKI, 1998).

Frameworks podem ser classificados dependendo de sua estrutura. Um *framework* caixa branca possui características de orientação a objetos como herança e possui vários componentes implementados. Um *framework* caixa preta é extensível por meio da definição de interfaces de componentes que podem ser implementados e plugados ao *framework*. De um ponto de vista abstrato, os diferentes tipos de *framework* são baseados nas mesmas características comuns. Os pontos onde o *framework* pode ser estendido são conhecidos como *extension points* (CAVANNES, 2003) (WIJNSTRA, 2000).

Um *framework* deve apoiar dois princípios básicos (WIJNSTRA, 2000):

- Reutilização binária de componentes;
- Divisão do sistema em partes genéricas e partes específicas.

A estrutura de um *framework* de componentes define uma estrutura de cooperação para um número fixo de componentes aonde alguns possuem implementações fixas e outros possuem implementações escaláveis (WIJNSTRA, 2000).

Um *framework* de aplicação é um meio termo entre uma aplicação customizada e componentes reusáveis específicos para um domínio específico. Sua intenção é customizar uma solução para um domínio específico (FREGONESE, ZORER, CORTESE, 1999).

Um *Framework* de aplicação deve possuir as seguintes características (CAVANNES, 2003):

- Compreender múltiplas classes ou componentes, cada qual provê uma abstração de um conceito em particular;
- Definir conceitos distintos que trabalharão juntos para resolver um determinado problema;
- Ser reutilizáveis;
- Permitir um alto nível de padronização de código.

2.1.6 Arquitetura em camadas

A arquitetura de *software* tem desempenhado papel fundamental no desenvolvimento de sistemas de *software*. Ela oferece um maior entendimento da aplicação por dividi-la em um conjunto de componentes que interagem entre si para realizar parte de uma funcionalidade, uma funcionalidade ou várias funcionalidades do sistema. Além disso, dispõe de métodos que garantem menores custos e tempo no desenvolvimento por prover reusabilidade de componentes ou estilos arquiteturais já existentes e, por permitir que vários times trabalhem, paralelamente, em partes diferentes do sistema (SILVA, PAULA, 2001).

Uma solução para o projeto de um sistema complexo é a utilização de uma arquitetura em camadas. A arquitetura em camadas propõe uma camada explícita para cada nível lógico do sistema de maneira a agrupar componentes que possuam o mesmo nível de abstração (MUNRO, 1993).

Um sistema em camadas é organizado hierarquicamente. As camadas mais altas são específicas da aplicação e as camadas mais baixas são de propósito geral, utilizadas por diferentes aplicações. Cada camada fornece um serviço para a camada acima e serve de cliente para a camada abaixo. Em alguns sistemas as camadas intermediárias são escondidas de tudo com exceção de suas camadas adjacentes e algumas funções cuidadosamente selecionadas para a exportação. Os conectores são definidos pelos protocolos que determinam como as camadas irão interagir. Tradicionalmente, as interações ocorrem somente entre camadas adjacentes mas a informação pode fluir em ambos os sentidos (SILVA, PAULA, 2001) (GARLAN, SHAW, 1994) (GARLAN, SHAW, 1993).

O exemplo mais comumente conhecido deste estilo arquitetural são as camadas dos protocolos de comunicação como o modelo de referência ISO da OSI. Nesta aplicação cada camada fornece um nível de abstração. As camadas mais baixas definem o nível baixo de interação, tipicamente definidas pelas conexões de hardware. Outras aplicações que utilizam este estilo arquitetural são bases de dados e sistemas operacionais (GARLAN, SHAW, 1994) (SILVA, PAULA, 2001).

A utilização de camadas auxilia o desenvolvimento pois, primeiramente, a construção de seus limites são claramente demarcados, reforçando as responsabilidades no sistema. Além disto, a divisão das responsabilidades guia para benefícios na manutenção do sistema facilitando a localização de erros ou falhas de desenvolvimento e, também, minimizando o volume e a complexidade do código fonte (DEARLE, 1988).

Caso o nível de abstração de cada camada seja bem definido, é possível criar uma interface de comunicação única para esta, evitando a comunicação direta entre os seus componentes. Neste caso cada camada no sistema é isolada das demais por meio da interface que ela apresenta (DEARLE, 1988). Isto aumenta o nível de manutenibilidade do sistema pois se a interface de uma camada é alterada, apenas a camada que possui contato com esta precisa ser alterada (MUNRO, 1993).

A comunicação entre as camadas ocorre por meio da troca de mensagens entre seus objetos. Tanto o conteúdo quanto a estrutura das mensagens para comunicação são diferentes de camada para camada sendo que as interações entre estas ocorrem apenas nos limites das mesmas. Isto causa uma abstração incremental e aumenta o reuso (GARLAN, SHAW, 1993) (BRAUN, DIOT, 1995).

Sistemas em camadas possuem várias características desejáveis. Primeiramente, eles apóiam o desenvolvimento baseado em níveis de abstração crescentes. Isto permite a

implementação dividir um problema complexo em uma seqüência de passos incrementais. Segundo, eles apóiam o crescimento. Como cada camada interage apenas com suas camadas adjacentes, alterações em suas funcionalidades afetarão no máximo outras duas camadas. Terceiro, elas apóiam o reuso. Como tipos abstratos de dados, diferentes implementações da mesma camada podem ser utilizadas caso elas utilizem a mesma interface. Isto leva à possibilidade de definir interfaces padrões para camadas de maneira que várias implementações possam ser feitas (GARLAN, SHAW, 1994).

Outra vantagem da utilização da arquitetura em camada é a possibilidade da existência de várias versões da mesma camada como um conjunto de ferramentas que pode ser composto ou alterado conforme a necessidade explícita do sistema. Esta generalização, baseada em um conjunto de ferramentas permutáveis como plug-ins, permite que o versionamento da arquitetura possa ser especializado de maneira mais apropriado para uma determinada implementação (BROWN et al., 1990) (BROWN, 1988).

Sistemas em camadas também possuem desvantagens. Nem todos os sistemas são facilmente estruturados em camadas. Mesmo que o sistema possa ser estruturado logicamente em camadas, considerações sobre desempenho podem necessitar um acoplamento maior entre funções que podem estar em camadas distantes. Ainda há a dificuldade de encontrar os níveis corretos de abstração para o desenvolvimento de um sistema em camadas (GARLAN, SHAW, 1994) (SILVA, PAULA, 2001).

2.2 Trabalhos relacionados

Há na literatura alguns trabalhos que vêm sendo desenvolvidos na área de Ambientes de Desenvolvimento de *Software*. Esta seção fará a apresentação de alguns destes. O ADS DiSEN será focado de maneira diferenciada por pertencer a nosso grupo de pesquisa e ser o ambiente aonde este *framework* será instanciado.

2.2.1 Adele/Tempo

O Adele / Tempo foi desenvolvido no IMAG de Grenoble. O Adele é uma base de dados versionada aonde os componentes do processo são armazenados. Os passos do processo são modelados como objetos e definem as operações, atributos e, recursivamente, outros passos do processo (BARTHELMESS, 2003). O Tempo é um PSEE baseado no Adele que busca uma maior estratégia de produção e evolução de sistemas de *software*. O tempo é guiado por um formalismo de execução que permite descrever o modelo de processo, a visão dos objetos, a elaboração das estratégias de coordenação e comunicação (BELKHATIR, MELO, 1993).

Além de ser uma base de dados versionada, o Adele é uma base de dados ativos, ou seja, com regras do tipo Evento-Condição-Ação. Ele suporta um modelo entidade-relacional que é ampliado com conceitos de orientação a objetos como métodos e encapsulamento. Objetos simples e compostos com atributos e relacionamentos podem ser descritos e gerenciados (BELKHATIR, MELO, 1994). O produto de *software* é descrito pelo modelo de dados do Adele e provido pela base de dados do Adele. (BELKHATIR, MELO, 1994-b).

O Adele tenta integrar dois aspectos do desenvolvimento de *software*: o aspecto estático através da modelagem dos dados e produtos; e ao aspecto dinâmico pela modelagem

do processo. Os requisitos restritos dos objetos são criados com o modelo de dados para suportar manutenção, versão, complexidade, propagação e gerenciamento de transações longas. O Adele possui ainda várias outras características como suporte ao trabalho cooperativo, gerenciamento de controle e configuração, RPC e GUI, cópias lógicas e fusão de dados (BELKHATIR, ESTUBLIER, MELO, 1994).

As partes básicas do Adele (BELKHATIR, MELO, 1993):

1. Um **gerenciador de recurso** que utiliza a base de objetos de persistência do Adele;
2. Um **gerenciador de atividades** que utiliza regras temporais e mecanismos automáticos para prover suporte ao ambiente. É função do gerenciador de atividades controlar a integração da plataforma;
3. Um **gerenciador de processo** que possui os conceitos de processos e funções. A execução de um processo é pelos ambientes de trabalho aonde as atividades são realizadas. O gerenciador de processo baseado no gerenciador de atividades gerencia a comunicação e a sincronização entre os agentes envolvidos no mesmo projeto. Ele ainda controla a consistência de objetos complexoss que são usados simultaneamente por diferentes agentes (BELKHATIR, MELO, 1994).

Estes componentes são integrados através da linguagem Tempo que descreve o esquema de dados e o modelo comportamental do sistema. O desenvolvimento de Adele foi focado na integração de dados e de controle através da construção de um banco de dados ativo de engenharia de *software* e, mais recentemente em integração de processo (LIMA REIS, 1998).

Em uma visão mais precisa, Tempo é uma linguagem de modelo de processo baseada em regras aonde as partes dos eventos podem incluir expressões temporais. Estas regras são utilizadas para controlar as atividades (BARTHELMESS, 2003).

A linguagem de programação de processo Tempo é baseada no conceito de papel (role). Um processo de *software* é definido como um conjunto de objetos exercendo um papel. Um tipo de processo identifica e descreve um conjunto de atividades. Uma instância de processo de *software* é executada por um ou mais usuários em um ambiente de trabalho (LIMA REIS, 1998).

No Adele / Tempo, os usuários operam em um ambiente de trabalho privado que contém copias dos artefatos armazenados na base de dados. Um gerenciador de transação gerencia os possíveis conflitos que podem ocorrer quando diferentes usuários operam sobre o mesmo artefato em paralelo. Neste ambiente os usuários executam suas atividades em seus próprios ambientes de trabalho. Há uma sub base de dados Adele associada com cada usuário. Os ambientes de trabalho são compostos por diretórios e arquivos, ferramentas e um executor de tarefas (BARTHELMESS, 2003).

2.2.2 CAGIS

De 1986 a 1996 um grupo de pesquisa coordenado por Reinar Conardi trabalhou em um protótipo de PCE (*Process Centered Environment*) chamado EPOS. O EPOS foi um ambiente avançado para gerenciamento de processos de *software* e também artefatos de *software* através de várias ferramentas. Em 1997 um projeto chamado CAGIS (*Cooperative Agents in Global Information Space*) foi iniciado. Um dos objetivos principais do CAGIS foi

quantificar o suporte ao trabalho distribuído, cooperativo e heterogêneo em um ADS entre equipes geograficamente dispersas (WANG, 2000)

No contexto do projeto foi desenvolvido um ambiente de suporte à execução de processos, o CAGIS PCE, que consiste de três componentes principais (RAMAMPIARO, WANG, BRASETHVIK, 2000) (WANG, 2000-a) (WANG, 2000-b) (WANG, 2002):

- O CAGIS SimpleProcess é uma ferramenta de workflow que provê suporte à definição e à execução de atividades que envolvem apenas uma equipe;
- O CAGIS *Distributed Intelligent Agent System* (DIAS) utiliza agentes móveis para prover suporte a atividades cooperativas, envolvendo mais de uma equipe;
- O CAGIS GlueServer especifica regras de cooperação entre atividades individuais e cooperativas, possibilitando a interação entre o CAGIS SimpleProcess e o CAGIS DIAS.

O foco do CAGIS é apoiar atividades que envolvam mais de uma equipe para sua execução. Ele interliga ferramentas de workflow, utilizando agentes de *software* para realizar atividades que envolvam mais de um workflow suportando federações de PSEEs (FREITAS, MAIA, NUNES, 2004-b) (FREITAS, MAIA, NUNES, 2004).

A arquitetura do CAGIS SimpleProcess é baseada na arquitetura Web tradicional sendo constituída de quatro aplicações CGI (WANG, 2002), um **servidor de processo** responsável por gerenciar mudanças no processo e o estado do processo, um **modelador de processo** para a definição do processo, um **gerenciador de agenda** que apresenta as atividades dos usuários e uma **ferramenta de monitoramento** e acompanhamento. Esta arquitetura pode ser visualizada na forma de quatro componentes (WANG, 2000) (RAMAMPIARO, BRASETHVIK, WANG, 2000):

1. **Agentes** – utilizados por possuir características autonômicas, de interação, reatividade e pró-atividade. Há três tiupos principais de agentes: *Work agents* para auxiliar nas atividades de produção de *software*, *Interactions agents* para auxiliar o trabalho cooperativo e *Systems Agents* para dar suporte aos demais agentes;
2. ***Workspace*** – Um *container* temporário para dados em um formato de arquivo interoperável com ferramentas para o processamento destes dados;
3. **AgentMeetingPlace** – O “local” aonde os agentes interagem. Uma espécie de *workspace* para agentes;
4. **Repositórios** – Podendo ser global, local ou distribuído[2], os repositórios são persistentes. É dos repositórios que os *workspaces* recebem seus dados.

No CAGIS os documentos e suas classificações são armazenados no servidor Web no formato XML e HTML/TXT. O modelo do domínio também é armazenado em XML e pode ser alterado para refletir o vocabulário utilizado nos documentos (RAMAMPIARO, BRASETHVIK, WANG, 2000).

---

[2] O repositório local se encontra na própria estação de trabalho. O repositório global encontra-se em um servidor. O repositório distribuído encontra-se em mais de um local podendo estes ser servidores ou estações de trabalho.

2.2.3 Endeavors

Desenvolvido pela *University of California at Irvine*, o Endeavors é um PSEE aberto e flexível baseado na Internet e seus protocolos básicos como HTTP e FTP. Ele surgiu de uma experiência com a linguagem de modelagem de processos Teamware (HITOMI, BOLCER, TAYLOR, 1997). Um de seus objetivos principais são prover a flexibilidade ao processo de *software* pela minimização dos esforços necessários para à alteração durante a sua execução. O Endeavors suporta a definição orientada a objetos e a especialização de atividades, artefatos e recursos associados com o processo de desenvolvimento de *software* (CUGOLA, GHEZZI, 1998) (HITOMI et al., 1998-b).

A especificação dos processos e objetos podem ser definidas hierarquicamente, uma atividade pode ser ainda sub-dividida em sub-atividades, ou uma equipe de desenvolvimento pode ser definida para incluir sub-equipes. Além disto, o Endeavors tem políticas customizadas de descentralização e distribuição que provê suporte para a distribuição transparente de pessoas, artefatos, objetos do processo e ambientes de execução (HITOMI, BOLCER, TAYLOR, 1997).

O Endeavors interage com o desenvolvedor através de agendas. Existem também vários mecanismos para a comunicação com outros desenvolvedores, como marcação de reuniões e anotações sobre documentos manipulados (SOUSA, LIMA REIS, REIS, PIMENTA, NUNES, 2001). As políticas do Endevors podem ser customizadas para reforçar a execução do processo, para manter a consistência dos dados ou para coordenar as atividades dos usuários evitando a perda de confiabilidade (HITOMI et al., 1998-b).

O Endeavors adota uma arquitetura complexa para distribuir o gerenciamento do processo. Várias engines de processos podem coexistir. Eles se comunicam com o padrão de conexão ponto-a-ponto. Este tipo de abordagem reduz a possibilidade de alteração no sistema em tempo de execução (CUGOLA, GHEZZI, 1999).

Ainda sobre sua arquitetura, a mesma é implementada em camadas como um conjunto de ferramentas componentizadas, leves, de elementos concorrentes, fornecendo ferramentas para a customização para cada camada lógica do sistema. Também são possíveis a extensão da arquitetura, interfaces e formato dos dados em cada uma das camadas (HITOMI, BOLCER, TAYLOR, 1997).

As camadas da arquitetura são divididas em: usuário, sistema e fundação. Destas três, a camada intermediária e a superior (Sistema e Usuário) foi aumentada com a especialização do servidor HTTP e email POP3 através de um componente maior chamado WebNavigation. O WebNavigation foi desenvolvido e planejado para fornecer uma ponte entre o Endeavors e o servidor Web, trocando informações e interpretando a estrutura de dados entre estes dois sistemas. A camada do usuário é responsável por manter consistente a visualização dos usuários através do gerenciamento de atualizações coordenadas (HITOMI, LEE, 1998).

A infra-estrutura do ambiente é aberta quanto à modelagem e execução de processos distribuídos que suporta comunicação, coordenação e controle. A solução arquitetural do Endeavors possui cinco características (HITOMI, BOLCER, TAYLOR, 1997):

- Manutenção de múltiplas camadas de modelos de objetos;
- Implementação da arquitetura como um conjunto de elementos componentizados e concorrentes;

- Disponibilização de componentização customizada de para cada camada do modelo de objetos;
- Utilização de modelo de objetos reflexivo para suportar alterações dinâmicas;
- Suporte a carga e alteração dinâmica de objetos.

O projeto do Sistema Endeavors provê mecanismos chaves para suporte à distribuição de processos e usuários, integração de ferramentas externas, adoção incremental de tecnologia, customização e reusabilidade de processos e suporte a alterações dinâmicas em tipos e comportamentos (HITOMI, BOLCER, TAYLOR, 1997).

A integração do ambiente a plataformas de *software* exige poucos esforços. Todos os objetos do processo são baseados em ASCII permitindo uma grande portabilidade mesmo em arquiteturas distintas. Os componentes do sistema, incluindo as interfaces dos usuários, podem ser baixados conforme necessários e não há necessidade de uma instalação explícita para visualizar e executar um processo. A comunicação do ambiente é bi-direcional entre ferramentas e objetos internos e externos e serviços. Isto graças a uma inteface aberta ortogonal a todos os níveis de sua arquitetura (HITOMI, BOLCER, TAYLOR, 1997).

O ambiente suporta também a comunicação baseada em eventos entre as camadas e componentes incluindo os componentes de interfaces do usuário. Seu modelo de componentes é reflexivo de maneira a manter um modelo interno customizável, dinâmico e distribuído. Além disto o Endeavors permite a carga de objetos, comportamentos e interfaces através da Internet (HITOMI, BOLCER, TAYLOR, 1997).

O projeto Endeavors propõe um mecanismo para execução de processos distribuídos e integração de ferramentas utilizando o HTTP com Java Servlets (HITOMI, LEE, 1998). O sistema utiliza um modelo de objetos para fornecer definições e especificações orientadas a objetos de artefatos do processo, atividades e recursos. A intenção da distribuição é suportar uma grande gama de configurações com vários níveis e tipos de distribuição (AVERSANO et al., 2004).

2.2.4 EPOS

O EPOS (*Expert System for Program and ("og") System Development*) foi desenvolvido pela *University of Tronheim*, Noruega, desde 1989, primeiramente como parte de um projeto nacional e depois como um projeto de Ph.D. (AMBRIOLA, CONRADI, FUGGETTA, 1997). Sua estrutura básica possui dois componentes (BARTHELMESS, 2003):

- Um **gerenciador de execução** que trabalha sobre pré-condições das tarefas e define quais tarefas devem ser atômicas.
- Um **planejador** que gera um novo conjunto de subtarefas e representa as pós-condições de uma tarefa como seu objetivo.

O EPOS é construído sobre uma base de dados própria (EPOS-DB) que oferece versão e transações cooperativas. O EPOS-DB oferece transações com controle de versões cooperativas e não serializáveis (AMBRIOLA, CONRADI, FUGGETTA, 1997).

O EPOS enfatiza a gerência de processo de *software* para múltiplos usuários cooperativos. EPOS é baseado em uma PML híbrida chamada SPELL, a qual consiste de uma extensão do sistema de banco de dados orientado a objetos EPOS-DB. O EPOS possui também uma linguagem específica chamada WUDL (*Workspace Unit Declaration Language*)

para descrever conflitos de acessos (BARTHELMESS, 2003). Este ambiente fornece ferramentas que incluem (SOUSA et al., 2001) (LIMA REIS, 1998) (AMBRIOLA, CONRADI, FUGGETTA, 1997):

- ***Schema Manager*** - responsável pela navegação, edição, definição, análise, tradução e desenvolvimento gráfico e textual do esquema de processo, e pode ser usado em todos os modelos;
- ***Task Network Editor*** - permite manipular a rede de tarefas antes e durante a execução, e suporta recursos como incluir/excluir/mover tarefas e produtos;
- ***Planner*** - auxilia na decomposição das tarefas em uma rede hierárquica,
- ***Project Manager*** - usado para iniciar e encerrar um projeto e recuperar métricas a partir do EPOS-DB.
- ***Process Engine*** **-** consiste no *SPELL Interpreter* e *Execution Manager* e é usado para executar a rede de tarefas instanciada pelo *Planner*;
- ***Workspace Manager*** **-** possui recursos para gerência do espaço de trabalho;
- ***Cooperation Manager*** **-** gerencia as transações cooperativas sobre os objetos. A instanciação do Gerente de Processos do EPOS é feita pelo componente *Planner*, o qual é invocado pelo Gerente de Execução responsável pela execução do processo.

A Arquitetura do EPOS é baseada em quatro camadas (AMBRIOLA, CONRADI, FUGGETTA, 1997):

- Uma base de dados proprietária e cliente-servidor chamada EPOS-DB que armazena modelos de processos com transações versionadas. Ela é estruturalmente um modelo de dados orientado a objetos com seu próprio modelo de versões orientadas a alterações.
- A habilitação do modelo de processo nas estações clientes através das ferramentas EPOS-PM (*Process Manager*).
- Um interpretador de SPELL nos locais clientes para fornecer reflexão e orientação a objetos nos dados acessados do EPOS-DB.
- Ferramentas de processo para manipular o modelo de processo como ferramentas de planejamento, agendamento de tarefas, gerenciador de cooperação, gerenciador de *workspace* entre outras.

No EPOS os documentos de *software* são armazenados na base de dados. A base de dados não sabe nada sobre o conteúdo interno dos arquivos e por esta razão a integração semântica dos dados não pode ser feita (AMBRIOLA, CONRADI, FUGGETTA, 1997).

O EPOS é mais próximo do alto nível que a maioria dos ambientes baseados em regras. Ele possui tarefas primitivas que são essencialmente as regras de planejamento do ambiente com suas regras de entrada / saída que troca parâmetros entre as tarefas (KAISER, POPOVICH, BEN-SHAUL, 1993).

O ambiente EPOS é executado em estações de trabalho baseadas no sistema operacional UNIX. É implementado em C e PROLOG. A base de dados EPOS (EPOSDB) baseia-se no banco de dados relacional INGRES em um ambiente cliente-servidor baseado em RPC (*Remote procedure Call*) da Sun. A interação com o usuário é feita através do pacote

gráfico PCE baseado em Prolog. Ele possui várias ferramentas de suporte incluindo um editor de produtos (CONRADI et al., 1991).

O repositório de artefatos permite as ferramentas convencionais de controle de versões como cópia, modificação e mesclagem. Além de permitir cópias do repositório e para o repositório (*checkin – check-out*). O repositório local é não versionado. Um repositório simples é composto basicamente de três partes (CONRADI et al., 1991):

- A base de conhecimento do projeto como pessoas e papéis
- A rede de tarefas associadas
- A base de dados de produtos como arquivos e diretórios representando uma versão selecionada de um subsistema em particular na base de dados real.

2.2.5 ExPSEE

Desenvolvido pelo grupo de Engenharia de *Software* da Universidade Estadual de Maringá de 1994 - 2002, o ambiente ExPSEE - *Experimental Process-centered Software Engineering Environment* – é apoiado por projeto financiado pelo CNPq (STEINMACHER et al., 2002).

O ExPSEE é um ambiente de engenharia de *software* orientado a processo que suporta a especificação e automação de processos de *software*, permitindo a integração de ferramentas CASE e a cooperação no desenvolvimento e execução dos processos envolvidos reduzindo os custos relativos ao processo de *software* e sua manutenção. Este ambiente é composto por três subsistemas: gerenciamento de interface, gerenciamento de processos e gerenciamento de banco de dados.

A primeira versão do ExPSEE foi desenvolvida na plataforma Sun Solaris, utilizando o sistema gerenciador de banco de dados orientado a objetos (SGBDOO) ObjectStore, além de outros mecanismos para a sua execução.

O ExPSEE está baseado na definição explícita do processo por meio do qual os artefatos de *software* serão concebidos, projetados, desenvolvidos e distribuídos. Esta definição explícita do processo se baseia no modelo de processo adotado para o ambiente.

As linguagens de programação utilizadas para a implementação dos módulos do gerenciador e dos seus módulos de interface foram, respectivamente, C++ e Tcl/Tk. A linguagem de programação C++ foi escolhida para a implementação dos módulos do gerenciador devido ao seu amplo uso e à adoção do SGBDOO ObjectStore com interface para C++. Para a implementação do ambiente, foram utilizados os compiladores C++ da Sun e o GNU C Compiler.

Como mecanismo de comunicação interprocessos foi utilizado o RPC (Remote Procedure Call). Este mecanismo permite a construção de programas distribuídos com chamadas de procedimentos remotos muito similar às chamadas de procedimentos convencionais.

O gerenciador de processos é o subsistema mais importante do ambiente ExPSEE, pois este é responsável por controlar a criação, modificação e execução dos processos de *software*, dando manutenção ao estado atual de cada tarefa e do processo como um todo. O gerenciador de processos também controla a disponibilidade de alocação de recursos

utilizados nas tarefas, bem como o controle dos cargos do processo, assegurando que nenhum direito seja violado (GIMENES, 1999) (STEINMACHER et al., 2002).

O gerenciador de processos do ExPSEE é composto por vários módulos gerenciadores, que interagem entre si e comunicam-se com o SGBDOO ObjectStore, sendo eles (STEINMACHER et al., 2002):

- **Gerenciador de Projetos** - responsável pelo controle e gerenciamento da execução dos processos de *software*;
- **Gerenciador de Meta-Processos** - responsável pelo controle e gerenciamento da construção e manutenção de arquiteturas de processo de *software* e sua instanciação através da definição de processos de *software*;
- **Gerenciador de Tarefas** - responsável pelo controle e gerenciamento das tarefas e ações a serem realizadas no processo de *software*;
- **Gerenciador de Artefatos** - responsável pelo controle e gerenciamento dos artefatos utilizados e produzidos pelas tarefas através das ferramentas;
- **Gerenciador de Ferramentas** - responsável pelo controle e gerenciamento das ferramentas utilizadas pelas tarefas no processo de *software*;
- **Gerenciador de Cargos** - responsável pelo controle e gerenciamento dos cargos existentes no processo de *software* a serem ocupados pelos atores presentes no processo;
- **Gerenciador de Ações** - responsável pelo controle e gerenciamento das ações que as ferramentas desempenham sobre os artefatos do processo de *software*; e
- **Gerenciador de Atores** - responsável pelo controle e gerenciamento dos atores, e suas agendas, envolvidos no processo de *software*.

2.2.6 GENESIS

O GENESIS (*Generalized Environment for Process Management In cooperative Software Engineering*) é uma plataforma distribuída de código aberto que suporta engenharia de *software* cooperativa que mantém as características de um sistema distribuído e que permite o trabalho em grupo. Todos os aspectos necessários para cooperação, colaboração e coordenação têm sido derivados de necessidades de usuários coletadas de algumas das maiores companhias de desenvolvimento de *software* do cenário europeu (BALLARINI et al., 2003) (AVERSANO, CIMITILE, DE LUCIA, 2003).

Uma estação do GENESIS inclui alguns componentes, entre eles (AVERSANO, CIMITILE, DE LUCIA, 2003) (BALLARINI et al., 2003):

- Um **gerenciador de *workflow*** para modelar e habilitar o processo de *software*;
- Um **gerenciador de artefato** para armazenar os artefatos produzidos no processo;
- Um **gerenciador de recursos** para a alocação de recursos, em particular a alocação de recursos humanos;
- Uma **engine de eventos** para coletar e disparar eventos surgidos durante o gerenciamento do processo como o término de uma atividade ou a produção de um artefato;

- Um **sistema de comunicação** e notificação para permitir que os usuários do ambiente comuniquem-se entre si.

O foco do projeto GENESIS é projetos multi-site aonde cada site é apto para executar instâncias de um processo de *software* ou subprocessos que aceitam artefatos de *software* como entrada do processo e gera artefatos como saída. Estes artefatos formam a base da comunicação e interação inter-sites (BOLDYREFF et al., 2003).

O gerenciamento de artefatos no GENESIS é um sub sistema chamado OSCAR. O controle de versões de artefatos é feito através de uma abstração sobre as fncionalidades do sistema de gerenciamento de configuração. Em uma instância particular do OSCAR há um mapeamento para um serviço de SCM (*Software Configuration Manager*) como o CVS. Desta maneira, a opção pelo SCM é livre para cada local (BOLDYREFF et al., 2003). Como base de dados o GENESIS utiliza o MySQL (AVERSANO et al., 2004).

O Sistema gerenciador de artefatos e configuração é baseado em quatro camadas. Na camada inferior há o repositório de artefatos propriamente dito. O repositório é responsável pela criação e destruição de artefatos em sistemas de armazenamentos externos. A camada de indexação e busca fornece a ordenação dos dados armazenados para a camada de apresentação que é responsável pela conexão de clientes, a transformação dos dados e a segurança. Através destas três camadas há a camada de métrica que armazena dados coletados por agentes (BALLARINI et al., 2003).

No ambiente GENESIS um protocolo assíncrono foi definido para a comunicação entre os níveis de coordenação global e a coordenação local durante a instanciação de um projeto de *software* distribuído (AVERSANO, CIMITILE, DE LUCIA, 2003). A comunicação entre a camada de coordenação e os diferentes subsistemas é baseada em Java RMI enquanto a comunicação entre locais diferentes é baseada em SOAP (AVERSANO et al., 2004). Há também no ambiente comunicação síncrona gerenciada por um sistema de fórum e comunicação assíncrona gerenciada por um sistema de email (AVERSANO et al., 2004).

A comunicação envolve não apenas a comunicação formal como a especificação de documentos mas também a comunicação informal como anotações pessoais a cerca da escolha de um projeto. Esta comunicação é realizada utilizando-se a integração de dados e controles pelo Gerenciador de artefato de cada local.

Na plataforma GENESIS a coordenação entre locais é baseada em Eventos e agentes de *software*. O gerenciador de eventos notifica eventos que acontecem durante a execução do processo, coletando métricas apropriadas para monitorar atividades locais. Um usuário se inscreve para receber eventos relacionados ao processo ou atividade de acordo com as políticas de visibilidade locais ou globais.

O GENESIS permite distribuição e controle das atividades de desenvolvimento de *software* entre locais geograficamente distribuídos. Cada site pode executar uma instância de um processo ou sub-processo de *software* aonde a entrada e a saída da atividade do processo é um artefato de *software*. Cada site executa uma instância localmente de seu próprio modelo de processo e interage com outros sites por meio dos artefatos a serem produzidos ou consumidos em cada atividade do processo (BALLARINI et al., 2003).

O ambiente GENESIS possui uma linguagem de modelagem de processos que é utilizada para decompor processos complexos em sub-processos que podem ser distribuídos e executados em diferentes sites. No GENESIS a tecnologia de gerenciamento de workflow tem sido integrada com o gerenciamento de artefatos e a comunicação entre serviços para

satisfazer os requisitos necessários para o gerenciamento entre equipes distribuídas (AVERSANO et al., 2004).

O aspecto de coordenação de cada site é feito por uma engine de workflow que permite a organização do processo de *software* aplicada a cada local. O gerenciador de Workflow é baseado em uma arquitetura cliente-servidor e disponibilizado para os usuários por páginas Web. Este gerenciador possui três componentes básicos: Uma engine que controla e gerencia a execução de processos e interage com a base de dados, uma ferramenta de administração e monitoramento que provê facilidades para os usuários iniciarem execução de um processo e monitorarem o seu estado durante a execução, e um cliente Web que fornece aos usuários acesso às atividades cuja responsabilidade lhe foi designada (BALLARINI et al., 2003).

O GENESIS partiu do mapeamento do interesse dos usuários quanto às funcionalidades desejáveis em um ADS. Foi observado que o mesmo está no controle e coordenação do projeto, nas métricas do projeto, no armazenamento e recuperação de artefatos de maneira colaborativa e na colaboração entre usuários. O projeto GENESIS identificou as seguintes funcionalidades principais (BALLARINI et al., 2003):

1. Controle do processo local e global;
2. Gerenciamento de conflitos e incidentes;
3. Métrica, monitoramento e alarme;
4. Repositório de documentos protegido, compartilhado e indexado;
5. Comunicação e coordenação;
6. Acompanhamento por Logs.

O Gerenciador de recursos é um componente responsável por gerenciar os dados de usuários e de projeto alocando usuários a projetos e controlar o acesso a plataforma. Suas principais funções são:

- Controlar os dados para acesso (*login*);
- Criar e gerenciar locais das organizações que colaboram em um projeto;
- Criar e gerenciar projetos de cada organização;
- Criar e gerenciar os recursos humanos;
- Criar e gerenciar os dados de usuários e seus perfis;
- Criar e gerenciar a posição de negócio da organização.

Para outros componentes, o gerenciador de recurso irá fornecer os seguintes serviços:

- Fornecer a lista de projetos;
- Modificar o estado do projeto;
- Fornecer a lista de usuários alocado em um determinado projeto;
- Adicionar, remover e alterar usuários;
- Buscar por um usuário;
- Criar e remover uma alocação de recurso.

Na plataforma GENESIS cada local tem total autonomia quanto à escolha de ferramentas ou modelo de processo.

2.2.7 MARVEL

O MARVEL (*Management Aggregation and Visualization Environment*) é um projeto da Columbia University liderado por Kaiser (BARTHELMESS, 2003). Uma primeira versão do ambiente para programação em C foi desenvolvida e chamada C/Marvel. A versão 2.01 do ambiente C/Marvel foi demonstrada no ACM SIGSOFT *Practical Software Development Environmement* em Novembro de 1988. Até a versão 2.6 o Marvel era mono usuário. Na 4a. ACM SIGSOFT *Symposium on Software Development Environment* o Marvel 3.0 foi apresentado. Esta versão do ambiente apoiava múltiplos clientes e acesso concorrente ao servidor em uma arquitetura cliente/servidor (KAISER, BEN-SHAUL, BARGHOUTI, 1990) (KAISER et al., 1993).

O projeto MARVEL foi continuado em área de pesquisa da AT&T Labs que investigou as limitações da tecnologia *Web* no gerenciamento de redes e propôs uma arquitetura que possa ser gerenciada de maneira escalável e convencional em termos de desempenho e escalabilidade (ANEROUSIS, 1999). O ambiente MARVEL deu origem ao PSEE Oz.

O MARVEL não é limitado apenas ao modelo de interação web. Qualquer tipo de aplicação cliente pode ser construída em seu modelo computacional de serviço distribuído, de aplicações tradicionais utilizando sua próprio GUI, a interfaces guiadas ao servidor. Para isto conceitos de agentes móveis, RMI e CORBA foram incorporados ao ambiente de maneira que o mesmo possa ser integrado pelos mais diversos aplicativos como ferramentas de modelagem ou de desenvolvimento (ANEROUSIS, 1999).

Todo sistema MARVEL é composto de uma hierarquia de servidores. A arquitetura distribuída foi escolhida por várias razões (ANEROUSIS, 1999):

- As visões do gerenciamento de informações requerem recursos de processamento consideráveis e as mesmas devem estar sempre atualizadas;
- A computação pode utilizar fontes de informações próximas evitando um aumento desnecessário do tráfego de rede que seria causado por um servidor centralizado;
- Uma arquitetura distribuída é mais confiável pois a falha de um servidor pode ser suprida por outro. Desta maneira a manutenção dos serviços pode ocorrer sem que os mesmos sejam interrompidos;
- Uma arquitetura distribuída pode ser implementada com a junção de componentes com um baixo custo e uma maior simplicidade.

Os objetos do MARVEL são definidos diretamente em Java e persistido em uma base de dados. Todo objeto é derivado (herança) de uma classe de objetos MARVEL que fornece um padrão para implementação que possui facilidade para o registro deste objeto junto ao servidor. Um cliente mínimo do MARVEL possui a capacidade de carregar o código de inicialização cliente de um servidor MARVEL. O código de inicialização permite que o mesmo acesse os serviços do MARVEL. A arquitetura favorece clientes mínimos que suportam a visualização de páginas HTML e *Applets* Java. Cada servidor mantém uma página Web que, quando carregada pelo cliente inicia um applet de navegação que permite ao cliente

examinar a base de dados do servidor e invocar funções de visualização de objetos (ANEROUSIS, 1999).

A arquitetura básica do MARVEL é composta de cinco componentes (KAISER, BEN-SHAUL, BARGHOUTI, 1990):

- Uma linguagem para definição do modelo de processo (chamada *MARVEL Strategy Language - MSL*);
- Um par de interfaces para o usuário, uma texto e outra gráfica;
- Uma *engine* para a persistência;
- Um sistema de gerenciamento de objetos.

O ambiente MARVEL é dividido em um repositório compartilhado chamado "Área Central" e uma coleção de *workspaces* privativos chamados "mini projetos" (KAISER et al., 1993).

O repositório MARVEL é uma base de dados orientada a objetos implementada sobre o sistema de arquivo do UNIX. Para utilizar um repositório orientado a objetos, os objetos persistidos no MARVEL são instâncias de uma classe (SOKOLSKY, KAISER, 1991). Cada objeto do ambiente é representado por arquivos e o ambiente possui uma linguagem de modelagem de dados que suporta composição e relacionamento entre objetos (BARGOUTHI et al., 1996). Cada artefato gera dois arquivos: um que é o próprio artefato e um segundo arquivo oculto com a representação textual dos dados deste artefato (KAISER, BEN-SHAUL, BARGHOUTI, 1990). Os artefatos persistidos podem possuir herança e a raiz desta classe de herança é uma classe chamada Entidade (*ENTITY*) (SOKOLSKY, KAISER, 1991).

O acesso ao sistema de gerenciamento de objetos (*OMS - Object Management System*) é restrito e o controle da concorrência das atividades é feita por um gerenciador de transações (TM). Para o acesso compartilhado a dados é utilizado um gerenciador de travas (*Lock manager*) que libera os objetos para leitura, escrita ou ambos. Quando dois usuários tentam acessar o mesmo objeto, o gerenciador de travas comunica o gerenciador de transações que resolve a concorrência por meio da serialização das transações. Para alterar esta serialização o MARVEL utiliza uma linguagem chamada CRL (*Control Rule Language*) para modificar as políticas de controle de concorrência (BARGOUTHI et al., 1996).

O sistema gerenciador de transação é constituído de três camadas que suportam controle a concorrência entre vários clientes. A camada inferior é um gerenciador de trava que detecta conflitos de acesso em potencial. A camada intermediária resolve conflitos através de um protocolo baseado em semântica que entende a distinção de consistência no modelo do processo. Esta camada suporta recuperação de falhas e é capaz de retornar a situação anterior caso haja uma inconsistência. A camada superior é um gerenciador de transação que serializa requisições ao gerenciador de objetos (KAISER et al., 1993).

O MARVEL possui um suporte limitado para integração e interoperabilidade. Como seu repositório é baseado em sistemas de arquivos, os dados tornam-se disponíveis para ferramentas externas acessá-los diretamente. O MARVEL possui uma interface de programação chamada SEL (*Shell Envelope Language*) para a definição da interface entre ele e ferramentas externas (BARGOUTHI et al., 1996).

As regras determinam o comportamento do MARVEL especificando o processo de desenvolvimento de *software* em termos de suas atividades e as interações entre atividades. As regras podem definir os requisitos necessários para cada passo do processo através da dependência das atividades. Isto é feito através de regras compostas de três partes: pré-

condição, ação e pós condição. Isto garante que o fluxo do processo seja obedecido e guia os usuários do ambiente a próxima fase do processo (SOKOLSKY, KAISER, 1991) (KAISER, BARGHOUTI, SOKOLSKY, 1990). Para a definição de processos o MARVEL possui a Marvel Strategy Language (KAISER, POPOVICH, BEN-SHAUL, 1993).

A ferramenta de política é um cliente do servidor MARVEL e comunica com o servidor usando o mesmo protocolo que a GUI do MARVEL (POPOVICH, 1992).

O ambiente possui apenas uma base de dados centralizada responsável por controlar o acesso e as transações de todos os clientes. A distribuição ocorre apenas no nível dos clientes. O ambiente suporta vários clientes distribuídos, porém, seu servidor é centralizado e único (BARGOUTHI et al., 1996).

O MARVEL possui um administrador que instancia o processo de *software* e define os dados para o mesmo. O usuário MARVEL, em contraste com o administrador, só utiliza o ambiente caso o processo esteja instanciado e os modelos de dados construídos (SOKOLSKY, KAISER, 1991).

2.2.8 MILOS

O MILOS é um sistema de suporte ao processo de desenvolvimento de *software* através da Internet. Ele é baseado em Web e possui controle centralizado e não distribuído com suporte a equipes de desenvolvimento virtuais (AVERSANO et al., 2004) (MAURER et al., 1999).

O ambiente integra modelagem de processos com planejamento e atuação. Sua *engine* flexível de workflow é refinada e suporta alteração no modelo de processo durante a execução de um projeto. A integração de ferramentas é acoplada por meio da utilização de aplicativos integráveis aos navegadores de Internet. O MILOS é implementado em Java utilizando o OODB GemStone/J 2.0 como servidor EJB que mantém o gerenciamento de transações e serviços de persistência (MAURER et al., 1999).

A arquitetura do MILOS é multi-camadas com uma abordagem cliente-servidor. Os clientes utilizam aplicações Web com Applets se comunicando com o servidor através de RMI.

A arquitetura do sistema é constituída de vários componentes interligados:

1. ***Resource Pool***: agente de gerenciamento de componentes. Permite representar a estrutura da companhia e organizações. É utilizado para agendamento de tarefas e sua associação com ferramentas;
2. ***Process Modeling***: Componente que gerencia a definição formal de processos. Isto inclui controle e especificação de fluxo de informação com pré e pós condições;
3. ***Project Plan Management***: Componente que suporta gerenciamento de projeto com planejamento e customização. O gerente pode adicionar datas para início e término do planejamento e associar agentes aos processos. Possui interoperabilidade com o MSProject;
4. ***Workflow Management***: O gerenciador de workflow é responsável por executar o projeto e gerenciar os produto. Ele gera uma lista de afazeres para os agentes e mantém controle sobre o estado atual do projeto. Sua engine é

capaz de reagir dinamicamente às mudanças de planejamento do projeto durante a sua execução sem interrompê-lo.

2.2.9 Odyssey

O ambiente Odyssey foi desenvolvido na linguagem de programação Java e está equipado com toda a infra-estrutura necessária à modelagem dos domínios e das aplicações que uma equipe precisa para executar suas atividades (TEIXEIRA, MURTA, WERNER, 2001-a) (TEIXEIRA, MURTA, WERNER, 2001-b).

Entre os requisitos do Odyssey, encontram-se: apoio à encenação de processos em equipes distribuídas, o incentivo à comunicação e à socialização, apoio à atividade individual e em pequenas equipes. O Odyssey tem por objetivo uma maior eficácia e eficiência no desenvolvimento de *software* e, sobretudo, uma maior satisfação dos indivíduos que participam desse processo (WERNER et al., 2002).

Ao estabelecer um processo definem-se, entre outras coisas, a seqüência de atividades, as ferramentas a serem utilizadas, os papéis dos desenvolvedores e os artefatos consumidos e produzidos. Estes artefatos são recuperados em repositórios de componentes disponíveis local ou remotamente, através de mediadores e ontologias capazes de integrar as informações armazenadas em repositórios distribuídos (WERNER et al., 2002).

O repositório do Odyssey é denominado Odyssey-VCS. A arquitetura do Odyssey-VCS, que foi implementado em Java para evitar a dependência dos VCS baseados em arquivos, é composta de três camadas: cliente, transporte e servidor. Esta arquitetura parte do pressuposto que as ferramentas de modelagem UML são capazes de gravar modelos UML utilizando XMI. O cliente Odyssey-VCS pode ser utilizado como um plugin ou como uma ferramenta (OLIVEIRA, MURTA, WERNER, 2005).

A camada de transporte é responsável por distribuir os modelos UML utilizando a Internet. A implementação desta camada utiliza Web Services como protocolo de transporte. Todavia, esta camada pode ser substituída por outros protocolos como WebDAV, RMI ou sockets. Antes de ser enviado o artefato é compactado para evitar o aumento de throughput da rede (OLIVEIRA, MURTA, WERNER, 2005).

O fato de o ambiente Odyssey ter sido construído na linguagem Java foi um fator decisivo na construção e integração do sistema LockED, o qual foi desenvolvido na linguagem Java e JSP (*Java Server Pages*). A tecnologia JSP se apresentou como uma vantagem considerável nesta implementação, uma vez que esta possibilita a construção de sistemas web totalmente compatíveis com a linguagem Java (TEIXEIRA, MURTA, WERNER, 2001-a) (TEIXEIRA, MURTA, WERNER, 2001-b).

A infra-estrutura Odyssey pode ser vista como um arcabouço onde modelos conceituais do domínio, modelos de projeto (arquiteturas de *software* específicas do domínio) e modelos implementacionais (*frameworks*) são especificados para um determinado domínio, disponibilizando componentes reutilizáveis neste domínio. Durante o processo de desenvolvimento de aplicações, estes componentes são refinados e adaptados ao contexto de um projeto específico (TEIXEIRA, MURTA, WERNER, 2001-a) (TEIXEIRA, MURTA, WERNER, 2001-b).

A Engine de Workflow do Odyssey, chamada Charon, foi construída para apoiar e coordenar desenvolvimento colaborativo (WERNER et al., 2003).

O projeto OdysseyShare objetiva a construção de um ambiente de desenvolvimento de *software* cooperativo com suporte a DBC (WERNER et al., 2003). No Odyssey, a construção de uma arquitetura de componentes ocorre a partir da ligação destes através de suas interfaces, sem um suporte específico a um estilo arquitetural que defina a arquitetura do domínio (BLOIS, BECKER, WERNER, 2004).

2.2.10 Oikos

O Oikos foi desenvolvido pela Universidade de Pisa por Montangero (BARTHELMESS, 2003). O projeto teve início em 1989 em um *framework* de um projeto de pesquisa em engenharia de *software* iniciado no *Italian National Research Council*. O objetivo do Oikos é facilitar a construção de PSEEs. Ele foi desenvolvido em Prolog e é executado em uma rede de estações de trabalho Sun (AMBRIOLA et al., 1993).

Em seu manifesto inicial, o Oikos foi descrito como um ambiente para especificar, desenvolver e implementar PSEEs. Outros objetivos eram a compreensão e a documentação de processos de *software* (AMBRIOLA, CONRADI, FUGGETTA, 1997).

Oikos é a palavra em grego antigo para casa, é também a raiz da palavra "eco" em "ecologia". Para seu grupo de pesquisa seu significado é "ambiente".

O Oikos possui um conjunto de serviços padrões. São eles (AMBRIOLA, CIANCARINI, MONTANGERO, 1990):

- ***Tool Kit Server*** – fornece um conjunto de ferramentas para manipular documentos.
- ***Service and Theory Server*** – Armazena as regras de negócio em Shared Prolog;
- ***History Server*** – oferece acesso restrito às informações da base de dados;
- ***User interface*** – GUI;
- ***Workspace Server*** – Serviço que permite trabalhar documentos localmente;
- ***DataBase server*** – responsável por armazer globalmente informações e documentos;
- ***Run Time Support*** – Acesso ao sistema operacional.

O Oikos oferece um conjunto rico de metáforas que manipulam diferentes níveis de cooperação: A mesa, o ambiente e o escritório. Uma mesma mesa pode ser ocupada por vários usuários, um ambiente pode ter várias mesas e um escritório vários ambientes (BARTHELMESS, 2003).

O processo de *software* no Oikos é uma coleção dinâmica de agentes cooperando via *black boards* hierárquicos. Cada agente é conectado a um *black board* e reage a presença ou ausência de atividades nestes, removendo atividades para outros black boards ou buscando ativiades de outros black boards (AMBRIOLA, CIANCARINI, MONTANGERO, 1990).

O Oikos utiliza a estrutura de árvores e sub-árvores dos *black boards* para ocultar a informação utilizando as especificações dos serviços. A especificação de um serviço é a definição de sua interface escondendo sua implementação e especificando o protocolo de interação como uma hierarquia de serviços. Com excessão do Oikos *Runtime Support* que é único, qualquer outro serviço pode ser ativado várias vezes e integrado a diferentes *black boards* (AMBRIOLA, CIANCARINI, MONTANGERO, 1990).

O Oikos não possui um suporte específico para a cooperação síncrona. A cooperação assíncrona é suportada através do controle do ambiente e da interação das ferramentas utilizadas pelos desenvolvedores. Para o controle de concorrência, os usuários com o mesmo perfil cooperam dentro do mesmo processo através da troca de mensagens e da sincronização de acesso a documentos (AMBRIOLA, CONRADI, FUGGETTA, 1997).

O kernel do Oikos é escrito em Shared Prolog e sua comunicação é feita usando sockets no sistema operacional UNIX (AMBRIOLA, CIANCARINI, MONTANGERO, 1990). Seu modelo é baseado em duas linguagens distintas: Limbo e Pate`, cobrindo a manutenção e especificação de requisitos e as fases de implementação (AMBRIOLA, CONRADI, FUGGETTA, 1997).

A arquitetura do Oikos é baseada na máquina virtual Expo 2.0 e a sua comunicação é baseada em um protocolo customizado e próprio. O OIKOS ainda utiliza uma camada de serviço para gerenciamento de ferramentas e documentos baseada em uma base de dados centralizada utilizando Salad (AMBRIOLA, CONRADI, FUGGETTA, 1997).

No OIKOS a base de dados é centralizada e toda requisição é serializada. A camada de serviços abstrai esta visão permitindo a criação de um conjunto de repositórios independentes. Cada repositório é controlado por um serviço que fornece operações para mover documentos do repositório para o *workspace* e vice versa. A base de dados fisicamente é dividida em duas partes: a descrição dos documentos que é mantida no Salad e os documentos que são armazenados como arquivos Unix em um sistema de arquivo distribuído (AMBRIOLA, CONRADI, FUGGETTA, 1997).

2.2.11 Prosoft

O ambiente Prosoft é resultado do esforço cooperativo de estudantes e pesquisadores do PPGC-UFRGS e da *Fakultät Informatik da Universität Stuttgart* (Alemanha), sob coordenação do Prof. Dr. Daltro José Nunes. O principal objetivo do grupo de pesquisa é a construção de um ambiente de desenvolvimento de *software* para apoiar o engenheiro de *software* desde a fase de análise do problema até a fase de construção do programa. Este apoio é feito por meio de suporte ao uso de métodos formais durante o desenvolvimento de *software* complexos. Ele é baseado no paradigma orientado a atividades utilizando-se da estratégia data-driven de desenvolvimento de *software* para descrever um processo como um conjunto de atividades inter-relacionadas (REIS, LIMA REIS, NUNES, 2001) (FREITAS, 2005) (LIMA REIS, 1998).

Os componentes do Prosoft são os ATOs - Ambientes de Tratamento de Objetos. Todo ATO especifica algebricamente um tipo abstrato de dados e é composto, essencialmente, de uma classe e de um conjunto de especificação semântica de operações que atuam sobre os objetos dessa classe. Além disso, a ICS - Interface de Comunicação de Sistema, provê o mecanismo básico de comunicação dos ATOs PROSOFT (LIMA REIS, REIS, NUNES, 1998).

As ferramentas do Prosoft visam auxiliar o desenvolvedor de *software* desde a especificação de requisitos até a implementação do sistema. Além disto o ambiente possibilita a integração de novas ferramentas. O paradigma de construção de ferramentas no Prosoft foi influenciado pelos itens a seguir (LIMA REIS, 1998):

- A estratégia *data-driven*, que determina a definição das estruturas de dados em primeiro lugar e depois as operações;

- O conceito de modelos, onde a solução de um problema é o modelo de alguma teoria;
- O conceito de tipo abstrato de dados, onde tipos complexos podem ser definidos a partir da instanciação de outros mais simples;
- O paradigma de orientação a objetos;

O ambiente Prosoft foi implementado em diferentes versões, até chegar à sua versão atual, denominada Prosoft-Java implementada em Java e beneficiando-se dos mecanismos de distribuição e portabilidade fornecidos por esta linguagem (REIS, 2001). A primeira versão, monousuária, foi desenvolvida em Solaris-Pascal. Posteriormente, foi desenvolvido o Prosoft-Distribuído, utilizando as linguagens Pascal e C. A implementação mais recente do ambiente, desenvolvida em Java, resultou em um ambiente portável, distribuído e cooperativo que integra ferramentas CASE para auxiliar diferentes fases do processo de *software* (FREITAS, 2005).

De uma forma geral, as características que distinguem o Prosoft Distribuído do Prosoft tradicional são (LIMA REIS, REIS, NUNES, 1998):

- O ambiente foi projetado de acordo com um modelo cliente/servidor. A implementação da ICS foi reprojetada (baseada no mecanismo RPC) para permitir uma comunicação transparente entre clientes e servidores;
- ATOs específicos podem ser ativados por demanda, de acordo com as necessidades dos usuários. Os servidores que processam as requisições são chamados de ATO-Server e seus serviços podem ser compartilhados simultaneamente por vários usuários;
- O usuário pode configurar a distribuição do sistema de forma flexível, isto é, a adição ou remoção de ATOs da sua seção é facilitada.

Recentemente o ambiente Prosoft vem sendo utilizado para experimentação e validação de diferentes técnicas, ferramentas e metodologias para apoiar de forma integrada os aspectos de interação cooperativa de profissionais no desenvolvimento de produtos de alta complexidade (REIS, 2001) (FREITAS, 2005).

2.2.12 PROSYT

O PROSYT (*PROcess Support sYstem capable of Tolerating deviations*) foi desenvolvido no *Politecnico di Milano* por Cugola baseado em sua experiência prévia com o SPADE e SENTINEL (BARTHELMESS, 2003).

O PROSYT é um ambiente de engenharia de *software* centrado em processo baseado em artefatos que integra um *middleware* para infraestrutura de comunicação baseada em eventos distribuídos que oferece mobilidade de código para atender usuários nômades. O PROSYT é desenvolvido em Java com sua infraestrutura de comunicação baseada no *framework* JEDI utilizando RMI em seu canal de comunicação (BARTHELMESS, 2003) (CUGOLA, GHEZZI, 1998) (FREITAS, MAIA, NUNES, 2004-b) (CUGOLA, GHEZZI, 1999).

Esta comunicação simplifica a integração de ferramentas externas ao ambiente e facilita a reconfiguração do sistema. Novos componentes podem ser adicionados, componentes existentes podem ser removidos e componentes podem ser alterados sem que haja uma mudança no ambiente como um todo (CUGOLA, GHEZZI, 1998).

Cada artefato produzido durante o processo é uma instância de um tipo de artefato que descreve sua estrutura interna e seu comportamento. Cada tipo de artefato é caracterizado por um conjunto de atributos cujos valores definem o estado interno de suas instância, um conjunto de operações exportadas que podem ser invocadas pelo usuário através da instância do tipo do artefato, e um conjunto de operações exportadas que são automaticamente executadas quando determinado evento acontece e são utilizadas para automatizar o processo para reagir a alterações no estado das ferramentas de controle do ambiente (CUGOLA, GHEZZI, 1999) (FREITAS, MAIA, NUNES, 2004).

Todo o *workflow* deste modelo é baseado nos artefatos e nas operações que os mesmos podem ter (AVERSANO et al., 2004). Além disto, o PROSYT também é dividido em aplicações e gerenciadores de processo (CUGOLA, GHEZZI, 1999). Os componentes que constituem o ambiente PROSYT são:

- Uma ***engine* de processo** para cada repositório e uma engine do processo que define o tipo da instância do projeto que representa o processo de negócios corrente;
- Um **navegador de projeto** para cada usuário, utilizado para navegar entre as entidades (artefatos, diretórios, repositórios) que compõe o processo definido, invocando as operações que eles exportam;
- Um **gerenciador de *login*** com função de limitar o acesso ao ambiente. Este componente gerencia toda a informação sobre os agentes humanos envolvidos no processo;
- A **ferramenta administrativa** usada pelo gerenciador de processo para alterar as políticas de execução e para adicionar e remover usuários;
- As **ferramentas** invocadas pelo sistema para executar tarefas específicas do processo como editores e compiladores;
- Uma **ferramenta de monitoramento** com função de monitorar a execução através da identificação de conflitos em conjunto com uma ferramenta que analisa o resultado deste monitoramento.

O PROSYT possui duas características que o distingue (CUGOLA, GHEZZI, 1998):

- Ele permite que os desenvolvedores lidem com situações não esperadas pela alteração do processo e mudança em seu controle;
- Ele suporta distribuição geográfica das equipes de trabalho através de uma infraestrutura cooperativa.

2.2.13 Comparação entre os ambientes estudados

Tendo este trabalho o objetivo de definir um *framework* para a infra-estrutura de ADDS, algumas comparações quanto às características de infra-estrutura puderam ser feitas analisando os trabalhos relacionados, conforme segue abaixo.

Tabela 1 – Linguagens de programação dos PSEE estudados

PSEE	Perl	Java	Shared Prolog	C / Prolog	C++	ADA
Adele/Tempo						
CAGIS						
Endeavors						
EPOS						
ExPSEE						
Genesis						
Marvel						
Milos						
Odyssey						
Oikos						
Prosoft						
Prosyt						

Por meio da Tabela 1 é possível afirmar que a maior parte dos PSEEs relacionados neste trabalho baseia-se no conceito de orientação a Objetos (OO) e buscam linguagens de programação que tenha suporte a uma maior interoperabilidade em relação às plataformas de *Hardware* e Sistemas operacionais. Conforme aparece nos trabalhos relacionados, a opção por linguagens orientadas a objetos ocorre não apenas por questões de implementação, mas também, por decisões de projeto. Isto porque os ambientes que optam por linguagens OO buscam nos objetos a abstração de seus componentes e entidades envolvidas no processo de um ADS.

Tabela 2 – Protocolo de Comunicação

	JEDI	CORBA	SOAP	RPC	Java RMI	Unix sockets	HTTP
Adele/Tempo							
CAGIS							Cgi
Endeavors							Servlets
EPOS							
ExPSEE							

Genesis							
Marvel							Servlets
Milos							Applets/EJB
Odyssey							
Oikos							
Prosoft							
Prosyt							

Quando tratamos comunicação dentro de um ambiente de *software*, a mesma pode ser Síncrona ou assíncrona conforme ilustra a Tabela 2. A opção por protocolos como CORBA, RPC, Sockets e RMI remetem a uma opção por comunicação síncrona enquanto que protocolos de Internet como o HTTP, o *framework* JEDI e Web Services (Protocolo SOAP) remetem à comunicação assíncrona com troca de mensagens. Uma simplificação desta tabela seria como a que segue:

Tabela 3 – Tipos de comunicação

	Comunicação Síncrona	**Comunicação Assíncrona**
Adele		
CAGIS		
Endeavors		
EPOS		
ExPSEE		
Genesis		
Marvel		
Milos		
Odyssey		
Oikos		
Prosoft		
Prosyt		

Com estes dados em mãos pode-se notar que não há um modelo de comunicação padrão para ADSs e que alguns ambientes optam por ter comunicação síncrona e assíncrona de maneira a atender necessidades distintas do ambiente. A comunicação síncrona é utilizada para pontos críticos de comunicação enquanto a comunicação assíncrona é utilizada para dados que não comprometem o funcionamento do ambiente. Esta divisão ocorre pelo fato de haver um custo maior para a implementação da comunicação síncrona.

Tabela 4 – Sistema de Armazenamento de dados (meta-dados / Artefatos)

Adele/Tempo	Adele (Repositório próprio) + Sistema de arquivo Unix
CAGIS	Arquivos XML
Endeavors	Arquivos ASCII
EPOS	SGBD próprio (EPOS-DB baseado no Ingres) + Sistema de arquivos Unix
ExPSEE	SGBDOO ObjectStore
Genesis	OSCAR (CVS)
Marvel	Sistema de arquivos Unix
Milos	OODB GemStone/J 2.0
Odyssey	Odyssey-VCS
Oikos	Sistema de Arquivos Unix
Prosoft	**Não encontrado**
Prosyt	Sistema de arquivo

Em relação ao armazenamento de seus dados, pode-se notar pela Tabela 4, que a maior parte dos ambientes utiliza ferramentas externas ao ambiente como SGBDs ou o próprio sistema de arquivos. Poucos são os ambientes que possuem uma ferramenta de armazenamento interna ao ambiente e de desenvolvimento próprio.

2.2.14 Requisitos desejáveis de ADDS

Por meio da observação das características de vários ambientes com suporte ao processo de *software*, é possível obter uma visão geral sobre os requisitos necessários que um ADDS deve prover para tornar-se orientado a processos (LIMA REIS, 1998). Partindo destas observações e do trabalho de (BEN-SHAUL, KAISER, 1998), é possível estabelecer uma intersecção e definir os requisitos para ADDS, conforme ilustra a Tabela 5 – Requisitos desejáveis para um ADDS.

Tabela 5 – Requisitos desejáveis para um ADDS

Infra-estrutura de comunicação	Para que os membros das equipes possam trocar informações referentes a seus projetos;
Definição de processo	Os ambientes devem prover suporte à modelagem e a execução de processos;
Integração e interoperabilidade	Ferramentas com exportação de dados à gerência de processos e desenvolvimento em um formato comum e que permita a integração com outras ferramentas;
Autonomia local	Cada instância local do PSEE deve funcionar de forma autônoma e independente das demais durante a definição e a execução de atividades que envolvam apenas recursos e dados locais. Durante a modelagem e a execução de atividades, puramente locais, não será necessária a comunicação com outras instâncias.
Percepção	Cada instância do PSEE deve ter conhecimento de quais são as demais instâncias com os quais ela interage;
Configuração dinâmica	Tanto as configurações de projeto, de processo, dos membros da equipe, das alterações na infraestrutura, dos locais envolvidos quanto a alteração na configuração de *workspace* devem ser permitidas;
Persistência	A execução de processos de *software* costuma ter uma duração longa. Em virtude disso, o estado e os artefatos do projeto devem ser armazenados de forma persistente;
Semântica transacional	Uma atividade de um processo de *software* pode depender de outras atividades durante o projeto. A falha de uma atividade pode, portanto, ter efeito sobre a execução de outras. Mecanismos para recuperação, em caso de falhas, devem ser fornecidos, para que os resultados da execução de uma atividade sejam propagados.
Gerência de Acesso	O acesso ao ambiente deve ser controlado utilizando-se de papéis. Os papéis desempenhados por um usuário devem limitar o acesso deste ao ambiente.
Gerência de Visão	O ambiente deve prover interação com o usuário independente da forma do controle do ADS ser centrada em artefato, ferramentas ou agenda.
Gerenciamento de Recursos	O gerenciamento de recursos do ambiente deve permitir a distribuição de ferramentas e, também, o controle dos membros de equipes (recursos humanos).

2.3 Conclusão

Este capítulo apresentou os principais conceitos envolvidos neste trabalho como ambientes de desenvolvimento de *software*, Desenvolvimento global de *software*, arquitetura em camadas, sistemas distribuídos, *frameworks* e componentes. A apresentação destes conceitos foi feita para esclarecer o posicionamento do trabalho em relação a estes conceitos.

Também foram apresentados os trabalhos relacionados. Vários ADSs foram analisados quanto à sua infra-estrutura e uma comparação entre os mesmos foi feita com o intuito de encontrar uma intersecção entre os ADS existentes.

Há vários outros ADS disponíveis na literatura, porém sem o aprofundamento necessário para um estudo comparativo. Estes outros ADS são apresentados no ANEXO I – Outros ADSs encontrados na literatura.

3 O *Framework* FRADE

Este capítulo apresenta o *framework* desenvolvido nesta dissertação. Será apresentada a arquitetura deste *framework*, sua divisão lógica em camadas, os componentes que compõe cada camada e a integração destas camadas através da interface de seus componentes.

Feita a especificação dos componentes de cada camada, na conclusão deste capítulo é apresentada a interdependência destes componentes permitindo assim uma visão mais ampla da implementação do FRADE.

3.1 Visão Geral

O FRADE possui uma divisão lógica em camadas. Um dos motivos de se utilizar uma arquitetura em camadas é poder dividir entre as camadas do sistema suas funcionalidades em diferentes níveis de abstração. A opção pela arquitetura em camadas também foi feita devido à facilidade de manutenção.

Além da arquitetura em camadas, este *framework* utiliza o desenvolvimento baseado em componentes. Com isto cria-se uma coleção de ferramentas que permita variações e combinações com vários *front-ends*. Desta maneira, o sistema é descrito de acordo com seus serviços e não de suas implementações (CHAMBERS, LANG, 1999).

Seguiu-se a definição de CUGOLA e GHEZZI (1999) para um ADS típico que consiste em uma *engine* de processo, que interpreta o processo como um modelo e fornece condições para a execução das ferramentas durante o processo, e uma interface gráfica utilizada pelos usuários para interagir com o ambiente. Os componentes interagem em uma arquitetura cliente-servidor aonde a engine do processo age como o servidor e o front-end gráfico dos clientes agem como cliente. Neste trabalho a Linguagem de processo e o modelo de processo não serão abordados.

As camadas lógicas do *framework* são ilustradas na Figura 2. Esta figura apresenta o ambiente sendo composto por duas estações de trabalhos. Caso o *framework* esteja inteiramente instanciado em ambas as estações, estas podem operar de maneira independente. Além de prover estrutura para esta operação independente, este *framework* trabalha sobre um cenário aonde cada uma destas estações possui alguns dos serviços propostos pelo *framework* e o ambiente se torna dependente de ambas as estações, ou seja, para o completo funcionamento do ambiente estas estações são inter-dependentes. Há no *framework* um serviço que controla a distribuição dos demais serviços no ambiente que será melhor explicado na seção 3.4.5 deste trabalho.

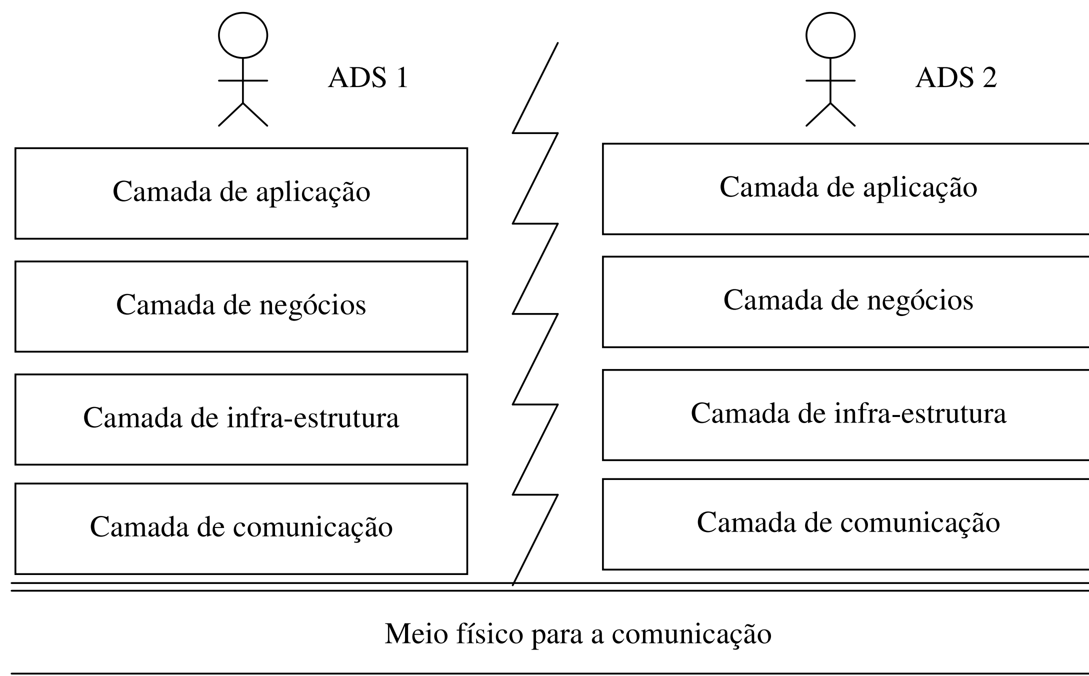

Figura 2 – Camadas lógicas do *Framework*

A camada superior, chamada de **camada de aplicação**, possui facilidades para a implementação de aplicativos e ferramentas no domínio de Ambientes de desenvolvimento de *software*. Esta é a camada mais próxima ao usuário do ambiente e, é por meio desta camada que o usuário consegue interagir com o ambiente. Na **camada de aplicação** há componentes gráficos para o desenvolvimento de ferramentas de maneira que estas possam ser adicionadas dinamicamente ao ambiente. Esta camada será melhor explicada na seção 3.3 deste trabalho.

A **camada de negócios** do ambiente mapeia classes – entidades que representam o meta-modelo de processo para o *framework*. Seguindo a proposta inicial do DiSEN, estas classes são agrupadas em gerenciadores, de maneira a criar componentes independentes, que representem o estado do processo de *software* em suas mais diversas circunstâncias. Atuando como os dados da *engine* de processo, as classes desta camada permitem dividir as questões tangentes ao gerenciamento do desenvolvimento em O QUE, COMO e PORQUE (CONRADI et al., 1991).

Os componentes da **camada de negócios** do *framework* são representados na forma de gerenciadores. No contexto deste trabalho, Gerenciadores são abstrações dos requisitos funcionais de ADDSs encontrados no estudo dos trabalhos relacionados do capítulo anterior. Por se tratarem de abstrações, os mesmos contêm toda a definição de seus atributos e a definição de sua funcionalidade. Seguindo o DBC, os gerenciadores serão implementados na forma de componentes de *software*.

Cada gerenciador possui um conjunto de classes que o representa, e utiliza um conjunto dos suportes da camada inferior para cumprir suas funções. A intenção desta camada é levar para a **camada de aplicação** as funcionalidades dos suportes da camada de infra-estrutura associada às classes de negócios aqui representadas para permitir a criação de

ferramentas que atuem na *engine* do processo. A **camada de negócios** e seus gerenciadores serão melhor explicados na seção 3.4 deste trabalho.

A **camada de infra-estrutura** possui uma divisão entre serviços e suportes. Segundo POPOVICH (1992), o servidor é uma escolha natural para a reutilização de componentes. Esta camada terá dois tipos de componentes: suportes e serviços. Partindo do conceito da arquitetura cliente / servidor, os suportes serão utilizados como clientes para os serviços. Por meio dos suportes é possível ativar serviços locais ou serviços remotos. É função do suporte ativar um serviço, independente de sua localização, e disponibilizá-lo, localmente, de maneira transparente ao usuário. A ativação de um serviço remoto é feita através de seu suporte local. Todo serviço local mapeado dentro do ambiente pode ser visto por outra estação de trabalho como um serviço remoto.

A ativação e utilização de um serviço remoto pelo suporte são feitas através da **camada de comunicação**. Os suportes conhecem apenas as interfaces dos serviços e há um serviço em especial que é responsável por localizar os diversos serviços do ambiente. A comunicação entre suportes e serviços é feita por meio de requisição e respostas. A alteração da implementação de um objeto ou, sua realocação, não devem causar efeitos aos clientes (DOGAC, DENGI, SZU, 1998). A intenção de ativar serviços remotos no ambiente é possibilitar o acesso a serviços em outras máquinas (CHAMBERS, LANG, 1999). A visão de caso de uso do suporte local é apresentada na Figura 3.

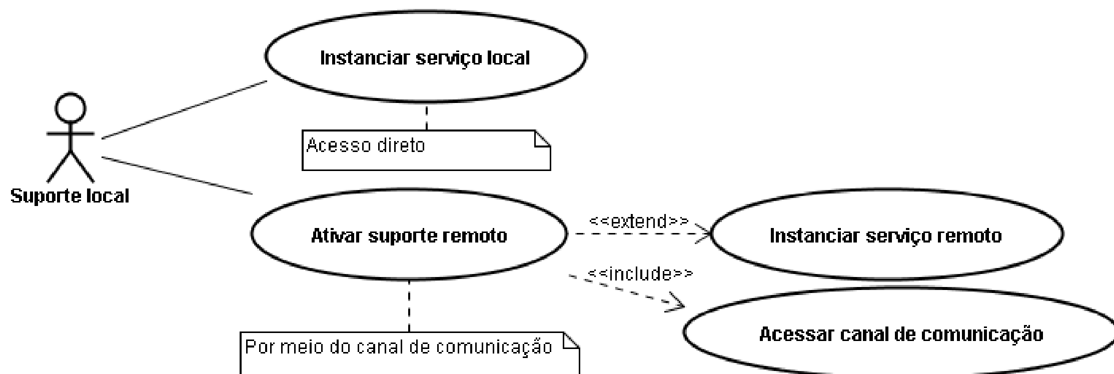

Figura 3 – Caso de uso para os suportes locais

No caso de serviços locais, a ativação dos mesmos pelos suportes é feita diretamente no *workspace* do usuário. Estes serviços locais são definidos para que seja possível ao usuário trabalhar desconectado. A idéia de oferecer serviços locais é aumentar o desempenho de funções críticas nos pontos aonde erros, ou atrasos, podem comprometer a segurança de sistemas ou causar prejuízos econômicos (MONTRESOR, DAVOLI, BABAOĞLU, 2001).

Serviços locais devem replicar dados remotos. O conjunto de serviços locais compõe o *workspace* para o *framework*. A replicação é a principal técnica para o desenvolvimento de serviços dependentes. Ao executar o mesmo serviço em várias réplicas distribuídas pode-se aumentar a garantia de disponibilidade e corretude (MONTRESOR, DAVOLI, BABAOĞLU, 2001). A sincronização do serviço local com o serviço remoto é de responsabilidade da camada de infra-estrutura (DOGAC, DENGI, SZU, 1998).

Como este *framework* se propõe a implementar ambientes distribuídos, a escalabilidade, transparência e tolerância à falhas são aqui incorporados à camada de infra-estrutura. Um serviço pode ser executado em várias máquinas que executarão seus processos de maneira independente e ele será visto pelo ambiente como um único serviço. A responsabilidade de manter a consistência dos dados, gerenciar as transações e a

sincronização de serviços distribuídos será da camada de infra-estrutura. A **camada de infra-estrutura** será detalhada ainda neste trabalho na seção 3.5.

A **camada de comunicação** é a camada mais baixa do ambiente, na estrutura do *framework*. Ela implementa o *middleware* do *framework*. Esta camada permite a comunicação entre as máquinas que integram o ambiente como um todo. A implementação da **camada de comunicação** permite que a mesma seja síncrona ou assíncrona. A comunicação síncrona é bloqueante, ou seja, o aplicativo que enviou uma mensagem à aplicação remota aguarda o retorno da resposta para continuar sua execução. A comunicação assíncrona é não-bloqueante, sendo implementada apenas com troca de mensagens entre aplicações.

A **camada de comunicação** estabelece ainda alguns tipos de mensagens que podem ser trocadas no ambiente e permite que serviços e aplicações se registrem no ambiente para estabelecer sua comunicação nos dois sentidos: serviço a serviço, ferramenta a ferramenta e serviço a ferramenta. A **camada de comunicação** será detalhada na seção 3.6 deste trabalho.

3.2 Contexto

O contexto onde este *framework* está inserido é o ambiente DiSEN. A arquitetura inicial deste ambiente, conforme proposto em (PASCUTTI, 2002), é apresentada na Figura 4.

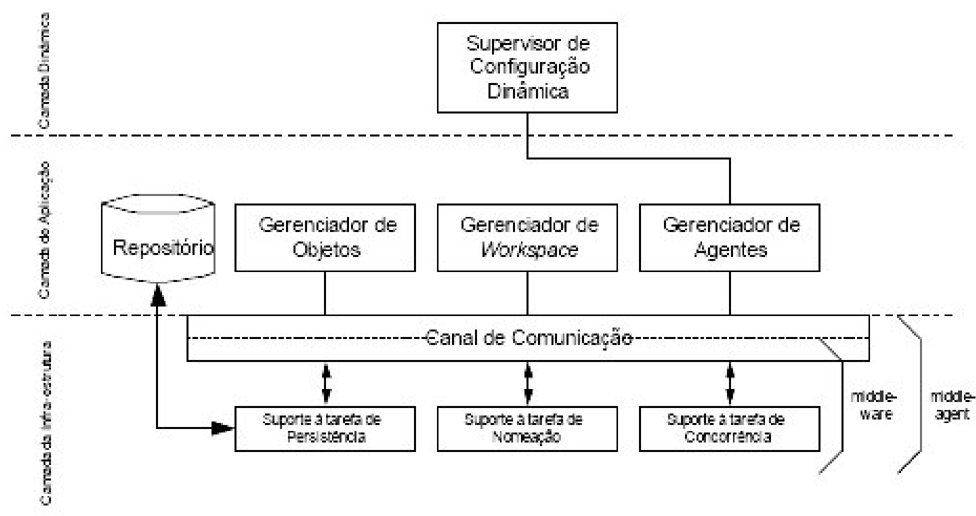

Figura 4 – Arquitetura inicial proposta para o DiSEN (PASCUTTI, 2002)

A arquitetura em camadas do ambiente proposto continua mantida neste *framework*, porém a visão da arquitetura no *framework* FRADE parte de uma única estação de trabalho para alcançar a distribuição entre as várias estações que podem compôr o ambiente, dividindo a infra-estrutura do ADS entre estas várias estações.

O repositório da arquitetura inicial foi dividido no FRADE em três tipos de repositórios: repositório de ferramentas, repositório de meta-dados e repositório de artefatos. Cada um destes repositórios é aqui apresentado na forma de um serviço. Como há a possibilidade de acessar serviços remotos, no FRADE não há a necessidade de que todas as

estações possuam os três repositórios, mas que exista no ambiente ao menos um repositório de cada tipo, para que todas funcionalidades do ambiente possam ser executadas corretamente.

O suporte à configuração dinâmica da arquitetura inicial foi modelado neste *framework* por meio de vários suportes/serviços de maneira a dividir os vários tipos de configurações que podem ser dinâmicas em um ADS como, por exemplo, configurações de ferramentas, configurações de serviços, configurações de papéis / perfis de usuários ou configurações de acesso.

O gerenciador de *workspace* da arquitetura inicial limita o *workspace* a um repositório de artefatos local. O gerenciador de *workspace* do FRADE utiliza o conceito de *workspace* que consta no trabalho de POZZA (2005) e estende o *workspace* incluindo ao mesmo os vários dados que o espaço de trabalho de um usuário pode possuir. Entre estes dados estão aqueles necessários para a conexão remota, informações da *engine* de processos, atualizações, ferramentas, e o próprio espaço de trabalho com suporte a *awareness*.

O gerenciador de Objetos da arquitetura inicial também foi redesenhado de maneira a atender a especificação de todos os objetos encontrados no modelo de negócios do ambiente. A alteração do gerenciador de Objetos está descrita na seção 3.4.

3.3 Camada de Aplicação

A Camada de Aplicação é a camada de abstração mais alta no *framework*, aonde o usuário irá interagir junto ao ambiente, conforme ilustra a Figura 5.

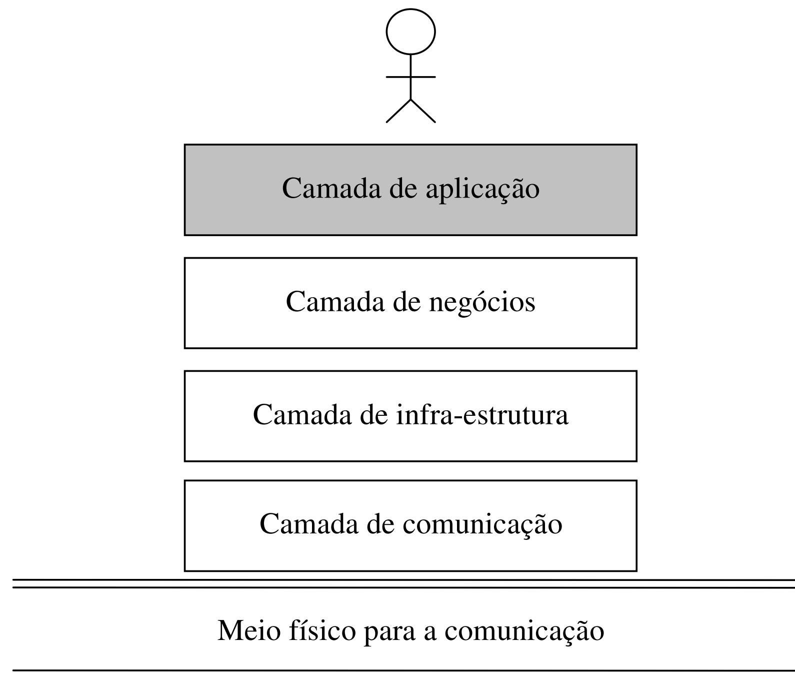

Figura 5 – Posicionamento da Camada de Aplicação

Segundo TAYLOR et al. (1988), a interface de usuário de um ambiente deve interagir com os usuários e com as demais ferramentas, de maneira a se tornar o mais uniforme possível. Por causa da variedade de *software* que pode vir a integrar o ambiente, este deve

apoiar a interface de interação com usuário de forma gráfica ou textual. Apesar de parecer ultrapassado, o conceito de interfaces textuais para usuários ainda é muito utilizado para estações que operam somente como servidores dentro de um ambiente.

O conceito de camada de aplicação é reforçado por CHAMBERS e LANG (1999). Segundo estes autores o ambiente deve apoiar o desenvolvimento de GUIs para diferente tipos de usuários e permitir a integração de outras aplicações. No trabalho de BARGHOUTI e KAISER (1999) a integração de outras ferramentas é chamado, notadamente, de evolução do ambiente.

Além de múltiplas interfaces para clientes e servidores, a Interface do usuário deve ainda se apresentar de maneira distinta para usuários com funções distintas no ambiente (TAYLOR et al., 1988). Multiplas visões devem permitir que vários usuários com funções distintas e, em diferentes fases do ciclo de vida do processo, possam dispor de diferentes funcionalidades e ferramentas para que os mesmos cumpram seus papéis (BARGHOUTI, KAISER, 1999).

Um exemplo desta visão distinta do ambiente é apresentado por HEIMANN et al. (1996), que lembram que, enquanto um gerente de processo pode visualizar toda a instanciação de um processo e a utilização do mesmo por um projeto, um desenvolvedor, provavelmente, deverá visualizar apenas sua agenda.

Completando estes conceitos, BARGOUTHI et al. (1996) lembram que a interface multi-usuário deve apoiar múltiplos clientes e programas e deve ser, na medida do possível, distribuída.

No FRADE, a integração de ferramentas é feita por meio do gerenciador de ferramentas. Este gerenciador permite instalar novas ferramentas e integrá-las ao ambiente.

Não é escopo deste trabalho tratar o desenvolvimento de ferramentas. Para manter a GUI das ferramentas padronizadas quanto aos seus componentes gráficos e comportamentos, várias classes de suporte ao desenvolvimento foram integradas ao FRADE. Entre elas estão:

- **Manipulador de exceções**: Classe abstrata de exceção que apresenta o erro ao usuário. A implementação desta classe permite que mensagens padrões possam ser enviadas ao usuário no tratamento de erros. Isto inclui caixas de diálogo ou geração de logs para serviços;
- **Pacote de componentes gráficos**: Um pacote baseado em Swing[3] foi implementado de maneira que a apresentação gráfica de todas as aplicações possa ser configurada de maneira uniforme. Estes componentes facilitam a implementação e incluem várias implementações, que se comunicam com o gerenciador de objetos, entre elas tela de *login* padrão, lista de usuários online, entre outras. Estes componentes não são aplicações, mas componentes que podem ser usados para compor aplicações de maneira mais simples;
- **Aplicação principal**: A aplicação principal, como é chamada, representa a interface padrão do ambiente. Ela pode ser gráfica ou textual. No caso de esta interface ser gráfica, a mesma possui a capacidade de ter adicionado a ela itens de menu, botões em sua barra de ferramentas, Frames e Painéis. A integração

[3] Swing é um pacote de ferramentas para GUI da API padrão Java que é composto de vários widgets como botões, painéis, caixas de texto e tabelas.

de novas aplicações pelo gerenciador de ferramentas pode ser vista de maneira simplificada como a adição destas aplicações a aplicação principal;

A interface desta camada inclui o acesso direto a todos estes componentes pelas ferramentas que poderão integrar o ambiente. Além dos componentes para interação com o usuário, esta camada possui também a responsabilidade de fornecer às ferramentas acesso à camada de negócios do ambiente, constituída pelo Gerenciador de objetos. Para isto, é definida uma interface para a Camada de Aplicação conforme ilustra a Figura 6.

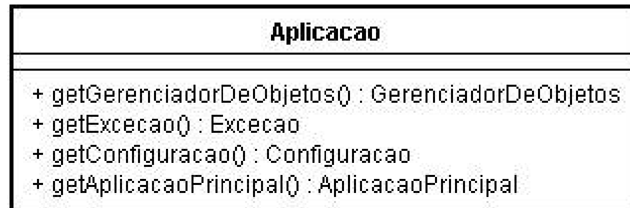

Figura 6 – Interface da Camada de Aplicação

Os tipos retornados pela Camada de Aplicação baseiam-se em Interfaces dos componentes da mesma. O FRADE inclui uma implementação para estas Interfaces e uma fábrica para a instanciação da implementação instalada em cada estação de trabalho. A fábrica que constrói a camada de aplicação baseia-se em um arquivo XML, que define qual implementação será utilizada para cada componente desta camada, conforme ilustra a Figura 7.

```
<config>
    <mapping key="disen.aplicacao.abstract.Excecao"
target="disen.aplicacao.LogExcecao" />
    <mapping key="disen.negocio.abstract.GerenciadorDeObjetos"
target="disen.negocio.GerenciadorDeObjetos" />
    <mapping key="disen.aplicacao.abstract.Configuracao"
target="disen.aplicacao.Configuracao" />
    <mapping key="disen.aplicacao.abstract.AplicacaoPrincipal"
target="disen.aplicacao.AplicacaoPrincipalGUI" />
</config>
```

Figura 7 – Arquivo XML para configuração da Camada de Aplicação

No arquivo exemplo, a Aplicação Principal utiliza **GUI** para a interface com o usuário e arquivo de **Log** para o retorno das Exceções. Para alterar esta característica do ambiente é necessário instalar outra aplicação principal e alterar o arquivo XML, dizendo que outra classe implementa esta Interface. A definição destas Interfaces é apresentada na Figura 8.

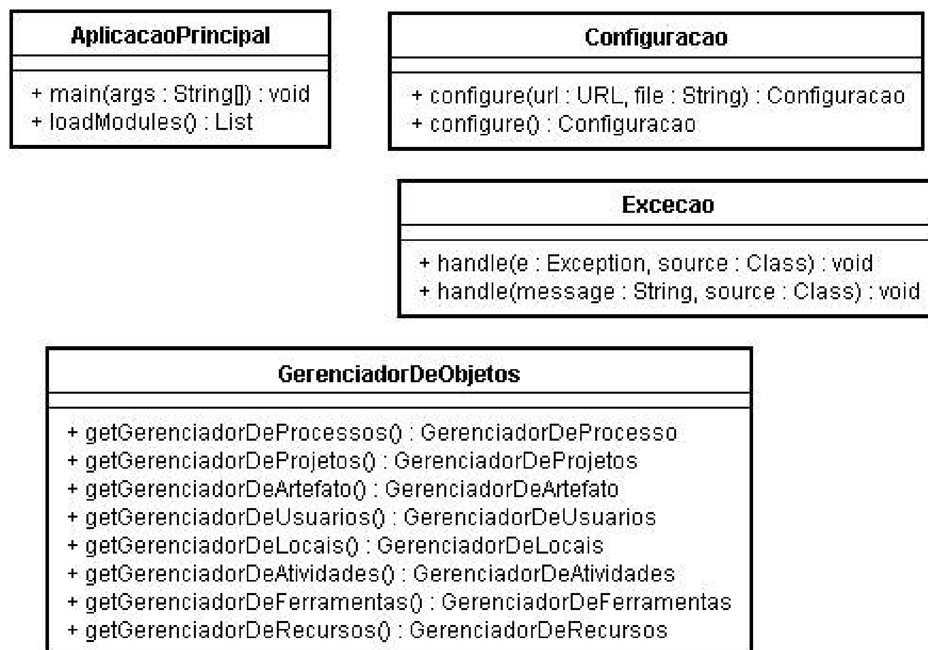

Figura 8 – Interfaces dos componentes da camada de Aplicação

A Aplicação Principal será chamada através de seu método main. Seu método loadModules possui a função de carregar as ferramentas que serão iniciadas com a GUI. A Classe Configuração será responsável por carregar esta configuração de ferramentas. O gerenciador de ferramentas cria esta configuração no momento da instalação / desinstalação de uma ferramenta. Para isto, um arquivo de configuração da ferramenta, chamado descritor XML, será enviado junto com o pacote de sua instalação. O conteúdo deste arquivo está descrito na seção 3.4.7.

A interface do Gerenciador de Objetos é a interface da camada de negócios e será definida na próxima seção.

3.4 Camada de Negócios

Segundo LEWANDOWSKI (1998), objetos de negócio são componentes usados para representar objetos chaves ou processos de um sistema real, em determinado domínio de aplicação.

O agrupamento dos objetos de negócio deste *framework* será chamado de Gerenciador de Objetos ou *Core*. O *core* de um processo deve permitir que o mesmo seja visualizado de uma maneira simples pela *engine* do processo (CONRADI, LIU, 1995).

O *core* mínimo de um processo deve prover suporte para seis elementos de processo: atividades, artefatos, papéis, humanos, ferramentas e meta processo (CONRADI, LIU, 1995).

HOLDER, BEN-SHAUL e GAZIT (1999) lembram ainda que a API *Core* deve fornecer vários serviços para as aplicações, incluindo a ativação inicial do *Core*, manipulação dos objetos, movimentação, nomeação, instanciação remota de objetos e monitoração.

A camada de negócios do *framework* tem por objetivo criar entidades encapsuladas que representem todos os passos do processo de *software* do ADS e mapeiem as

funcionalidades da infra-estrutura de maneira a conter os atributos de negócio que representem os meta-dados do ambiente e os métodos que representam as funcionalidades do ambiente (BELKHATIR, MELO, 1993). Como ilustra a Figura 9, esta camada se posiciona entre as de Aplicação e a de Infra-estrutura.

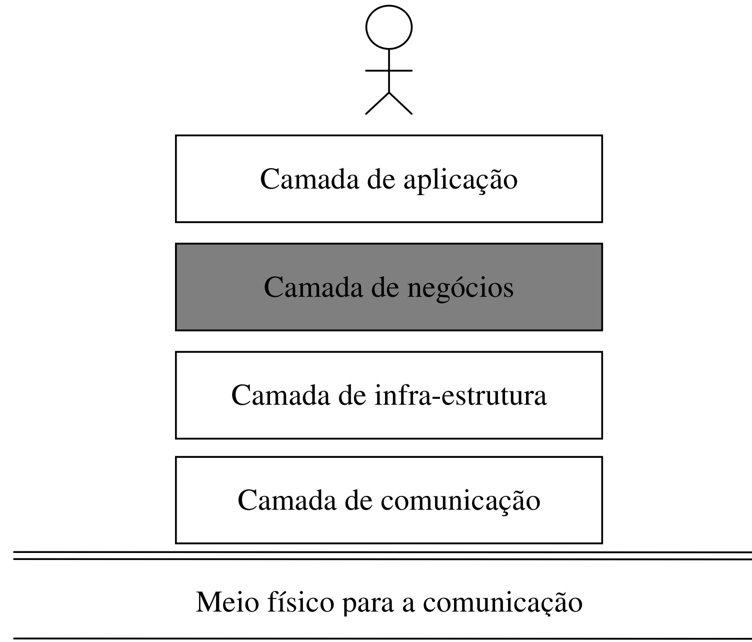

Figura 9 – Posicionamento da camada de negócios

Além disto, os objetos gerenciados pelo ambiente devem ser associados a identificadores que indicam um conjunto de propriedades. Estas propriedades podem ser consultadas para permitir a seleção de objetos pelo usuário. Algumas das propriedades que devem ser armazenadas para cada objeto são (REIS, 1998):

- Nome da ferramenta que manipula o objeto;
- Informação sobre a criação do objeto (quando e quem criou);
- Identificação dos usuários que podem manipular o objeto em um instante específico;
- Uma descrição textual sobre o objeto.

O conjunto das entidades do *framework* é chamado de gerenciador de objetos. Segundo TAYLOR et al. (1988), o gerenciador de objetos deve ser capaz de efetivamente armazenar e retornar objetos para um conjunto amplo de ferramentas. Os objetos de *software* podem ser estruturas de dados internas como árvores, tabelas simbólicas, grafos de sintaxe abstrata ou produtos externos como códigos fonte, planos de teste e projetos. Os objetos de *software* podem ter tamanhos variados e serem armazenados por pouco ou muito tempo. Além disto, os objetos podem ser alterados conforme a flexibilidade e a extensibilidade do ambiente.

O Gerenciador de Objetos possui o modelo do processo de *software*. Um processo de *software* pode ser visto como o conjunto de atividades, métodos, práticas e transformações que guiam pessoas na produção de *software*. Um processo eficaz deve considerar as relações entre as atividades, os artefatos produzidos no desenvolvimento, as ferramentas e os

procedimentos necessários e a habilidade, o treinamento e a motivação do pessoal envolvido. Neste contexto, os seguintes conceitos são importantes (MIAN, NATALI, FALBO, 2001):

- **Atividade**: É uma tarefa ou trabalho a ser realizado, requerendo um ou mais recursos para executá-la. Uma atividade pode consumir ou produzir artefatos. Num processo real, as atividades são divididas em sub-atividades, que, por sua vez, podem também ser subdivididas. Dependendo da complexidade, uma atividade pode ser subdividida em vários níveis. Uma atividade de processo é instanciada em cada projeto podendo ser customizada;
- **Artefatos**: São produtos de *software*. Podem ser produzidos ou consumidos por uma atividade. São exemplos de artefatos: documentos de especificação de requisitos, manuais de qualidade, atas de revisão, diagramas de classes e código fonte;
- **Recursos**: São as pessoas da organização, as ferramentas de *software*, os equipamentos ou quaisquer outros recursos, necessários à execução de uma atividade;
- **Processo**: Um processo de desenvolvimento de *software* é um conjunto organizado de atividades, métodos, ferramentas, procedimentos e técnicas, com o fim de desenvolver e manter um *software*;
- **Projeto**: Consiste no desenvolvimento de *software*, que emprega um processo de *software*, englobando o conjunto de atividades necessárias, artefatos consumidos e produzidos e recursos utilizados.

A arquitetura inicial proposta por PASCUTTI (2002) inclui um Gerenciador de Objetos que é composto de sete gerenciadores específicos, conforme é apresentado na Figura 10.

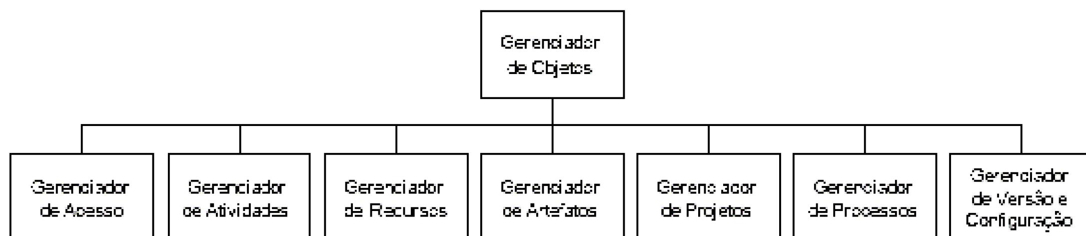

Figura 10– Gerenciador de objetos proposto por PASCUTTI (2002)

A organização do Gerenciador de Objetos da arquitetura inicial é mantida neste trabalho, porém, com alguns ajustes. Por questões de implementação, o Gerenciador de Recursos foi estendido em mais três gerenciadores:

- Gerenciador de Usuários;
- Gerenciador de Locais;
- Gerenciador de Ferramentas.

O Gerenciador de Recursos, em sua visão mais ampla, pode ser utilizado para descrever ferramentas externas ao ambiente, recursos físicos como máquinas ou ser estendido para a criação e especialização de outros recursos ao ambiente. O Gerenciador de Ferramentas é o responsável por gerenciar as ferramentas internas ao ambiente e agrega o antigo **Gerenciador de Versão e Configuração**. O **Gerenciador de Acesso** foi dividido entre o

Gerenciador de Usuários para o acesso de pessoas e Gerenciador de Locais para o acesso autonômico. Isto porque cada um destes gerenciadores pode ser visto como uma especialização de recursos e utiliza suportes distintos para seu funcionamento.

O **Gerenciador de Atividades** foi incorporado ao Gerenciador de Processo e Gerenciador de Projetos. As atividades definidas em um processo são instanciadas em um projeto. Durante a execução de um projeto as atividades podem ser customizadas sem que esta alteração afete a atividade do processo.

Os demais gerenciadores foram mantidos. Uma visão geral do Gerenciador de Objetos no FRADE está na Figura 11.

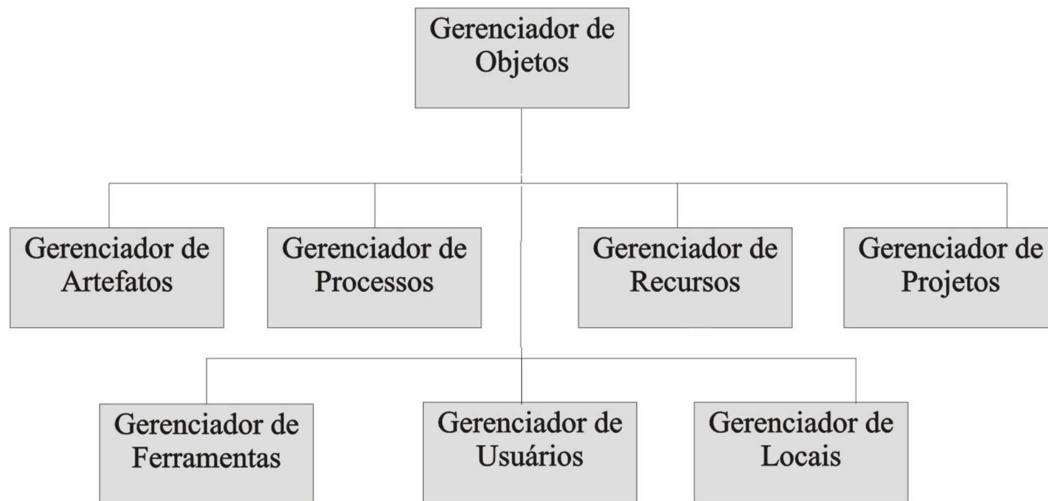

Figura 11 – Gerenciador de Objetos do FRADE

Por representar a camada de negócios, o Gerenciador de Objetos será a Interface desta camada. Seus métodos incluem métodos de acesso a todos os demais gerenciadores e estão ilustrados na Figura 12.

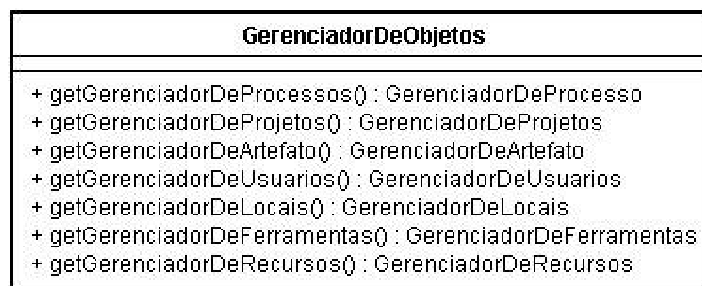

Figura 12 – Iterface do Gerenciador de Objetos

Para permitir que estes gerenciadores possam ser modificados em cada instância do ambiente, sem que seja necessário mudar a implementação deste gerenciador, o mesmo

também se baseia em um arquivo XML e uma fábrica[4] para a sua instanciação. A Figura 13 mostra o arquivo de configuração para esta camada.

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
    <mapping key="disen.negocio.abstract.GerenciadorDeArtefato"
target="disen.negocio.GerenciadorDeArtefato" />
    <mapping key="disen.negocio.abstract.GerenciadorDeProcessos"
target="disen.negocio.GerenciadorDeProcessos" />
(...)
    <mapping key="disen.suporte.abstract.Suporte"
target="disen.suporte.Suporte" />
</config>
```

Figura 13 – Arquivo de configuração do Gerenciador de Objetos

Cada gerenciador da camada de negócios é composto por classes encapsuladas que guardam seus meta-dados e classes que guardam suas regras de negócio. Para a implementação destes gerenciadores e a divisão entre os meta-dados dos objetos e suas regras de negócio foi utilizado o padrão de projeto DAO (*Data Access Object*) e BO (*Business Object*) do *core* do J2EE[5]. Segundo este padrão, todos os gerenciadores do ambiente possuirão um Java Bean e um BO. A Interface comum aos gerenciadores é mostrada na Figura 14.

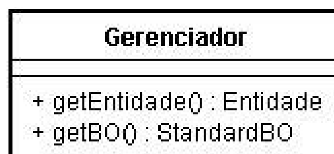

Figura 14 – Interface dos Gerenciadores do *Framework*

Por entidade, entende-se os meta-dados das classes de negócio. O conceito de entidade será melhor explicado na seção 3.5.1 quando será tratado o suporte à persistência de meta-dados. As regras de negócio dos objetos são expressas pelo seu BO. Todo BO possui uma implementação para a interface mostrada pela Figura 15.

---

[4] A fábrica aqui descritas é uma implementação do padrão de projeto *Abstract Factory*. Há no *framework* uma fábrica abstrata capaz de instanciar fábricas para que estas instanciem os objetos de cada camada. As fábricas para cada camada são implementações do padrão de projeto *Factory*.

[5] Um DAO pode ser visto como uma representação do repositório a fim de persistir e recuperar objetos. Um BO contém os métodos que implementam as propriedades não persistentes e as regras de negócio que podem ser necessárias ao objeto.

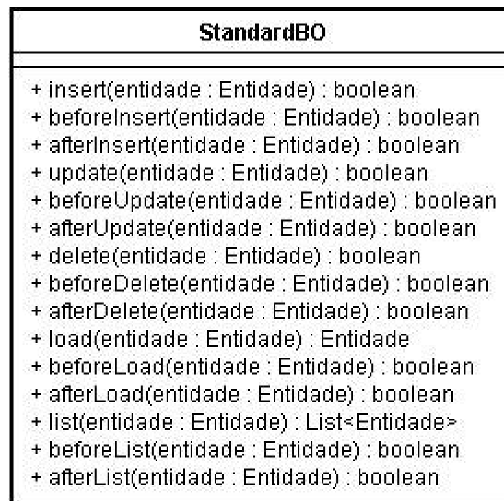

Figura 15 – Interface dos Objetos de negócios dos Gerenciadores

Os métodos que todo BO possui tratam como cada Objeto deve ser persistido, alterado, removido, carregado e listado. Além disto, os métodos desta interface permitem definir o que fazer antes e depois de cada uma destas operações sobre o objeto. Assim, pode-se, por exemplo, verificar se um determinado usuário pode ou não alterar um objeto antes de o mesmo ser alterado, ou iniciar uma ferramenta após um objeto ser carregado.

Além de garantir o acesso ao gerenciador de objeto, a interface desta camada garante, também, o acesso à camada de infra-estrutura. A sua interface é definida como mostra a Figura 16.

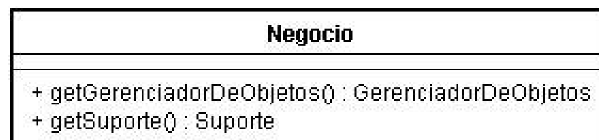

Figura 16 – Interface da camada de Negócios

Como na camada de aplicação, a intenção é criar uma camada constituída de componentes que possuam uma Interface bem definida. A Interface de cada um dos gerenciadores do *Framework* será explicada a seguir. A interface dos suportes será definida na seção 3.5.

3.4.1 Gerenciador de Processos

Segundo CUGOLA e GHEZZI (1998), a *engine* de um processo é o núcleo de um PSEE. Ela é composta de três componentes lógicos principais:

- Um interpretador do modelo de processo que executa o modelo de processo por meio do controle das ferramentas utilizadas durante o desenvolvimento.

- Um ambiente de interação humana que é composto pelas ferramentas utilizadas no processo. Estas ferramentas podem ser: editores, compiladores, agendas, ferramentas de gerenciamento, entre outras.
- Um repositório para armazenar artefatos durante o processo.

Seguindo esta visão, o gerenciador de processo seria o mais abstrato dos gerenciadores do Gerenciador de Objetos por ser composto de todos os demais gerenciadores. Mesmo assim posui vários atributos que representam os processos de *software* e torna-o capaz de ser instanciado e persistido permitindo sua reutilização. Os modelos de processo podem ser replicados entre os usuários, e diferentes versões podem ser executadas (FREITAS, MAIA, NUNES, 2004). Além disto, cada local deve ter autonomia para escolher entre processos ou para alterar o processo de desenvolvimento de *software* (BEN-SHAUL, KAISER, 1994).

Segundo FRANCH e RIBO (1999), um detalhe importante sobre as linguagens de modelo de processo é a falta de uma notação que expressa formalismo às propriedades do modelo e de suas fases. A notação formal das atividades, invariantes, pré e pós-condições seria a garantia de confiabilidade da definição do processo.

A estrutura dos objetos de negócio do processo é apresentada na Figura 17. Um processo possui um nome e é composto de fases, sendo estas fases definidas por atividades. Além disto, um processo será utilizado para especificar projetos. As atividades do Processo são definidas por tarefas. Cada tarefa pode possuir um ou mais artefatos de entrada e saída, além de poder ser atribuída a um ou mais usuário responsável. Como esta definição de atividade pertence ao processo, as tarefas não são associadas aos usuários, mas a um papel.

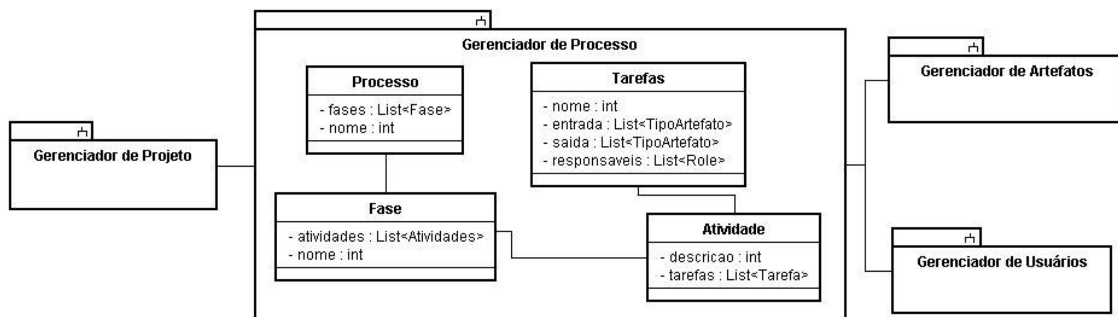

Figura 17 – Diagrama de pacotes do Gerenciador de Processos

Assumindo o Gerenciador de Processos como um ator dentro do ambiente, o mesmo teria os casos de uso descritos pela Figura 18.

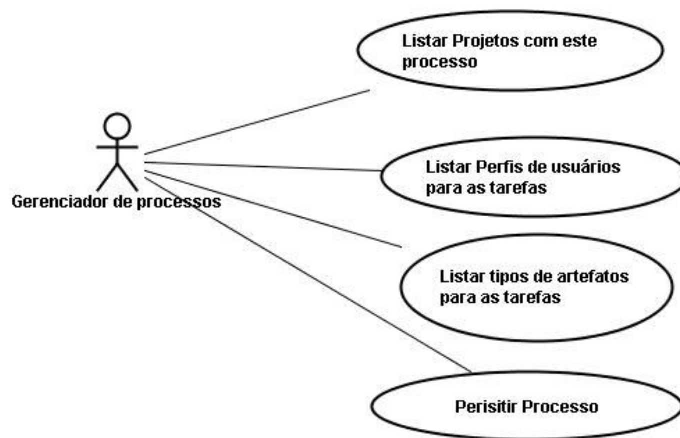

Figura 18 – Casos de uso do Gerenciador de Processos

As classes Processo e Fase serão persistidas e por isto devem estender Entidade (Figura 67). Devido à necessidade de composição entre processo e projeto, processo e artefato, e processo e perfis de usuário, há uma dependência entre estes gerenciadores. Estas dependências são representadas pela Figura 19.

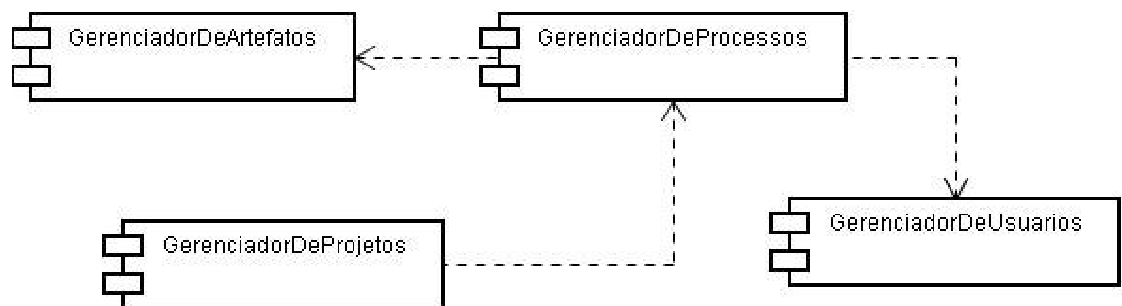

Figura 19 – Dependência do Gerenciador de Processo

Como o gerenciador de processo necessita persisitir o processo, o mesmo utiliza o suporte a persistência da camada de Infra-estrutura para armazenar seus meta-dados. A estrutura do gerenciador e seu acesso ao suporte é mostrada de maneira simples pela Figura 20.

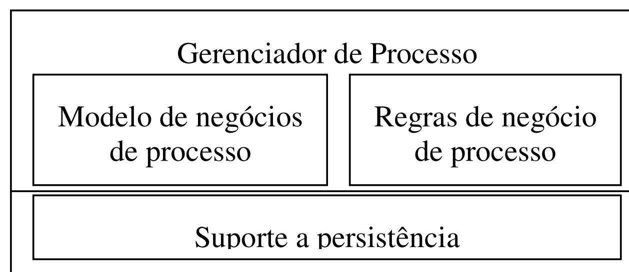

Figura 20 – Acesso do Gerenciador de processo aos suportes

As regras de negócio do processo, chamado ProcessoBO, deverão conter apenas chamadas para o suporte à persistência com o objetivo de persistir a definição do processo a partir de suas propriedades, como ilustra a Figura 21.

```
public class ProcessoBO implements StandardBO{

boolean insert(Entidade entidade){
if (this.beforeInsert(entidade))
      return false;
disen.suporte.processo.processoDAO.insert(entidade);
if (this.afterInsert(entidade))
      return false;
return true;
}
(...) }
```

Figura 21 – Método insert na Classe ProcessoBO

Esta implementação do ProcessoBO não verifica se o usuário possui permissão para inserir um novo processo. Esta verificação pode ser feita no método beforeInsert desta mesma classe, porém, não está no escopo deste trabalho definir as políticas de acesso de um ADDS, mas criar uma infra-estrutura que permita que estas políticas possam ser implementadas.

Definido o conteúdo deste gerenciador, é possível definir sua interface, como ilustra a Figura 22.

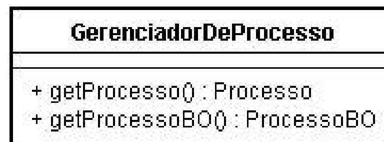

Figura 22 – Interface do Gerenciador de Processo

A interface do componente GerenciadorDeProcesso tem a função de atuar como padrão de projeto FACADE, escondendo sua implementação e sua estrutura de classes interna.

3.4.2 Gerenciador de Projetos

O Gerenciador de Projetos possui a função de representar projetos e permitir que os mesmos sejam instanciados dentro do ambiente. O Gerenciador de Projetos do *framework* se relaciona com outros gerenciadores e possui uma função mais abrangente por ser, depois do Gerenciador de Processo, o mais genérico dos gerenciadores.

A estrutura dos objetos de negócio do projeto é apresentada na Figura 23. Um projeto possui um nome e é instanciado a partir de um processo que irá definir as fases e as atividades deste processo. Um Projeto necessita também da alocação de recursos para que o mesmo possa ser instanciado.

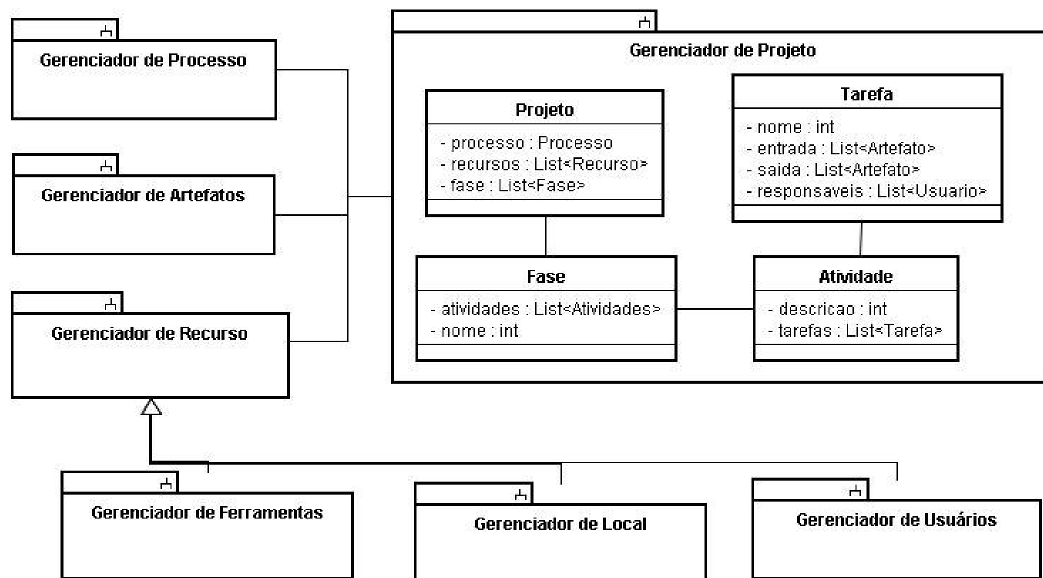

Figura 23 – Diagrama de pacotes do Gerenciador de Projetos

Ao ser instanciado o projeto a partir do processo, o mesmo poderá ser alterado sem comprometer a alteração do processo ou de outros projetos que dependam deste processo. Uma alteração do Processo também não deverá repercutir nos projetos em andamento. A diferença entre as tarefas do processo e do projeto é que, no projeto, a mesma é passada para usuários que serão os responsáveis por esta tarefa, e que, gerarão os artefatos que podem resultar da execução desta tarefa.

Assumindo o Gerenciador de Projetos como um ator dentro do ambiente, o mesmo teria os casos de uso descritos pela Figura 24.

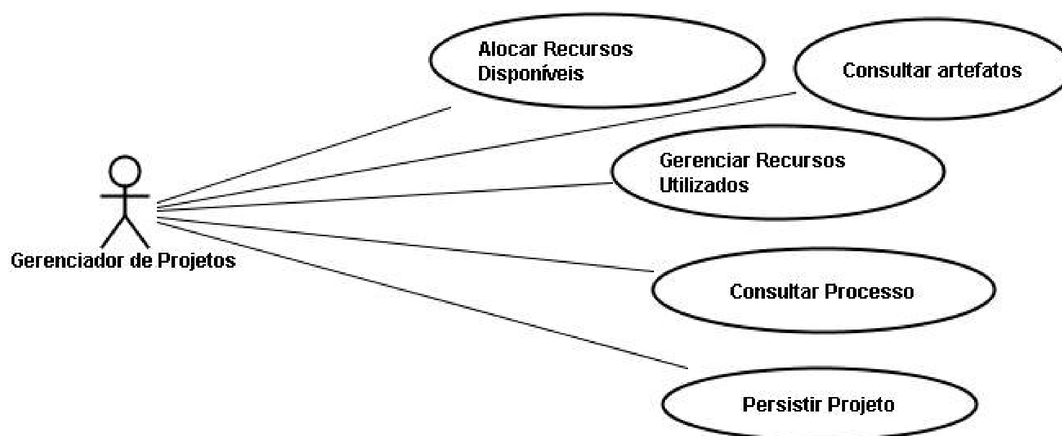

Figura 24 – Casos de Uso do Gerenciador de Projetos

A classe Projeto será persistida e por isto deve estender Entidade (Figura 67). A definição da classe Recurso que será utilizada em um projeto será feita na seção 3.4.4. O GerenciadorDeProjetos depende do GerenciadorDeArtefatos, GerenciadorDeRecursos e GerenciadorDeProcessos. A dependência do GerenciadorDeProjetos, seus artefatos, seu processo e os recursos de um projeto é expressa pela Figura 25.

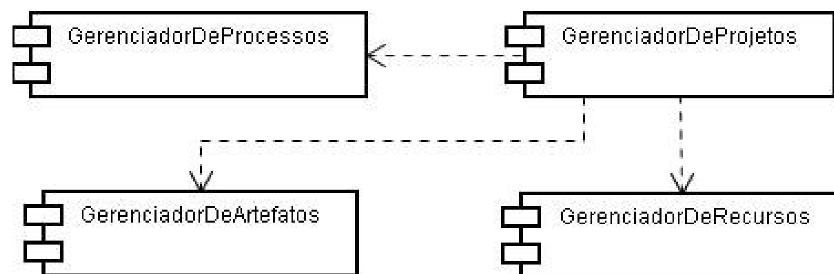

Figura 25 – Dependências do Gerenciador de Projetos

A Figura 23 ilustra o acesso das regra de negócios de Processo por parte do ProcessoBO.

```
public class ProjetoBO implements StandardBO{

boolean insert(Entidade entidade){
if (this.beforeInsert(entidade))
      return false;
disen.suporte.processo.processoDAO.insert(entidade);
if (this.afterInsert(entidade))
      return false;
return true;
}
(...)
}
```

Figura 26 – Estrutura do Gerenciador de Projetos

Assim como o Gerenciador de Processo, o Gerenciador de Projeto depende apenas do suporte a persistência para armazenar os meta-dados dos projetos instanciados. Desta maneira, sua comunicação com o suporte é feita apenas com o suporte a persistência, como ilustra a Figura 27.

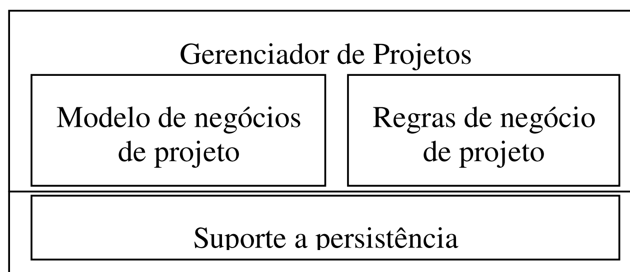

Figura 27 – Acesso do Gerenciador de Projetos aos suportes

A interface do GerenciadorDeProjetos possui o acesso à sua estrutura interna, como ilustra a Figura 28.

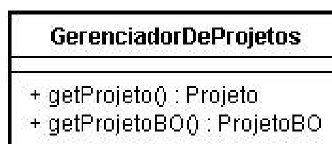

Figura 28 – Interface do Gerenciador de Projetos

3.4.3 Gerenciador de Artefatos

Segundo DOSSICK e KAISER (1999), os projetos de desenvolvimento de *software,* normalmente, envolvem mais que códigos fonte. Até mesmo desenvolvimentos pequenos ou médios podem envolver centenas de artefatos: documentos de projeto, requisições de alteração, documentos de revisão de código e documentos relacionados.

BEN-SHAUL, KAISER e HEINEMAN (1992) lembram que é necessário distinguir os dados do produto e o controle dos dados. Os dados do produto são os artefatos de *software* propriamente dito. Mais que armazenar os artefatos é necessário armazenar os meta dados de sua localização e informações de acesso (DOSSICK, KAISER, 1999).

Segundo ESTUBLIER (2000), um gerenciador de artefato deve apoiar:

1. Versões;
2. Modelo de produto;
3. Composição;
4. Construção e reconstrução;
5. Suporte a *workspace* (cópias locais).
6. Suporte a trabalho cooperativo

Além disto, ESTUBLIER e CASALLAS (1994) identificaram três classes de versões:

- **Temporal** – Mantém o histórico dos objetos, controle e atividades.
- **Lógica** – aonde o objeto possui múltiplas variações de representação.
- **Dinâmica** – Aonde o objeto possui alterações múltiplas e concorrentes.

Por esta razão, o Gerenciador de Artefatos possui uma grande integração com o suporte à persistência pois os dados sobre os artefatos persistidos são relacionados as atividades de um determinado usuário do ambiente aonde o ambiente está instanciado

O gerenciamento de artefatos em ADSs varia significativamente, de acordo com sua granularidade. Enquanto a maioria dos ADSs consideram os produtos de acordo com seu desenvolvimento e seu relacionamento no nível documental (como relacionados a diagramas, documentos de projetos ou módulos de testes), a estrutura de um produto com uma granularidade mais fina como entidades, relacionamentos e atributos também pode ser considerada (POHL et al., 1999).

O componente de gerenciamento de informação de um ADS precisa lidar com vários tipos de dados distintos, entre eles (BARGOUTHI et al., 1996):

- Dados de produtos como códigos fonte, dados de gerenciamento de configuração, documentação e demais artefatos;
- Dados de processo como a definição formal de um processo de *software*, seus dados, projetos e demais classes que o compõe;
- Dados organizacionais como o gerenciamento de recursos e configuração.

As classes que compõe o Gerenciador de Artefatos são ilustradas na Figura 29 a seguir.

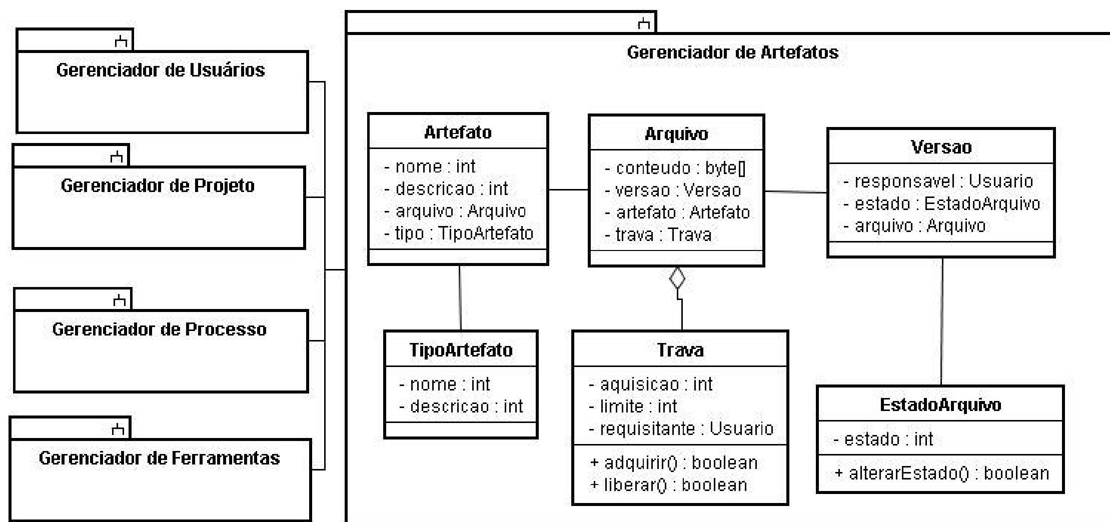

Figura 29 – Diagrama de pacotes do gerenciador de artefatos

Analisando o Gerenciador de Artefatos como um ator em nosso *framework*, o mesmo teria os casos de uso que estão na Figura 30.

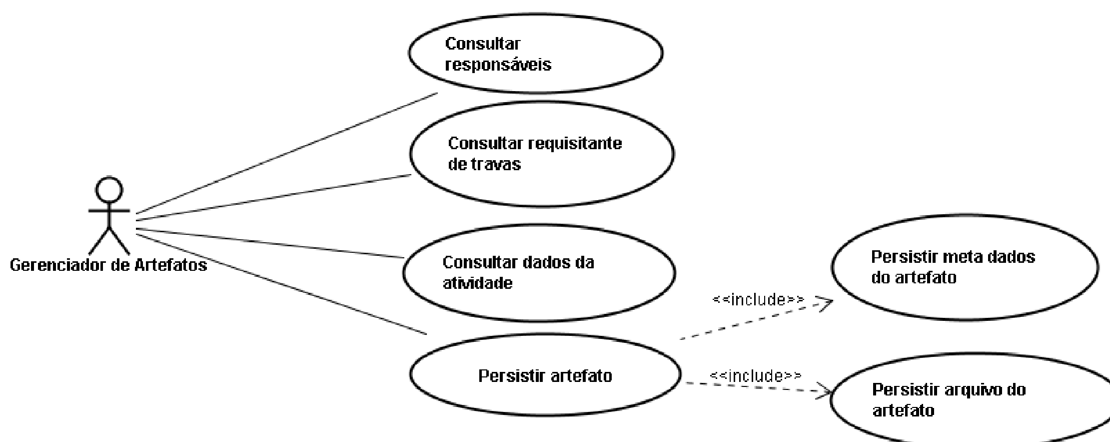

Figura 30 – Casos de uso do Gerenciador de Artefatos

Um artefato pode possuir várias versões. Uma versão de um artefato foi desenvolvida por um Usuário. A dependência entre o Gerenciador de Artefato e o Gerenciador de Usuários é mostrada na Figura 31. O Gerenciador de Usuários será melhor explicado na seção 3.4.6.

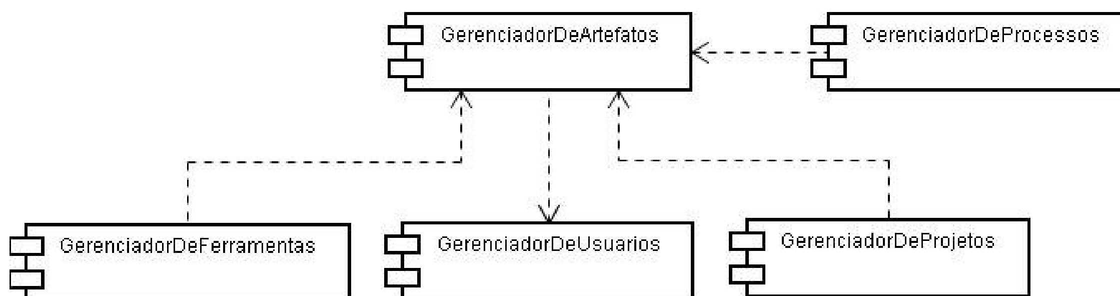

Figura 31 – Dependência do gerenciador de Artefatos

Entre as classes que compõem o modelo de dados do Gerenciador de Artefato apenas Artefato, TipoArtefato, Versão e Usuários necessitam de persistência de seus meta-dados. As demais classes são apenas classe de controle para a transação de Artefatos. A classe Arquivo representa o arquivo propriamente dito. Este arquivo não será persistido com o suporte de persistência mas com o suporte a artefatos. Por esta razão, o Gerenciador de Artefatos irá operar sobre dois suportes, conforme ilustra a Figura 32.

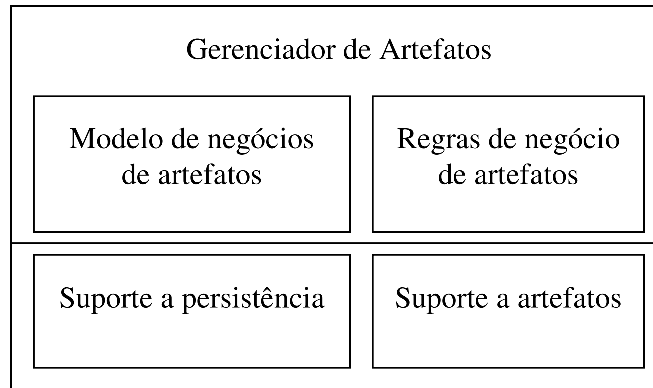

Figura 32 – Acesso do Gerenciador de Artefatos aos suportes

A implementação dos Objetos de negócios do Gerenciador de Artefatos é distinta dos demais por necessitar ativar o suporte a artefatos. Um exemplo de implementação do ArtefatoBO é ilustrado pela Figura 33 .

```
public class ArtefatoBO implements StandardBO{

boolean insert(Entidade entidade){
if (this.beforeInsert(entidade)
      return false;
disen.suporte.artefato.artefatoDAO.insert(entidade);
disen.suporte.artefato.enviarArquivo(((Artefato)entidade).getArquivo());
if (this.afterInsert(entidade)
      return false;
return true;
}
(...)
```

Figura 33 – Implementação do ArtefatoBO

A interface do componente GerenciadorDeArtefatos permite o acesso aos dados dos artefatos e é apresentada na Figura 34.

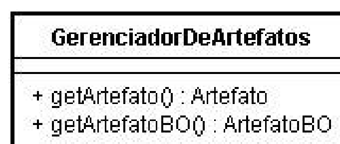

Figura 34 – Interface do Gerenciador de Artefatos

3.4.4 Gerenciador de Recursos

O Gerenciador de Recursos do *framework* possui um nível de abstração que permite uma representação ampla de recursos para um ADS. Ele é especializado em 3 gerenciadores: Gerenciador de Usuários, Gerenciador de Locais e Gerenciador de Ferramentas. Apesar destas especializações, o Gerenciador de Recursos por si só permite representar os demais recursos de um ambiente de *software* como equipamentos, livros ou recursos financeiros que precisam ser gerenciados dentro de um processo de *software*. Esta seção tratará do Gerenciador de Recursos em sua visão genérica. Os Gerenciadores de Usuários, Locais e Ferramentas serão tratados nas seções 3.4.6, 3.4.5 e 3.4.7, respectivamente.

A informação em um modelo de recurso pode ser utilizada para controlar o compartilhamento de recursos, utilizações futuras e razões que interfiram no tempo de execução das atividades. A associação de um recurso a um projeto deve ser definida com as suas tarefas, e não é responsabilidade do gerenciador de recursos controlar esta associação. A responsabilidade do Gerenciador de Recursos é fornecer informações sobre os tipos de recursos e sua disponibilidade e acompanhar sua utilização. Além disto, um Gerenciador de Recursos deve manter um histórico de requisições por recursos que falharam e até prever a necessidade de mais recursos de um tipo particular em determinada situação (LERNER et al., 2000).

Para identificar todos os recursos de dão suporte a uma necessidade particular, o cliente da especificação de recursos especifica o nome de uma classe de recursos e seus atributos. O Gerenciador de Recursos então busca um recurso que atende ao requisitado. Entre os recursos que o desenvolvimento de um sistema pode necessitar estão os recursos humanos, ferramentas, plataformas computacionais e dados e diversas outras associações entres estes tipos que são necessárias para prover uma alocação inteligente destes (LERNER et al., 2000).

A função desta visão mais genérica de recursos dentro do ambiente é permitir as transições dos recursos em um workflow que permita (LERNER et al., 2000):

- **Alocar recursos**: Alocar um recurso é associá-lo a uma requisição. A aquisição de um recurso por uma tarefa é a identificação do recurso necessário a partir de sua associação. Quando um recurso é disponibilizado para uma tarefa ele se torna indisponível para as demais tarefas.
- **Reservar recursos**: A aquisição resulta na indisponibilização imediata do recurso para outras atividades. A reserva suporta a habilidade de planejar recursos para seu uso futuro. A requisição de uma reserva é a especificação de um recurso ou classe de recursos a ser reservada no momento que o mesmo será utilizado e, por quanto tempo a futura alocação será necessária. A reserva é feita no nível da especificação do recurso.
- **Liberar recursos**: Se um recurso é alocado ou reservado, o mesmo será redisponibilizado para outras atividades através da sua liberação. Um recurso só é liberado quando a atividade associada é completada ou cancelada. O recurso também pode ser liberado quando seu tempo de alocação expira e há outra atividade com reserva para o mesmo.

A Figura 35 apresenta o diagrama de pacotes do Gerenciador de Recursos. O estado do recurso representa a situação atual do recurso, que pode ser liberado, alocado/reservado, alocado ou liberado. A alocação / reserva de um recurso é feita por projeto e depende de uma data para isto. A data prevista para a liberação do recurso depende do cronograma deste

projeto. Há ainda uma descrição do tipo de recurso que define, de maneira textual, o tipo deste recurso. Um recurso ainda pode possuir um local, que significa aonde o mesmo se encontra fisicamente.

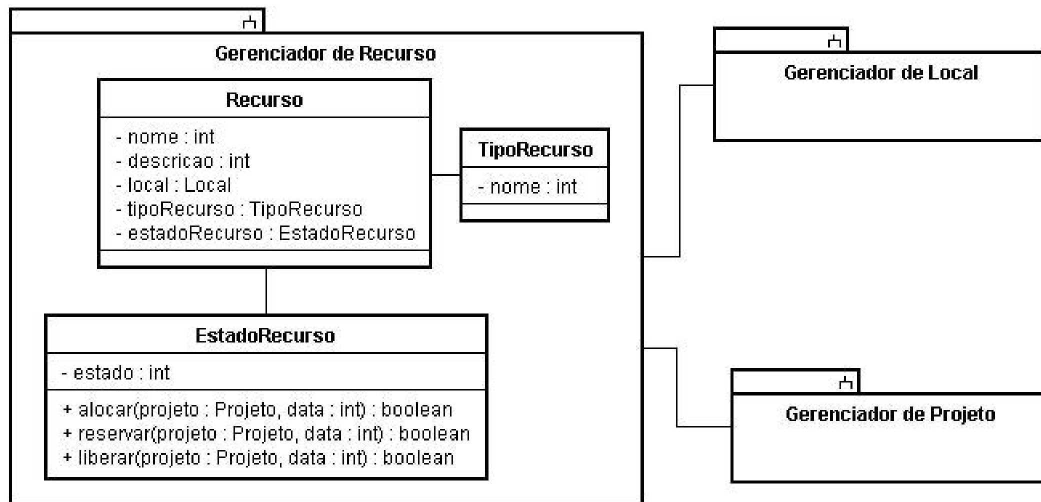

Figura 35 – Diagrama de pacotes do Gerenciador de Recursos

Analisando o Gerenciador de Recursos como um ator dentro do FRADE, o mesmo teria os casos de uso ilustrados pela Figura 36.

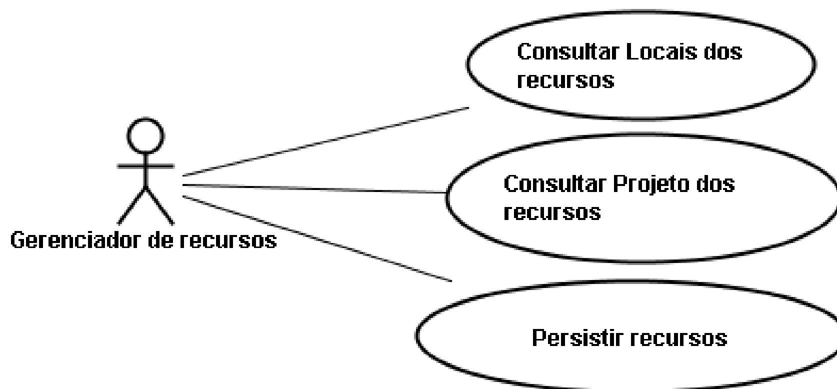

Figura 36 – Casos de uso do gerencidor de recursos

A dependência entre o Gerenciador de Recursos, o Gerenciador de Projetos e o Gerenciador de Locais é ilustrada na Figura 37.

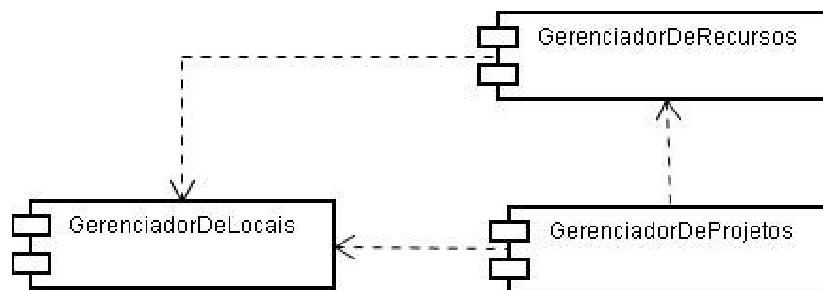

Figura 37 – Dependência do Gerenciador de Recurso

O gerenciador de recursos necessita persisitir os dados sobre os recursos do ambiente. Para isso ele faz uso do suporte à persistência, como ilustra a Figura 38.

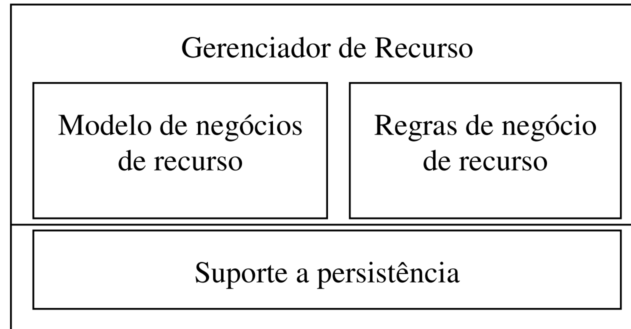

Figura 38 – Acesso do Gerenciador de Recursos ao suporte

O Gerenciador de Recursos traz em sua interface o acesso ao recurso e ao seu Objeto de negócios, como demonstra a Figura 39.

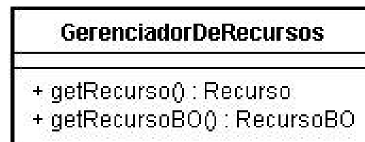

Figura 39 – Interface do Gerenciador de Recursos

3.4.5 Gerenciador de Locais

Caso um serviço se torne indisponível na rede, é necessário reconfigurar o sistema de maneira a buscar outro servidor que possua o mesmo serviço. Segundo TSAI et al. (2004), há duas circunstâncias para a reconfiguração e atualização de um sistema:

- **Iniciada pelo cliente**: O cliente possui uma lista com os locais que fornecem determinados serviços e tem capacidade de conectar-se em outro local caso o serviço que está sendo usado no momento deixar de operar;
- **Iniciado pelo ambiente**: Como os servidores sabem quais usuários estão conectados aos demais servidores, estes podem alterar dinamicamente suas listas de locais para adicionar outro servidor ou para dividir a carga de serviço de um nó da rede que esteja sobrecarregado.

O Gerenciador de Locais permite a reconfiguração dinâmica da rede de distribuição do ambiente, de maneira a adicionar, remover ou alterar a função de um sub-ambiente dentro do ambiente global. Isto deve ocorrer pois organizações podem compartilhar um projeto apenas por determinado tempo (BEN-SHAUL, KAISER, 1994)

Desta maneira, o ambiente conseguirá prover facilidade para processos distribuídos (CHAMBERS, LANG, 1999) através da distribuição de seus serviços dentro da rede. Isto torna o ambiente facilmente modificável e escalonável (NARENDRA, 2000).

O ambiente a ser implementado por este *framework* possui vários serviços, conforme serão vistos na seção 3.5, que descreve a sua infra-estrutura. Para a localização destes serviços o Gerenciador de Locais oferece um serviço de páginas amarelas que lista qual serviço é oferecido por qual estação de trabalho. Várias estações podem estar executando os mesmos serviços e/ou vários serviços podem estar distribuídos entre várias estações.

Há os serviços, como o repositório local de artefatos, que são executados localmente pelo usuário para garantir que os mesmos possam trabalhar desconectados do ambiente quando necessário. Estes serviços locais devem armazenar apenas os dados relacionados ao usuário que está operando o ambiente. Porém, para os serviços que são compartilhados por todos os usuários é necessário que esta estação possua autorização do gerente de local para oferecer este serviço. A distribuição dos serviços precisa garantir que os locais listados por este gerenciador sejam confiáveis. Esta garantia é obtida por meio da autorização para executar serviços.

A autorização de executar ou não um serviço em uma estação é dada pelo suporte a acesso. Esta autorização é dada a um determinado usuário, que passa a ser o responsável por este Local / Serviço. Este usuário só pode disponibilizar este serviço quando estiver acessando o ambiente a partir de um determinado IP / Porta. As classes de negócio do Gerenciador de Locais é apresentada na Figura 40.

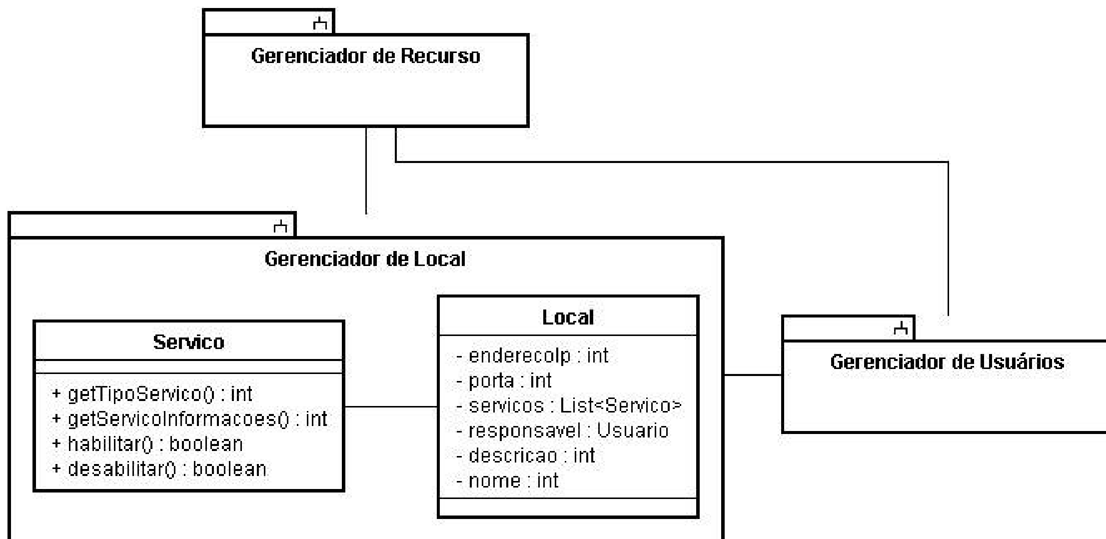

Figura 40 – Diagrama de pacotes do Gerenciador de Locais

A classe Serviço é uma Interface implementada por todos os Serviços do ambiente e será explicada na seção 3.5. Por possuir um usuário conectado em um local, o gerenciador de Locais possui uma dependência com o gerenciador de Usuários, como ilustra a Figura 41.

Figura 41 – Dependência do Gerenciador de Locais

Esta dependência entre o Gerneciador de Locais e o Gerneciador de Usuários pode ser vista com a visão de ator do Gerenciador de Locais, conforme ilustra a Figura 42.

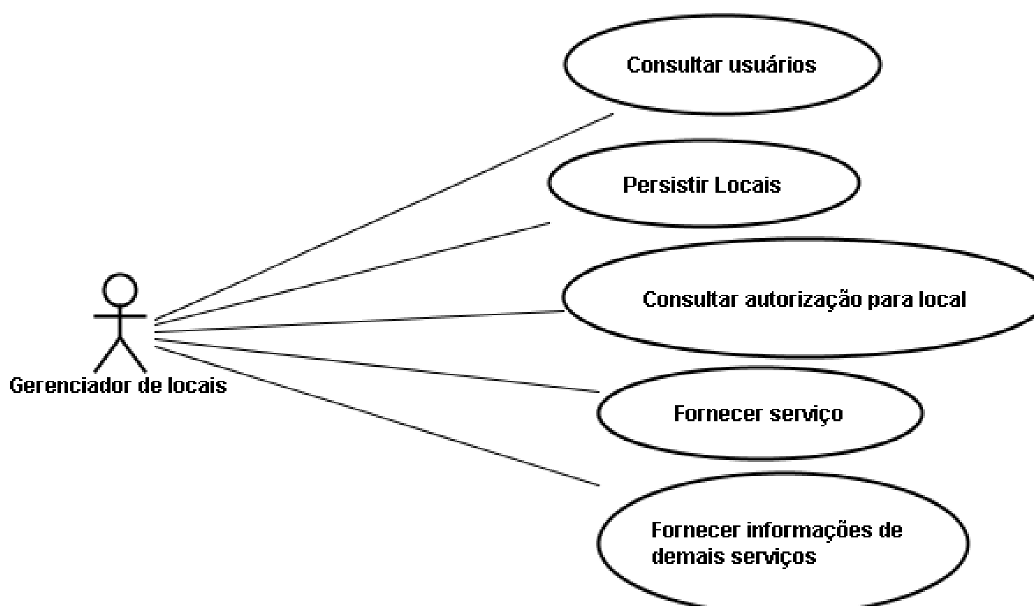

Figura 42 – Casos de uso do Gerenciador de Locais

Este diagrama ilustra que, além de fornecer um determinado serviço, um local deve também fornecer aos usuários informações sobre os demais serviços do ambiente. Desta maneira, a informação de um novo serviço que seja incluído ao ambiente será repassada aos usuários quando os mesmos conectarem-se a um serviço qualquer do ambiente. Todos os locais devem possuir a lista de locais/serviços do ambiente. Esta replicação de informação sobre os locais do ambiente é responsabilidade do suporte e serviço de distribuição que serão apresentados nas seções 3.5.11 e 3.5.12 deste trabalho.

Por possuir a função de descrever os serviços, o Gerenciador de Locais deve operar sobre o suporte à distribuição do ambiente que será o serviço de páginas amarelas para os usuários. Conforme é ilustrado pelo diagrama de caso de uso, Figura 42, como um local precisa de autorização para oferecer um serviço, este gerenciador também atua sobre o suporte ao acesso. Para persistir os meta-dados dos locais que compõe o ambiente, este gerenciador utiliza também o suporte à persistência. A Figura 43 apresenta a utilização dos suportes por este gerenciador.

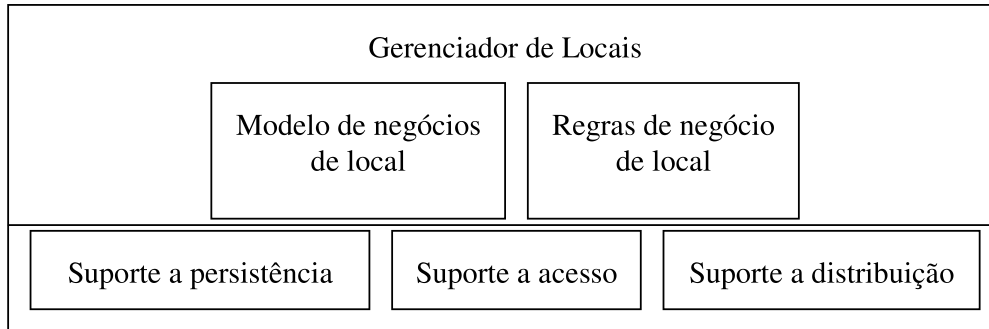

Figura 43 – Acesso do gerenciador de Locais aos suportes

Para acessar estes suportes, as regras de negócio do LocalBO estendem o StandardBO para garantir a persistência e possui outros métodos, como ilustra a Figura 44.

Figura 44 – Regras de negócio para Locais: LocalBO

A interface do Gerenciador de Locais fornece acesso à classe de negócios de Local e às suas regras de negócio, como apresenta a Figura 45.

Figura 45 – Interface do Gerenciador de Locais

3.4.6 Gerenciador de Usuários

Segundo CREIGHTON, DUTOIT e BRUEGGE (2003), os recursos humanos em uma organização podem ser agrupados de maneira arbitrária e as associações este estes usuários possuem atributos que suportam alterações na estrutura organizacional a qualquer momento. Um projeto pode ser representado pela soma dos recursos que o mesmo usa e, em um nível maior de abstração, uma lista de controle de acesso que exibe as funções que cada membro desempenha dentro do projeto.

Mais do que verificar as permissões de acesso dentro do projeto, as funções de cada membro da equipe podem facilitar a identificação da necessidade que cada um pode possuir dentro do projeto, suas tarefas e os recursos e artefatos que os mesmos podem estar utilizando (CREIGHTON, DUTOIT, BRUEGGE, 2003).

Além disto, o cadastro da equipe deve conter as funções de cada membro incluindo informações de contato pessoal. Esta informação de contato pode ser um simples telefone de contato ou um endereço de email. Porém, em uma equipe distribuída pode ser necessário incluir também informações quanto ao fuso horário. De uma maneira mais elaborada, estas informações devem incluir, até mesmo, sugestões de características físicas deste contato de maneira a sugerir ao sistema qual a melhor maneira de entrar em contato com cada membro da equipe (DAMIAN et al., 2003).

O fornecimento de informação sobre as características de cada empregado em uma organização é uma informação vital para a manutenção desta. Para obter um modelo de informação que possua o mapeamento de características necessárias para determinada organização é necessário, primeiramente, definir explicitamente o modelo organizacional de maneira que seja possível a uma ferramenta se adaptar ao modelo pré-estabelecido (CREIGHTON, DUTOIT, BRUEGGE, 2003).

Tomando por metáfora a lista telefônica, se o Gerenciador de Locais é a seção de páginas amarelas, o Gerenciador de Usuários seria a seção de páginas brancas. Segundo CREIGHTON, DUTOIT e BRUEGGE (2003), indo além da lista telefônica, o mecanismo de localização de usuários deve permitir a busca por usuário que vai além da ordenação alfabética.

Isto porque, segundo WERNER et al. (2003), os processos de desenvolvimento de *software* são notavelmente colaborativos. A rotina diária dos engenheiros de *software* e da equipe de *software* requer coordenação das atividades e tarefas, compartilhamento contínuo de informação sobre o estado do projeto e conhecimento do domínio.

Por esta razão, além de conter os dados sobre os recursos humanos da organização, o Gerenciador de Usuários deve permitir também (BEN-SHAUL, KAISER, 1994):

- **Colaborar**: Múltiplos usuários em múltiplos locais devem conseguir colaborar para desenvolver o projeto como um todo;
- **Comunicar**: De maneira a permitir a comunicação entre usuários geograficamente distantes.

O modelo de negócios para o Gerenciador de Usuários implica na alocação de recursos humanos para o projeto. O trabalho de LIMA (2004) propõe um mecanismo para suporte a seleção de recursos humanos a ser empregado em ADS. Este mecanismo baseia-se em características dos usuários como Habilidades, Conhecimento, Afinidades. O diagrama da Figura 46 apresenta os dados de usuário que foram inicialmente mapeados para este gerenciador.

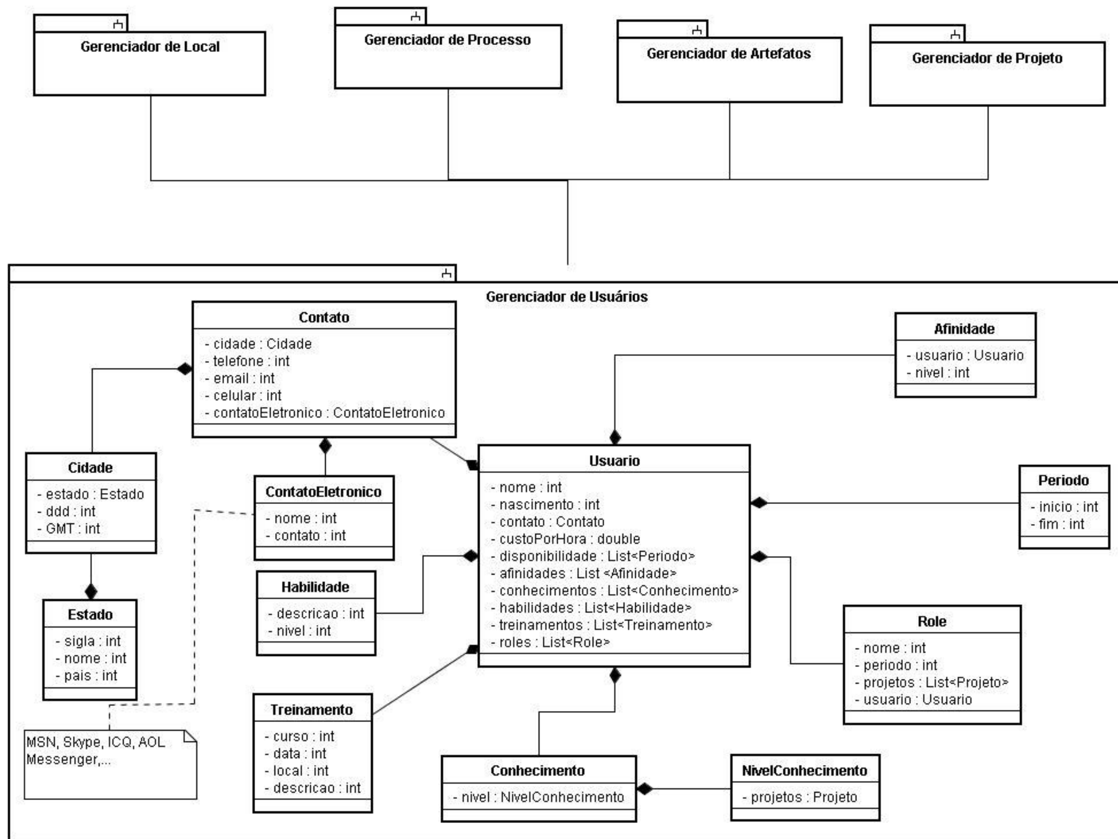

Figura 46 – Diagrama de pacotes do Gerenciador de Usuários.

Para ter acesso aos dados de projeto, um usuário é associado a uma função (*Role*) dentro do projeto. A participação em projetos define o conhecimento de um usuário, ou seja, os projetos nos quais ele realmente atuou. Um usuário pode ainda possuir a responsabilidade por manter um determinado serviço dentro do ambiente. Estas dependências entre o Gerenciador de Usuários e os gerenciadores de Projetos e Locais são expressas na Figura 47.

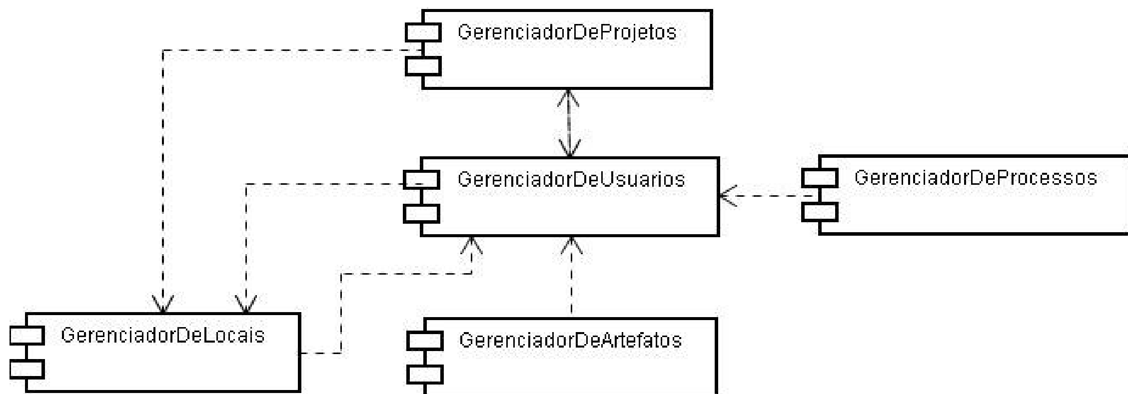

Figura 47 – Dependências do Gerenciador de Usuários

A integração do Gerenciador de Usuários ao ambiente pode ser vista através de seus casos de uso, conforme ilustra a Figura 48.

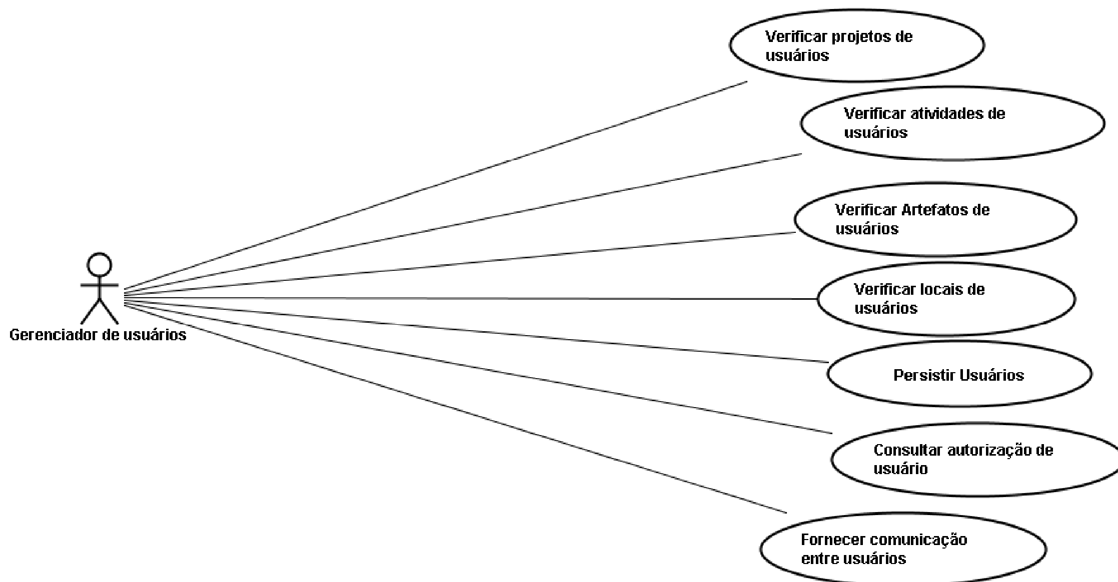

Figura 48 – Casos de uso do Gerenciador de usuários

Como é ilustrado por este diagrama de caso de uso, para que haja controle de acesso e permissões dentro do ambiente, o Gerenciador de Usuários utiliza o suporte a acesso para validar os usuários. Mais que garantir a validação de usuário e senha, o controle de acesso deve garantir se determinada tarefa pode ser realizada por um usuário com determinado perfil, conhecimento ou papel dentro do projeto ou do ambiente.

Para a colaboração e cooperação, o Gerenciador de Usuários utiliza o suporte à comunicação. Desta maneira, também é função do Gerenciador de Usuários permitir a comunicação, colaboração e cooperação entre participantes de um projeto de *software* instanciado no ADS. A Figura 49 ilustra a utilização dos suportes pelo Gerenciador de Usuários.

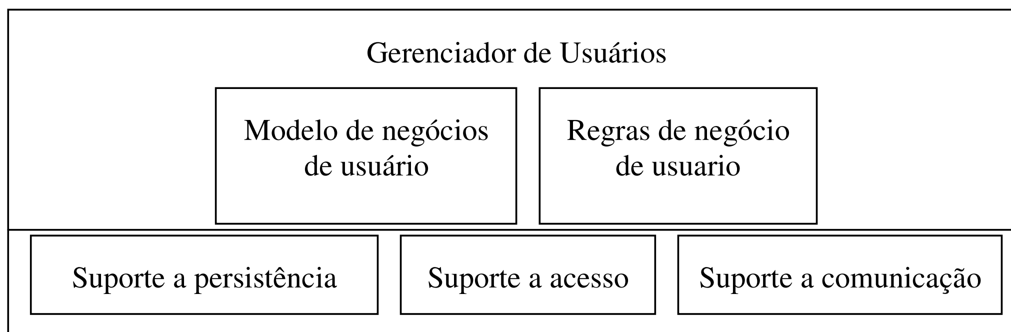

Figura 49 – Acesso do Gerenciador de Usuários aos suportes

As regras de negócio do UsuarioBO estendem o StandardBO o que garante a persistência dos seus meta-dados. O UsuarioBO possui ainda outros métodos que serão utilizados para ativar os demais suportes, como ilustra a Figura 50.

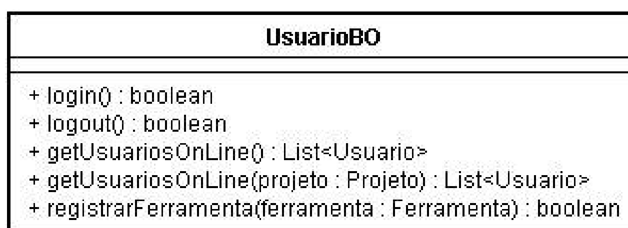

Figura 50 – Regras de negócios para Usuarios: classe UsuarioBO

O acesso ao Gerenciador de Usuários é feito através de sua interface que é apresentada na Figura 51.

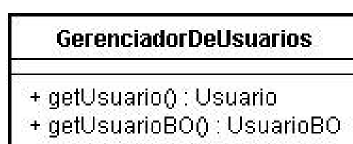

Figura 51 – Interface do Gerenciador de Usuários

3.4.7 Gerenciador de Ferramentas

Segundo FELDMAN (1979), um dos requisitos principais para o desenvolvimento distribuído é a garantia de acesso a um conjunto de ferramentas de desenvolvimento comum a todos os desenvolvedores. No desenvolvimento privado a organização se responsabiliza pelo fornecimento destas ferramentas. No caso do desenvolvimento não privado, por exemplo, o desenvolvimento do Linux, a instalação comum do sistema operacional conta com boa parte das ferramentas utilizadas em seu desenvolvimento, como as ferramentas CVS, gcc, make, autoconf, vi, Bugzilla, entre outras. (GERMAN, 2003).

As ferramentas de trabalho dos *stakeholders* atuam como uma fonte de informação em relação ao seu envolvimento no projeto. A ferramenta pode ser vista como um compartilhamento de uma lista de contato com anotações sobre as funções de cada usuário no projeto. A comunicação pode ser feita através da troca de mensagens ou por salas de bate papo (LANUBILE, 2003).

Segundo POHL et al. (1999), um serviço de ferramentas é uma funcionalidade fornecida por uma ferramenta que pode ser chamada externamente como no momento de criação de um determinado artefato, na compilação de um código ou na impressão de um documento. O serviço de ferramentas pode variar em sua complexidade. Mais que disponibilizar as ferramentas ao ambiente, pode ser necessário, também, disponibilizar bibliotecas que uma ferramenta necessita para operar corretamente.

As ferramentas de um ambiente deverão ser facilmente substituíveis, modificáveis e escalonáveis. Uma maneira de alcançar esta independência e flexibilidade é partir de uma arquitetura modular que permita que cada componente ou ferramenta possa ser desenvolvido de maneira independente de modo que seja possível escolher uma melhor opção para satisfazer os objetivos de um usuário (NARENDRA, 2000).

A possibilidade de integração de ferramentas ao ambiente pode garantir a extensibilidade do próprio ambiente. Tornar o ambiente estensível por ferramentas é garantir sua escalabilidade. Esta escalabilidade é alcançada com a escrita de novos processos ou a modificação dos processos existentes, com a incorporação de novas ferramentas, subprocessos, tipos ou objetos (TAYLOR et al., 1988).

No *framework* FRADE, é função do Gerenciador de Ferramentas disponibilizar a disponibilização de novas ferramentas a serem integradas ao ADS instanciado. Também é função do Gerenciador de Ferramentas integrar estas ferramentas ao *front-end* do usuário levando em consideração que o CORE deve ser usável por multiplos Front-Ends como GUI e texto (CHAMBERS, LANG, 1999)

Representar métodos e ferramentas em um nível conceitual é um pré-requisito para a comparação e mapeamento da definição de tarefas em um modelo de processo com serviços oferecidos por ferramentas ao ambiente. Para armazenar ferramentas sensitivas ao processo POHL, et al. (1999) usaram modelar as ferramentas não apenas em termos do serviço fornecido, mas, em termos de GUI e capacidade de interação.

Para a integração das ferramentas ao ambiente é necessário que haja uma funcionalidade no Gerenciador de Ferramentas que permita checar as dependências de uma aplicação antes de sua instanciação. FELDMAN (1979) cita a ferramenta make que utiliza um arquivo de descrição de pacote. Um arquivo de descrição contem três tipos de informação: definições de macro, informações de dependência e comandos executáveis. Estas descrições sobre a ferramenta a ser instalada são representadas no meta-modelos do Gerenciador de Ferramentas.

As classes do Gerenciador de Ferramentas apresentam os dados de dependência, informações sobre as versões das ferramentas, o seu pacote de instalação e a relação com o tipo de artefato que a mesma pode gerar. Nem todas as ferramentas em um ADS possuem um tipo de artefato relacionado. Uma ferramenta para comunicação entre usuários, por exemplo, não tem associação a um artefato. Por esta razão, ferramentas possuem uma categoria que permite classificá-las quanto à sua utilização. O diagrama da Figura 52 apresenta o diagrama de pacotes do Gerenciador de Ferramentas.

Gerenciador de Artefatos

Gerenciador de Ferramentas

Pacote
- descritor : int
- arquivo : byte[]
- nome : int

Ferramenta
- nome : int
- versao : VersaoFerramenta
- tiposArtefatos : List<TipoArtefato>
- categoria : int

VersaoFerramenta
- ferramenta : Ferramenta
- versao : int
- pacote : Pacote
- dependencias : List<Dependencia>

Dependencia

DependenciFerramenta
- ferramentas : List<Ferramenta>
- versao : VersaoFerramenta

DependenciaServico
- servico : Servico
- versao : int

DependenciaSuporte
- suporte : Suporte
- versao : int

DependenciaGerenciador
- gerenciador : Gerenciador
- versao : int

Servico
+ getTipoServico() : int
+ getServicoInformacoes() : int
+ habilitar() : boolean
+ desabilitar() : boolean

Suporte
+ configurarServico() : void
+ sincronizarServico() : boolean

Gerenciador
+ getEntidade() : Entidade
+ getBO() : StandardBO

Figura 52 – Diagrama de pacotes do Gerenciador de Ferramentas

Uma ferramenta pode ter associado um tipo de artefato que a mesma consome ou gera. Por esta razão, o gerenciador de Ferramentas possui uma dependência com o Gerenciador de Artefatos, como expressa a Figura 53.

Figura 53 – Dependência do Gerenciador de Ferramentas

Uma ferramenta pode ainda ser utilizada para a comunicação com o ambiente, seja na comunicação com os serviços do ambiente ou na comunicação com outros usuários. Para que seja possível ao usuário registrar uma ferramenta para a comunicação, é necessário que esta

esteja apta para ser instanciada remotamente ou para receber mensagens. Para ser instanciada remotamente, é necessário que ela possua uma interface de comunicação. Para receber mensagens é necessário que a ferramenta possua métodos para receber / enviar mensagens e, também, a definição de sua mensagem. Estes dois tipos de ferramenta são apresentados na Figura 54.

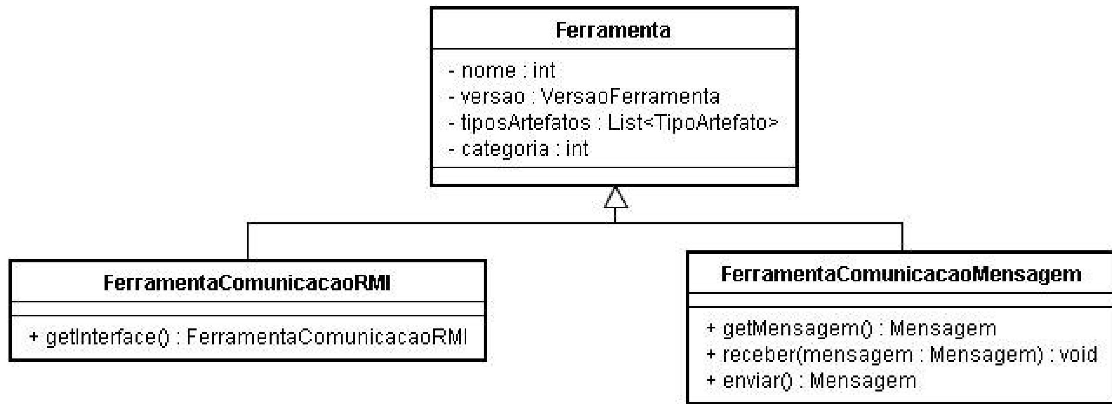

Figura 54 – Ferramentas com suporte a comunicação

No caso da comunicação ser através da troca de mensagens, a mesma deve possuir o método getMensagem aonde é possível conhecer o tipo de mensagem que a ferramenta pode receber.

Analisando o Gerenciador de Ferramentas como um ator do *Framework* aqui proposto, o mesmo teria os casos de uso apresentados na Figura 55.

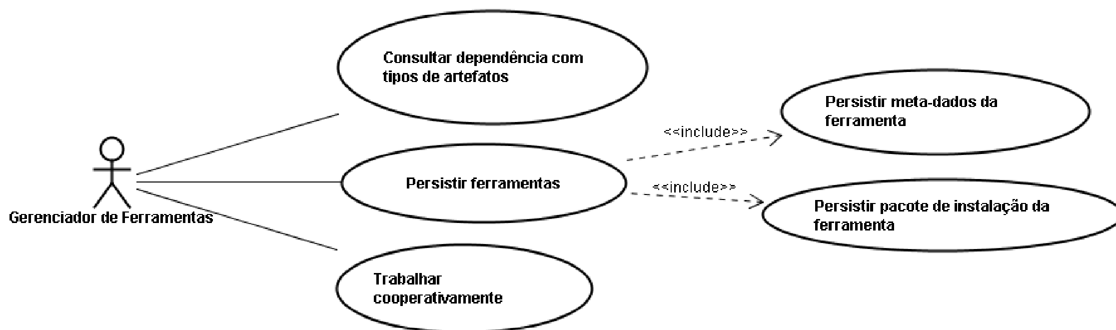

Figura 55 – Casos de uso do Gerenciador de Ferramentas

As ferramentas necessitam armazenar seus meta-dados junto ao suporte a persistência. Para a instalação, remoção e atualização de ferramentas, será utilizado o suporte à atualização. O Gerenciador de Ferramentas depende ainda do suporte à comunicação para as ferramentas que serão utilizadas para a comunicação no ambiente. A Figura 56 apresenta esta dependência.

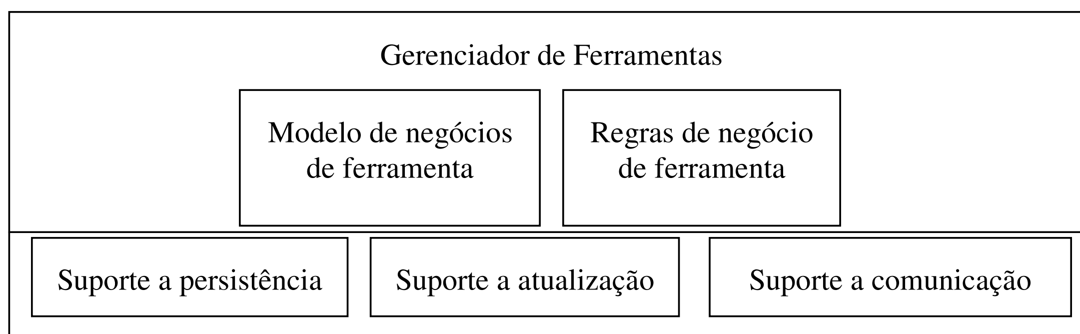

Figura 56 – Acesso do gerenciador de Ferramentas aos suportes

Para o trabalho cooperativo / colaborativo é necessário que a ferramenta possa utilizar o canal de comunicação. Para isto, é necessário que ela seja registrada junto ao suporte de comunicação. Por esta razão, suas regras de negócio devem incluir o registro da mesma junto ao suporte a comunicação. Além disto, a classe FerramentaBO deve incluir métodos para a ativação do suporte à comunicação como atualizarFerramenta, removerFerramenta, instalarFerramentas. A classe FerramentaBO é ilustrada pela Figura 57.

Figura 57 – Regras de negócios para Ferramentas: classe FerramentaBO

A interface do Gerenciador de Ferramentas é apresentado pela Figura 58.

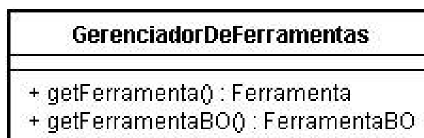

Figura 58 – Interface do Gerenciador de Ferramentas

3.5 Camada de infra-estrutura

A camada de infra-estrutura possui um conjunto de Serviços e Suportes para atender as necessidades dos gerenciadores. A comunicação da camada de negócios com a camada de infra-estrutura se dá através de seus suportes. Aos suportes cabe a tarefa de comunicar com os serviços. Isto porque toda estação de trabalho deverá possuir todos os suportes mas nem todas necessitam possuir os serviços. A Figura 59 apresenta o posicionamento da camada de infra-esturura no FRADE.

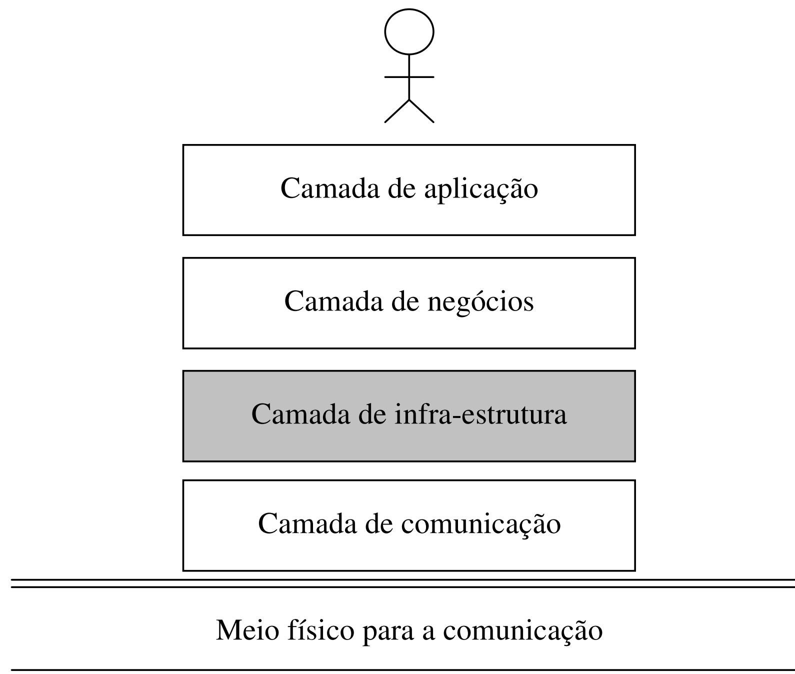

Figura 59 – Camada de infra-estrutura

A necessidade de infra-estrutura do ambiente foi definida pela necessidade dos gerenciadores da camada de negócio. As seguintes funcionalidades foram mapeadas:

- **Persistência**: A infra-estrutura de persistência deverá se encarregar de persistir as informações dos objetos de negócio e meta-dados envolvidos no desenvolvimento de *software*;
- **Artefato**: A infra-estrutura de artefato será encarregada da persistência dos itens de configuração de *software* e produtos em suas várias versões;
- **Acesso**: A infra-estrutura de acesso será responsável pela validação de usuários e serviços no ambiente e evitar possíveis intrusões;
- **Atualização**: Esta infra-estrutura será responsável por adicionar/remover aplicações no ambiente. Isto inclui a distribuição de novos recursos de *software* e, a atualização de recursos pré-existentes;
- **Comunicação**: A infra-estrutura de comunicação permitirá que um usuário do ambiente encontre outro usuário para que os mesmos possam realizar trabalhos cooperativos ou colaborativos;
- **Distribuição**: No caso de um participante do ambiente não possuir um serviço localmente instalado e configurado, é necessário que ele conheça outro participante que disponibilize este serviço. A infra-estrutura de Distribuição deverá configurar para cada participante onde estão disponíveis os serviços que não estão localmente.

O mapeamento destas funcionalidades pelos Gerenciadores da camada de negócios pode ser visto na Figura 60.

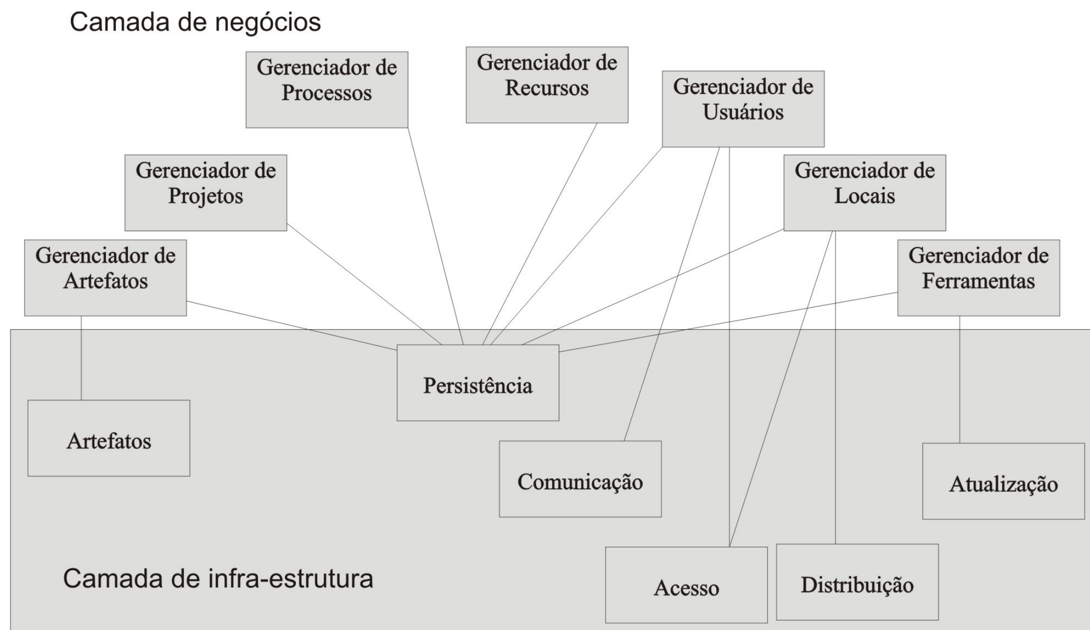

Figura 60 – Mapeamento dos gerenciadores de negócio para a camada de infra-estrutura

Toda necessidade de infra-estrutura será dividida em dois componentes: Um suporte e um serviço. O que é denominado neste trabalho de infra-estrutura de acesso, por exemplo, é a definição do suporte a acesso e do serviço de acesso.

Mapeadas estas funcionalidades, esta seção traz a definição do suporte e serviço que atenda a cada uma destas necessidades. Para cada funcionalidade haverá a definição de um suporte e um serviço.

TURNER, BUDGEN e BRERETON (2003) lembram que os métodos de descrição de um serviço atual sofrem de uma limitação: embora eles forneçam as informações técnicas que um cliente requisita para invocar um serviço, eles não podem descrever as funções que o serviço fornece semanticamente. Eles também deixam faltar a descrição da entrega do serviço e seus aspectos de negociação. Por exemplo, embora seja de fato a linguagem padrão para *web services*, WSDL descreve um serviço somente em termos de seus tipos de dados aceitáveis, métodos, mensagens, formatos, protocolo de transporte e URL sem, contudo, descrever o que exatamente o serviço faz. No caso deste *framework*, a definição da interface dos serviços deve incluir a definição do tipo do serviço e suas informações, como ilustra a Figura 61.

Figura 61 – Interface dos serviços do *framework*

Com esta interface possibilita-se a definição de outros serviços não previstos neste trabalho mas que poderão ser conectados ao ambiente. Independente da implementação desta interface, o ambiente conseguirá habilitar e desabilitar um novo serviço.

Toda estação de trabalho do ambiente poderá executar, localmente, qualquer serviço, porém este serviço local servirá, a princípio, apenas para ser utilizado por esta estação

compondo seu *workspace*. Conforme foi descrito no Gerenciador de Locais na seção 3.4.5, para que um serviço possa ser considerado um serviço do ambiente é necessário que o mesmo possua uma autorização para isto. Caso a estação possua um serviço local e o mesmo não esteja autorizado a ser um serviço do ambiente, caberá ao suporte atualizar o serviço local e também um dos serviços autorizados do ambiente. Isto garante que o usuário consiga trabalhar desconectado dos serviços do ambiente. Por esta razão, pode-se separar os serviços em serviços locais e serviços remotos.

Os suportes do ambiente são a "fachada" dos serviços remotos e locais para os gerenciadores. Os gerenciadores devem acessar os serviços através dos suportes do ambiente e, não diretamente. Isto porque o acesso direto ao serviço não garante a sincronização dos serviços locais e remotos do ambiente.

O acesso do suporte aos serviços remotos é feito no ambiente através da camada de comunicação conforme ilustra a Figura 62.

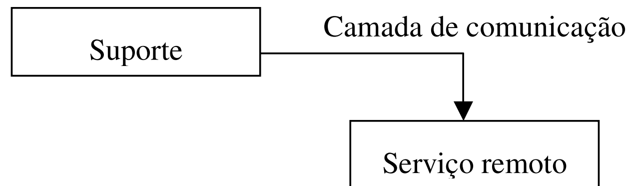

Figura 62 – Ativação de serviços remoto pelo suporte

A sincronização, entre o serviço remoto e o serviço local, deve ser feita de maneira transparente ao usuário. Uma vez que um suporte é ativado, o mesmo optará entre o serviço local, remoto ou ambos. A decisão depende de haver ou não uma instância local do serviço e haver ou não comunicação com a rede. Todos os suportes deverão implementar uma mesma interface que possuirá os métodos que irá escolher entre o serviço local e o remoto e, também, a sincronização entre o serviço local e o remoto. A implementação desta interface varia de suporte para suporte. Esta interface é vista a seguir na Figura 63.

Figura 63 – Interface dos suportes do *framework*

A camada de infra-estrutura possui o acesso a todos os serviços e suportes da mesma. Como nas camadas superiores, há uma interface que serve de fachada para a mesma. Esta interface é vista na Figura 64.

InfraEstrutura

+ getSuportePersistencia() : SuportePersistencia
+ getSuporteArtefato() : SuporteArtefato
+ getSuporteComunicacao() : SuporteComunicacao
+ getSuporteDistribuicao() : SuporteDistribuicao
+ getSuporteAtualizacao() : SuporteAtualizacao
+ getSuporteAcesso() : SuporteAcesso
+ getServicoPersistencia() : ServicoPersistencia
+ getServicoArtefato() : ServicoArtefato
+ getServicoComunicacao() : ServicoComunicacao
+ getServicoDistribuicao() : ServicoDistribuicao
+ getServicoAtualizacao() : ServicoAtualizacao
+ getServicoAcesso() : ServicoAcesso

Figura 64 – Interface da camada de infra-estrutura

A instanciação desta camada é feita de maneira dinâmica, em tempo de execução, de maneira que o usuário possa instalar, atualizar e remover serviços e suportes. Para isto é utilizado um modelo baseado em fábricas abstratas que será melhor explicado na seção 4.3 deste trabalho. Os suportes são obrigatórios a todas as estações do ambiente. A carga desta camada é feita através do arquivo XML descrito na Figura 65. Neste exemplo, o serviço local de persistência não está instalado.

Este arquivo configura a lista dos serviços locais. Isto não implica que os serviços remotos possuam a mesma implementação.

```
<config>
    <mapping key="disen.infraEstrutura.abstract.SuportePersistencia"
target="disen.infraEstrutura.suporte.Persistencia" />
    <mapping key="disen.infraEstrutura.abstract.SuporteArtefato"
target="disen.infraEstrutura.suporte.Artefato" />
    <mapping key="disen.infraEstrutura.abstract.SuporteComunicacao "
target="disen.infraEstrutura.suporte.Comunicacao" />
    <mapping key="disen.infraEstrutura.abstract.SuporteDistribuicao "
target="disen.infraEstrutura.suporte.Distribuicao" />
    <mapping key="disen.infraEstrutura.abstract.SuporteAtualizacao "
target="disen.infraEstrutura.suporte.Atualizacao" />
    <mapping key="disen.infraEstrutura.abstract.SuporteAcesso "
target="disen.infraEstrutura.suporte.Acesso" />
     <mapping
key="disen.infraEstrutura.abstract.ServicoPersistencia" target=""
/>
    <mapping key="disen.infraEstrutura.abstract.ServicoArtefato"
target="disen.infraEstrutura. servico.Artefato" />
    <mapping key="disen.infraEstrutura.abstract.ServicoComunicacao "
target="disen.infraEstrutura. servico.Comunicacao" />
    <mapping key="disen.infraEstrutura.abstract.ServicoDistribuicao "
target="disen.infraEstrutura. servico.Distribuicao" />
    <mapping key="disen.infraEstrutura.abstract.ServicoAtualizacao "
target="disen.infraEstrutura. servico.Atualizacao" />
    <mapping key="disen.infraEstrutura.abstract.ServicoAcesso "
target="disen.infraEstrutura. servico.Acesso" />
</config>
```

Figura 65 – Arquivo XML de configuração da camada de infra-estrutura

A configuração de serviços mínimo que uma estação poderá ter inclui: instalar o serviço local de nomes, acesso e comunicação. Isto porque estes serviços são fundamentais para o funcionamento da camada de comunicação.

3.5.1 Suporte à Persistência

A intenção do suporte à persistência é garantir que os produtos e as situações correspondentes aos produtos possam ser usados como parâmetros de entrada para as ferramentas (POHL et al., 1999).

O Suporte à persistência deve garantir processamento de consulta descentralizado de maneira a atender a busca por dados de produtos de maneira global ou em apenas alguns locais (BEN-SHAUL, KAISER, 1994). Para o acesso remoto, o suporte à persistência deve aceitar o suporte à transações imposto pelo serviço. Muitos modelos de transações atuais utilizam estas transações aninhadas e devem manipular a coordenação de transações interna ou externa ao banco de dados (CONRADI, LIU, HAGASETH,1995).

O suporte à persistência do ambiente está elaborado seguindo o padrão de projeto Core J2EE (*design pattern*) chamado DAO. Seguindo este padrão de projeto, os objetos de negócio serão os Business Objects (BO) e o acesso a estes objetos será feito através dos DAOs. Desta maneira, o suporte proposto neste trabalho possui uma coleção de DAOs, um para cada BO que há no ambiente, ou seja, um para cada gerenciador da camada de negócios. Todos os DAOs implementam a interface apresentada na Figura 66.

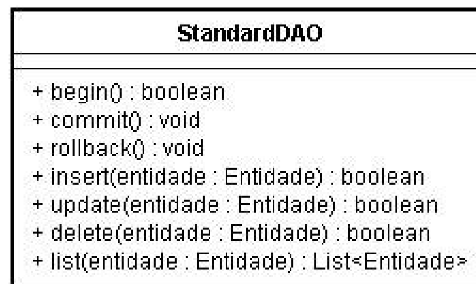

Figura 66 – Interface para suporte a persistência utilizando DAOs

Para que o serviço de persistência possa reconhecer quais são os objetos que podem ser armazenados, um tipo de dados chamado Entidade foi criado. Entidade para este *framework* é todo objeto que pode ser persistido. Nem todos objetos que se encontram na camada de negócios serão persistidos. Há objetos cuja finalidade é controlar o ambiente e que não precisam de persistência. A Trava para acesso a artefatos é um exemplo destes objetos de controle. A classe Entidade não possui, a princípio, métodos ou atributos definidos. A Figura 67 apresenta a classe Entidade.

Figura 67 – Classe Entidade: representa os objetos persistentes do *framework*

O suporte a persistência pode então ser visto como uma coleção de DAOs e os métodos de acessos a estes DAOs, como ilustra a Figura 68.

Figura 68 – Interface do suporte a persistência

Estes DAOs são acessados diretamente pelos BOs da camada de negócios. O suporte a persistência utiliza a comunicação síncrona da camada de comunicação com os DAOs do serviço de persistência. Por esta razão, os DAOs acessados pelos BOs no suporte a persistência estão, na verdade, invocando, remotamente, os DAOs que se encontram no Serviço de Persistência.

3.5.2 Serviço de Persistência

A persistência é uma característica obrigatória de uma arquietura de *software* uma vez que, é necessário, separar a utilização dos dados de seu armazenamento. Isto permite, que a estrutura de dados possa ser modelada da melhor maneira possível sem se preocupar de como isto será feito ou como o dado será armazenado (BROWN, 1988).

Um dos grandes problemas de um serviço de persistência é a utilização do mesmo por vários usuários simultâneos. Esta utilização simultânea é chamada concorrência. Uma solução trivial para a persistência é não ter controle de concorrência. Neste caso, o acesso aos dados é sempre possível e, caso dois usuários tentem acessar o mesmo dado, um usuário pode reescrever o trabalho de outro (BARGHOUTI, 1992).

Outra solução é o modelo tradicional de transações para o controle de concorrência utilizando o travamento em duas fases. Caso dois usuários tentem acessar o mesmo dado, um deles será bloqueado até que o outro termine seu acesso (BARGHOUTI, 1992). A transação é um mecanismo de abstração que permite que um grupo de operações de atualização possa ser tratado como uma ação única e atômica. Isto significa que, ou toda operação ocorre, ou nada ocorre. Transações existem em bases de dados e sistemas de arquivo (BROWN, 1988).

Uma terceira solução é permitir cópias paralelas dos dados a serem acessados e mesclar as alterações feitas por ambos os usuários. Desde que ambos os usuários não façam alteração na mesma parte do código o repositório se encarrega de mesclar a versão antiga com ambas as novas versões modificadas (BARGHOUTI, 1992).

É válido lembrar que nenhuma destas alternativas apresentadas traz uma solução para o desenvolvimento cooperativo por não levar em conta a interferência entre desenvolvedores durante a alteração de um dado (BARGHOUTI, 1992).

Portanto seria necessário um controle de concorrência flexível que permitisse aos usuários trabalharem simultaneamente sobre o mesmo objeto de maneira que as alterações sofridas por este fossem refletidas para todos os usuários que estão cooperando

(BARGHOUTI, 1992). Entre os ambientes estudados apenas o Shamu, (BARTHELMESS, 2003) possui tal suporte.

O suporte transacional em ADS foi extensivamente estudado em projetos como Marvel, Merlin, Adele, EPOS e SPADE (BEN-SHAUL, HEINEMAN, 1996).

Quando um ADSs é descentralizado, o mesmo deve fornecer uma arquitetura para a interoperabilidade entre vários ADSs. Com um ADS único, a atomicidade é uma característica que é aplicada apenas na base de dados do sistema. Quando o ambiente é descentralizado, a atomicidade deve ocorrer em todas as bases de dados que compõe o ADS de maneira que este conceito aplique-se ao ambiente como um todo (BEN-SHAUL, HEINEMAN, 1996).

O Serviço de Persistência do FRADE conta com os mesmos métodos de acesso aos objetos que o seu suporte, conforme descrito na Figura 69.

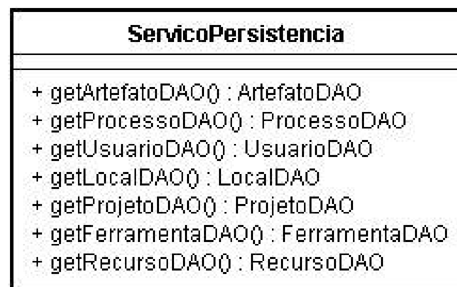

Figura 69 – Interface do Serviço de Persistência

Seguindo uma tendância adotada por vários ADSs, o serviço de persistência deste *framework* possui uma implementação baseada em bancos de dados relacionais. Para sua implementação utilizou-se o *framework* Hibernate. A opção por uma implementação utilizando este *framework* se deu devido à facilidade que há para a mudança de banco de dados sem que haja alteração em seus métodos. Graças à utilização do Hibernate, a persistência pode utilizar todos os bancos de dados relacionais suportados por este.

Os bancos de dados tradicionais têm um conceito de consistência estrito do acoplamento para serialização de transações. Para bancos de dados distribuídos e conectados por rede, existe o protocolo *two-phase commit* (CONRADI, LIU, HAGASETH,1995).

Devido a implementação deste *framework* não incluir a implementação de um banco de dados relacional, este suporte deve contar com a utilização de um banco de dados externo ao ambiente. Este banco de dados externo está, para o nível arquitetural, na camada de infra-estrutura.

Uma classe chamada HibernateDAO foi criada para criar o acesso dos DAOs à base de dados utilizando este *framework*. Os DAOs do ambiente implementam a interface StandardDAO e estendem HibernateDAO. A classe HibernateDAO é apresentada na Figura 70.

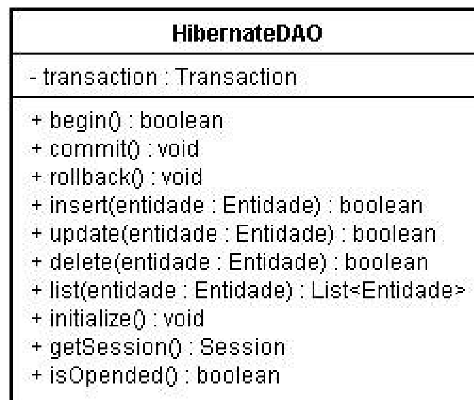

Figura 70 – Serviço a persistência: Classe HibernateDAO

Os métodos do HibernateDAO implementam todos os métodos da interface StandardBO. Isto permite que os DAOs do serviço de persistência não precisem implementar método algum. A criação de todos os DAOs, um para cada gerenciador, é justificada para o caso de alguma política de acesso ou alteração de dados ser tratada também no servidor. Como não está no escopo deste trabalho definir as políticas de acesso aos dados em um ADS, o código ou diagrama de classes destes DAOs não serão aqui apresentados. Um trecho do código na classe HibernateDAO é apresentado na Figura 71.

```
        public boolean insert(Entidade obj) throws Exception {
                boolean result = false;
                try {
                        boolean iniciouTransacao = begin();
                        session.save(obj);
                        obj = (Entidade) session.merge(obj);
                        if (iniciouTransacao) {
                                commit();
                        }
                        result = true;
                } catch (Exception e) {
                        try {
                                rollback();
                        } catch (Exception ex) {
                        }
                        throw e;
                }
                return result;
        }
}
```

Figura 71 – Método insert da classe HibernateDAO

O Hibernate utiliza arquivos XML de mapeamento objeto-relacional para sua configuração. Este mapeamento permite definir qual tabela armazenará qual Entidade e qual campo da tabela corresponde a qual atributo do objeto. Um exemplo deste arquivo é apresentado na Figura 72.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="disen.artefato.bean">
    <class name="Artefato" table="artefato">
        <id name="id" type="java.lang.Integer" column="id">
            <generator class="sequence">
                <param name="sequence">seq_artefato_id</param>
            </generator>
        </id>
        <property name="nome" type="java.lang.String"/>
        <property name="descricao" type="java.lang.String"/>
        <property name="estado"
type="disen.artefato.bean.EstadoArquivoUserType"/>
        <many-to-one name="tipoArtefato"
        class="TipoArtefato"
        column="tipo_artefato"
        lazy="false"/>
    </class>
</hibernate-mapping>
```

Figura 72 – Mapeamento objeto-relacional da classe Artefato

Nem todos os usuários precisarão armazenar localmente todas as entidades do Gerenciador de Objetos. Por esta razão, o serviço de persistência local pode não contar com a implementação de todos os DAOs. Há a possibilidade de utilizar localmente apenas os DAOs necessários para o cumprimento de determinadas tarefas como, por exemplo, os objetos necessários ao Suporte a Locais e os dados referentes a agenda de um participante.

Para evitar que os usuários necessitem instalar localmente uma base de dados, a persistência local pode ser feita em arquivos textuais. O acesso à base de dados e capacidade de exportar estes dados para arquivos textuais é descrito por CHAMBERS e LANG (1999).

Para evitar o acesso direto aos dados do serviço local, uma opção é a utilização de criptografia para o armazenamento local. A codificação e decodificação dos dados são tarefas do serviço que deve utilizar a validação do Serviço de Acesso para garantir o acesso de um usuário aos dados.

3.5.3 Suporte a Artefatos

Segundo SOKOLSKY e KAISER (1991), todo ADS possui um repositório de dados, nem que este repositório seja apenas o sistema de arquivo nativo aonde o artefato de *software* está fisicamente gravado. Por exemplo, enquanto o “repositório” UNIX tem comandos como mv, cp, ln, tar, etx, o repositório Marvel possui add, delete, copy, move, rename e join.

Segundo VILLELA et al. (2003), artefato é qualquer elemento produzido pelo homem e não por causas naturais, podendo exercer diferentes papéis em uma organização, tais como o de insumo ou produto de uma atividade. Na área de informática, artefatos podem ser classificados, de acordo com a sua natureza, em bens, documentos e componentes. Bens

podem ser classificados em bens de usufruto e bens de produção, sendo que bens de produção podem ainda ser classificados em hardware, *software* e equipamentos. Um componente pode ser um componente de hardware, um componente de *software* ou um objeto de uso comum.

Para este trabalho, artefato é uma visão genérica de todos os Itens de configuração de *software* (ICS) criados no desenvolvimento de um sistema computacional, sejam estes ICS diagramas, documentos ou códigos fonte.

O processo de transformação dos diversos artefatos de *software* que compõem um projeto no sistema executável é automatizado por um sistema de controle de construção e liberação de artefatos (LOPES, MURTA, WERNER, 2005). É através do Gerenciador de Configuração de *Software* (GCS) que usuários remotos devem poder compartilhar dados e artefatos através da rede do ADS (BEN-SHAUL, KAISER, 1994).

Para permitir versões de artefatos e controle sobre modificações de ICS por usuários um GCS atua sobre um Sistema de controle de versões (SCV). O SCV permite que os itens de configuração sejam identificados, segundo critérios estabelecidos pela função de identificação da configuração, e que eles evoluam de forma distribuída e concorrente, porém disciplinada (LOPES, MURTA, WERNER, 2005). Para isto, há no SCV um repositório compartilhado, normalmente remoto, e cada usuário do SCV pode possuir uma cópia local deste repositório.

A cópia local do repositório remoto é, comumente, chamada de *workspace*. Apesar deste trabalho seguir um conceito estendido de *workspace*, a definição comumente aceita é que um *workspace* é o local aonde as ferramentas de engenharia de *software* são executadas. Os objetos encontrados em um *workspace* variam de sistema para sistema mas, de uma forma geral, incluem arquivos e diretórios. Normalmente, apenas uma versão de cada arquivo existe em um *workspace*, uma vez que a maioria das ferramentas não trabalha simultaneamente com mais de uma versão do mesmo artefato (ESTUBLIER, CASALLAS, 1994). Além disto, um *workspace* deve possuir as propriedades de transação ACID: Atômica, Consistente, Isolada e Durável (Atomicity, Consistency, Isolation and Durability) (ESTUBLIER, CASALLAS, 1994).

A Figura 73 apresenta a interface do Suporte a Artefatos. Ela contém os métodos para enviar, receber, listar e remover arquivos no serviço de artefato. É importante notar que estes métodos não trabalham sobre a classe arquivo do Gerenciador de Artefatos mas sobre os arquivos físicos da estação de trabalho.

```
SuporteArtefato
---
+ listarArquivos() : List<String>
+ listarVersoes() : List<String>
+ receberArquivo(nome : String) : byte[]
+ enviarArquivo(arquivo : java.io.File, conteudo : byte[]) : boolean
+ adicionarArquivo(arquivo : java.io.File) : boolean
+ removerArquivo(arquivo : java.io.File) : boolean
+ compararVersoes(nome : String) : boolean
```

Figura 73 – Interface do Suporte a artefatos

Por esta razão, receber um arquivo significa receber o seu conteúdo, ou seja, os bytes que compõe este arquivo.

Os métodos requisitarTrava() e liberarTrava() do serviço de artefato é chamado internamente nos demais métodos do suporte a artefatos, não tendo o programador opção de

requisitar uma trava diretamente. Quando um usuário requisita um artefato, o suporte se encarrega de requisitar a trava para o mesmo. O serviço de artefato é explicado a seguir.

3.5.4 Serviço de Artefato

Segundo OLIVEIRA, MURTA e WERNER (2004), a Gerência de Configuração de *Software* (GCS) consiste numa abordagem disciplinada para controlar o processo de desenvolvimento e manutenção do *software*. A GCS é uma das disciplinas mais maduras e utilizadas da engenharia de *software*. No entanto, suas potencialidades são ainda pouco exploradas, pois somente o controle de versões sobre código fonte tem sido efetivamente adotado nos projetos.

Conforme citado no Capítulo 2 deste trabalho, ao estabelecer um processo definem-se, entre outras coisas, a seqüência de atividades, as ferramentas a serem utilizadas, os papéis dos desenvolvedores e os artefatos consumidos e produzidos. Estes artefatos são recuperados em repositórios de componentes disponíveis, local ou remotamente, através de mediadores e ontologias capazes de integrar as informações armazenadas em repositórios distribuídos (WERNER et al., 2002).

Segundo SOKOLSKY e KAISER (1991), os repositórios podem ser:

1. Um grupo de arquivos desorganizados no mesmo diretório, aonde todos os dados são representados como arquivos e o acesso aos mesmo é feito apenas pelo sistema operacional;
2. Um grupo de arquivos estruturados em diretórios, aonde um item de dado é representado, não apenas por um arquivo, mas por um diretório e por todos os arquivos nele contido. O acesso a estes itens de dados é feito através do sistema de arquivos;
3. Um grupo de arquivos organizados em diretórios, aonde os itens de dados são os diretórios. O ADS fornece meios para navegar nestes itens, reconhecendo os arquivos que contém dados ou meta-dados e, diferenciado-os para a camada de apresentação;
4. O repositório contém o sistema de arquivos e também uma base de dados para o armazenamento dos meta-dados. Os meta-dados possuem a localização dos arquivos que compõem o item de dados. O ADS é responsável por fazer a junção do meta-dado com o dado em arquivo e, utiliza o banco de dados e o sistema de arquivo para esta navegação;
5. O repositório utiliza princípios de orientação a objetos e o sistema de arquivo para o armazenamento dos dados. Mais que os meta-dados dos itens de dados, o repositório orientado a objetos armazena o relacionamento entre os itens de dados e suas dependências, fornecendo uma interface de acesso completa a conjuntos complexos de dados.

BARGOUTHI et al. (1996) afirmam que, a arquitetura de um ADS tipicamente inclui um repositório que armazena os dados do produto, os dados do processo ou ambos. As alternativas utilizadas para o repositório são tecnologias de base de dados comerciais, repositórios com propósitos específicos e arquivos estruturados.

O serviço de artefatos tem a função de persistir artefatos de *software* no ambiente, ou seja, arquivos fontes, diagramas, imagens, arquivos de configuração e demais ICS (itens de

configuração de *software*) em suas várias versões. Não é responsabilidade do serviço de artefatos a atualização dos dados sobre o artefato junto aos objetos de negócio do ambiente.

GULLA, KARLSSON e YEH (1991) reforçam o conceito de GCS especificando as seguintes caracterísitcas desejáveis:

- **Suporte ao trabalho em paralelo:** Não é realístico supor que todo o trabalho em um produto de *software* de grandes dimensões é feito seqüencialmente. Por esta razão, o sistema de controle de versões deve apoiar trabalho paralelo no mesmo componente. O problema é a propagação seletiva de alterações graduais em todos as divisões do projeto (*branchs*). Os usuários necessitam controlar quando eles irão visualizar as alterações paralelas. A possibilidade de inclusão de alterações graduais torna difícil a tarefa de encontrar simples erros de interferência;
- **Especificar o escopo da alteração:** Quando uma alteração é implementada, isto pode ser entendido que ela afetará certas versões, ela é ortogonal para outras alterações e depende de outras alterações. Entretanto, para outras alterações, a mudança pode não aparecer de maneira ortogonal. No GCS uma alteração só aparecerá na folha da divisão do projeto (*branchs*) e será invisível nas demais divisões. Uma mudança implementada por uma compilação condicional só alterará quem possuir sua condição contemplada;
- **Especificação de versões imutáveis:** Quando um produto é entregue, é bem provável que seja necessário recuperar sua versão de entrega para reproduzir os erros encontrados pelos usuários. Isto é comumente feito através do congelamento das revisões na árvore de artefatos do repositório. A revisão exata de cada componente que foi feito para a entrega do sistema deve ser armazenada separadamente.

Quando tratamos de ADS distribuídos e sua GCS, o primeiro aspecto a se considerar é o nível da distribuição. As duas alternativas óbvias são centralizar totalmente os serviços ou distribuí-los totalmente. No primeiro caso, os clientes podem ser mínimos e todo o controle e operações são feitos no servidor. No segundo caso, não deve haver um servidor dedicado mas apenas clientes com todo o controle e operações do produto a serem executadas no cliente e, o compartilhamento sendo feito diretamente através da comunicação entre os clientes. Há ainda, a possibilidade de uma abordagem híbrida aonde os clientes são responsáveis por atividades de longa duração e, os servidores são responsáveis por tarefas de curta duração como controle e sincronização. (BEM-SHAUL, KAISER, HEINEMAN, 1992).

Devido à possibilidade de alteração de artefatos binários como diagramas, imagens e documentos formatados (como o PDF), o serviço deverá permitir que haja uma trava de alteração destes documentos. Isto porque não é possível verificar as alterações que os mesmos sofreram utilizando métodos como o DIFF. Algumas ferramentas de controle de versões, como o SubVersion e o Rational ClearCase, utilizam esta técnica de trava para a atualização destes documentos binários.

Segundo FUJIEDA e OCHIMIZU (2003), a maioria dos projetos *open source* utiliza CVS para seu gerenciamento de configuração de *software*. O CVS permite a atualização do repositório através da Internet, porém, não fornece mecanismo para travar um arquivo. Isto permite que os usuários alterem a mesma parte do repositório, trabalhem de forma concorrente e depois resolvam a inconsistência em seus trabalhos.

Quando se versiona artefatos de alto nível de abstração (modelos UML, por exemplo), utilizando modelo de dados baseado em sistemas de arquivos, os resultados obtidos são pouco

satisfatórios. O problema está relacionado à granularidade, porque nesse modelo de dados um arquivo é um item de configuração indivisível. Portanto, um modelo que possui vários diagramas é versionado como um único arquivo e é impossível acessar um determinado diagrama sem acessar os demais (OLIVEIRA, MURTA, WERNER, 2004).

Quanto à utilização de travas para a garantia de transação sobre artefatos, há dois cenários. O cenário pessimista trava o acesso do usuário a um objeto até que o mesmo possua a trava que garanta a manipulação do objeto. Esta é a maneira mais simples e garantida de alcançar consistência absoluta. O cenário otimista permite que os usuários atualizem objetos sem que haja uma checagem de conflitos tornando a interação com os objetos algo maus natural. Todavia, quando um conflito ocorre, é necessário que o mesmo seja reparado. Isto não apenas torna o sistema complexo, mas, também, deixa o usuário confuso quanto as suas ações no sistema (LEE, LIM, HAN, 2002).

Quando há uma proposta de trabalhar vários serviços de artefatos ao mesmo tempo, a utilização de um controle de transação se torna ainda mais complexa. Para resolver este problema, surge a solução da utilização de réplicas.

A replicação de repositórios é um mecanismo importante para o gerenciamento de configuração distribuído. Esta replicação pode ser parcial ou total (FUJIEDA, OCHIMIZU, 2003).

Os sistemas de gerenciamento de configuração BitKeeper e ClearCase Multisite suportam um estilo de desenvolvimento com réplicas (FUJIEDA, OCHIMIZU, 2003). O BitKeeper é uma ferramenta comercial que se torna uma exceção notável por ser uma das poucas ferramentas proprietárias utilizadas gratuitamente pelos desenvolvedores do Linux como sistema de gerenciamento de configuração (GERMAN, 2003). O ClearCase Multisite é uma ferramenta proprietária da Rational / IBM e possui capacidade para o versionamento de diagramas gerados pela suíte de ferramentas da Rational, como, por exemplo, o Rational Rose.

De qualquer maneira, o acesso aos dados do produto e aos dados do controle deve ser sincronizado para que sua consistência seja mantida. Esta sincronização deve prever acesso concorrente tanto aos dados de produto, quanto aos dados de controle (BEN-SHAUL, KAISER, HEINEMAN, 1992).

Os gerenciadores de configuração de *Software* são necessários para todos os tipos de ferramentas CASE, porém, este tipo de suporte está tipicamente focado nas ferramentas CASE baixo nível provendo uma boa infra-estrutura para os artefatos código fonte (OLIVEIRA, MURTA, WERNER, 2005).

Todavia, devido ao crescimento da complexidade do desenvolvimento de *software*, o suporte a GCS é necessário para as ferramentas CASE de alto nível. O desenvolvimento guiado por Modelos está emergindo como uma técnica promissora para o controle de complexidade. Diante desta evolução, a infraestrutura atual de SCM não provê o suporte adequado à evolução dos artefatos baseados em modelos (OLIVEIRA, MURTA, WERNER, 2005).

Um primeiro pensamento seria adaptar as técnicas de GCS, formalmente aplicadas ao código-fonte, a este novo contexto. Todavia, a infraestrutura atual de GCS não é adaptada ao alto nível de granularidade das ferramentas CASE de alto nível.

O padrão de armazenamento dos GCS atuais é centrado em arquivos. Porém, nem sempre é possível mapear um artefato de alto nível em um arquivo. Em um mesmo arquivo é possível encontrar, por exemplo, diagramas de classe, casos de uso e seqüência. Por esta

razão, a relação de Item de Configuração de *Software* para arquivos não é tão simples quando tratado os artefatos de ferramentas CASE de alto nível (OLIVEIRA, MURTA, WERNER, 2005).

Os GCSs que utilizam um modelo baseado na estrutura do sistema de arquivos consideram os arquivos e diretórios como ICS. Esta abordagem divide os ICS em três categorias: textuais, binários e compostos. Os compostos são os diretórios que podem agregar ICS textuais ou binários. Os textuais são os que merecem um maior destaque pois podem sofrer a execução e a manipulação de operações de controles básicos do GCS como *diff*, *patch* e *merge*. Os binários são apenas controlados mas não internamente manipulados pelos GCS pois sua estrutura interna é opaca para estas ferramentas. Por esta razão, os arquivos textuais são vistos como caixas brancas para o GCS e os binários como caixas pretas. Além disto, os arquivos textuais são vistos de uma forma genérica. O mesmo merge utilizado para um arquivo Java é utilizado para um arquivo XML sendo que a estrutura interna destes arquivos é distinta (OLIVEIRA, MURTA, WERNER, 2005).

O desenvolvimento de *software* centrado em processo utiliza vários tipos de artefatos como descrições de caso de uso, diagramas de classe, diagramas de casos de uso, diagramas de seqüência, código, planos de testes, etc. Todos estes artefatos deveriam ser controlados de maneira consistente de maneira a fornecer um snapshot do sistema em diferentes momentos do desenvolvimento e manutenção (OLIVEIRA, MURTA, WERNER, 2005).

Este *framework* não propõe o desenvolvimento de um GCS. Por não propor uma implementação de ferramenta de controle de versões, é necessário utilizar uma aplicação externa como o CVS ou SubVersion. O Serviço de Artefatos proverá para o ambiente os arquivos que o repositório de artefatos está armazenando e, também, chaves de controle para o acesso aos mesmos. A interface do Serviço de artefatos é apresentada na Figura 74.

Figura 74– Interface do serviço de artefatos

O serviço de artefato deverá criar um mapeamento entre seus métodos e os comandos da ferramenta GCS sobre o qual o serviço está instalado. Um exemplo deste mapeamento é apresentado na Tabela 6.

Tabela 6 – Mapeamento entre os métodos do serviço de artefatos e o repositório CVS

Serviço de Artefato	CVS
listarArquivos	list
listarVersoes	diff
adicionarArquivo	add
removerArquivo	remove
receberArquivo	update
enviarArquivo	Commit
pegarTrava	Não possui esta funcionalidade
liberarTrava	Não possui este funcionalidade

Conforme ilustrado na Tabela 6, o Serviço de Artefatos local (*workspace*) poderá utilizar um diretório mapeado para armazenar arquivos localmente e, não necessariamente, uma ferramenta de controle de versões. Esta opção de armazenar os artefatos localmente vai ao encontro da necessidade de os usuários do ambiente em manterem localmente artefatos em fase de construção.

Segundo LEE, LIM e HAN (2002), a existência de um repositório local se justifica pois quando a comunicação atrasa ou há um aumento no número de usuários, o desempenho da interatividade diminui devido ao alto tempo de resposta.

Além disto, um repositório local é necessário para a cooperação de maneira que um usuário possa trabalhar de maneira completamente isolada ou de maneira pública. Desta maneira, os usuários podem mover os artefatos do espaço compartilhado para seus próprios *workspace*. Após encerrada a edição do artefato, este é enviado ao repositório compartilhado que irá persistí-lo (TATA, GODART, WIIL, 2002).

O serviço de persistência local não é sincronizado com o serviço de persistência remoto e a sincronização de ambos deve ser feita utilizando a camada de infra-estrutura. Quando o suporte buscar o respositório remoto para atualizar seus artefatos o mesmo deve, primeiramente, se encarregar de sincronizar o repositório remoto a partir do repositório local garantindo assim a sincronização entre ambos.

3.5.5 Suporte à Atualização

A atualização do ambiente, remoção de aplicações e instalação de novas aplicações e gerenciadores é responsabilidade da infra-estrutura de atualização. A infra-estrutura de atualização é responsável por manter o repositório de atualizações em um local, copiar novos pacotes, instalá-los e integrá-los ao ambiente.

O Suporte de Atualização deverá prover para as estações de trabalho do FRADE listas de pacotes instalados localmente e, também, listas de pacotes disponíveis no ambiente. Deverá prover também pacotes de *software* (arquivos) para a instalação ou atualização de novos módulos no ambiente.

A atualização do ambiente pode ocorrer em todos os níveis do ambiente e não apenas em suas ferramentas. A atualização pode tratar Gerenciadores, Serviços, Suportes e ferramentas. Por esta razão, a infra-estrutura de atualização tratará todos por **módulos**. A atualização / instalação de cada um destes módulos é distinta e tratada individualmente. A instalação do serviço de persistência, por exemplo, deverá conter a instalação e/ou configuração do banco de dados para a persistência no servidor. No caso do usuário, o instalador poderá configurar um primeiro servidor / projeto no seu Gerenciador de Local para o acesso do usuário.

A instalação de uma nova ferramenta no ambiente deverá seguir os seguintes passos:

1 – Usuário pede para o instalador verificar quais módulos se encontram disponíveis no servidor;

2 – Instalador fornece lista de módulos disponíveis no servidor com suas versões listando ainda quais já foram instalados e quais não;

3 – Usuário escolhe, pela GUI do instalador, qual ferramenta será adicionada;

4 – Instalador pede a ferramenta (arquivo);

5 – Suporte grava a ferramenta em disco (*workspace*);

6 – Instalação da ferramenta é executada e, por causa de seu padrão de código, se instala no ambiente podendo se tornar disponível em barras de menu, barras de ferramenta, arquivos de *help* e demais itens da GUI Cliente;

7 – GUI Cliente pode ser reiniciada para que a alteração tenha efeito.

O Suporte à Atualização atuará diretamente sobre os arquivos de configuração do ambiente. Os arquivos para a configuração das camadas e de cada camada internamente já foram apresentados (Figura 7, Figura 13, Figura 65). Além destes, um novo arquivo de configuração será criado para mapear as ferramentas instaladas no ambiente. Este arquivo será chamado pelo método loadModules da aplicação principal e se encarregará de integrá-lo a aplicação. A estrutura deste arquivo é apresentada pela Figura 75.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ferramentas>
    <mapping key="DiManager" target="DiManager.jar" />
    <mapping key="Requisite" target="Requisite.jar" />
(...)
    <mapping key="chat" target="Chat.jar" />
</ferramentas>
```

Figura 75 – Arquivo de configuração das ferramentas instaladas no ambiente

A estrutura de configuração do ambiente com a instalação de módulos é definido pela Tabela 7.

Tabela 7 – Configuração e atualização do ambiente e seus arquivos de configuração

Módulo	Arquivo de configuração
Aplicação principal	Aplicação.xml (Figura 7)
Gerenciadores	Negócios.xml (Figura 13)
Infra-estrutura (serviços e/ou suportes)	Infra.xml (Figura 65)
Ferramentas	Ferramentas.xml (Figura 75)

O arquivo a ser instalado é um arquivo compactado de extensão .jar. Este arquivo compactado deverá possuir um jar que é a aplicação e um arquivo XML para a sua definição. Este pacote poderá ainda ter uma segunda aplicação que funcionará como instalador do módulo. O conteúdo do pacote é ilustrado pela Figura 76.

Figura 76 – Conteúdo do pacote de instalação dos módulos

O descritor XML do pacote definirá:

- Dependências;
- Instalador;
- Nome do programa;
- Versão;
- Data;
- Tipo do módulo;
- Pacote e classe do módulo;
- Instalação de menu;
- Desenvolvedor;
- Ícone para barra de ferramentas.

Nem todos os módulos conterão todos estes descritores pois nem todos precisarão possuir instalador, ícones ou menus. Há ferramentas cuja instalação consiste em apenas adicionar seu pacote ao *workspace*, por exemplo, uma ferramenta de chat e serviços que não possuam GUI para interação com o usuário e, por esta razão, não necessitam de ícones ou menus.

Além de atualizar as estações de trabalho, o suporte à atualização pode enviar novos módulos para o repositório de atualizações. Isto significa que a atualização do repositório de atualização do ambiente poderá ser feita através do próprio ambiente. Certo de que nem todo usuário deverá estar autorizado a adicionar módulos ao repositório há a necessidade, que não será atendida por este trabalho por estar além do escopo deste, de definir as políticas de atualização do repositório. A intenção de atualizar o repositório diretamente é facilitar o intercâmbio entre ferramentas em uma equipe de desenvolvimento de *software*.

A interface do Suporte à Atualização é apresentada na Figura 77.

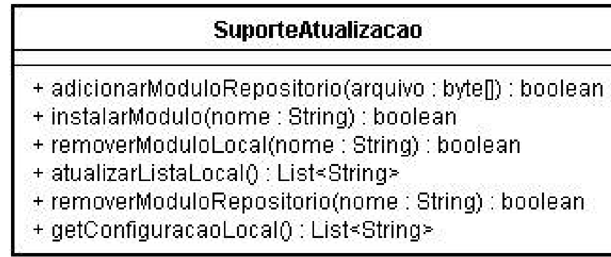

Figura 77 – Interface do suporte a atualização

Em todas as estações este suporte contará com um diretório para manter uma cópia dos pacotes de atualização que estão sendo utilizados. Até a sua instalação o pacote deverá ser mantido local e poderá ser removido após sua instalação.

3.5.6 Serviço de Atualização

Para que seja possível disponibilizar novas ferramentas para os usuários do ambiente, é necessário que haja um servidor responsável por um repositório de módulos[6]. O serviço de atualização é o responsável pela ESD (*Eletronic Software distribution*) no ambiente.

Com a utilização deste serviço, será possível aos participantes de um grupo de projeto compartilhar ferramentas ou, ainda, possuir uma configuração padrão de Usuário estabelecida pelo Gerente de projeto. Para isto, é necessário adicionar políticas ao serviço/suporte de atualização que permitam, por exemplo que o repositório de módulo só disponibilize uma ferramenta caso o usuário desempenhe papel de gerente no projeto.

A infra-estrutura local do serviço de atualização, chamada repositório de atualização, é um diretório do sistema de arquivos. A estrutura do repositório de atualização deverá seguir a organização em camadas do ambiente com uma identificação explícita do que é cada pacote. Uma visão geral deste repositório é ilustrada na Figura 78.

[6] Como citado anteriormente a infra-estrutura de atualização trata gerenciadores, suportes, serviços e ferramentas como módulo, indistintamente. Independente do papel do módulo dentro do ambiente o mesmo deverá possui seu arquivo de configuração que o descreve como módulo.

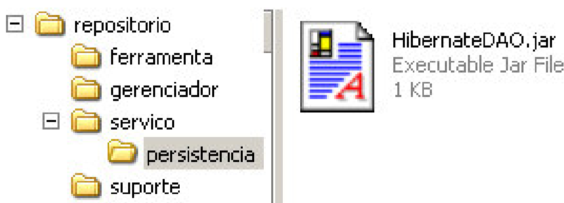

Figura 78 – Estrutura do repositório de atualizações

A interface do Serviço de Atualização é similar à do suporte à atualização e é apresentada na Figura 79

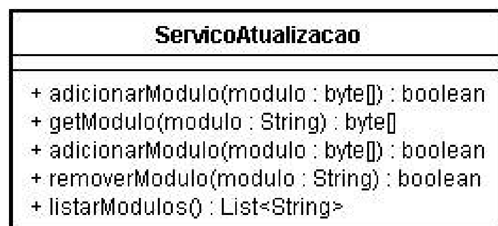

Figura 79 – Interface do Serviço de Atualização

Os métodos desta classe poderão ser implementados de forma a verificar a permissão de um usuário quando o mesmo requisitar uma destas operações. A adição de um novo módulo, por exemplo, pode requerer uma autorização especial para isto, evitando assim que ferramentas não autorizadas sejam disponibilizadas no ambiente.

3.5.7 Suporte a Acesso

A infra-estrutura de acesso possui em suas funções o controle de acesso e monitoramento do ambiente. Mais do que garantir a legitimidade de um usuário ao ambiente, o suporte a acesso precisa garantir que o mesmo tenha acesso apenas às informações que lhe são pertinentes.

Em um modelo de cooperação, os usuários possuem as seguintes características (REIS, 1998):

- Os tipos de usuários do ambiente são:
  - **Super-usuários:** São os usuários de mais alto nível definidos no ambiente, podendo realizar qualquer atividade sobre as informações armazenadas;
  - **Gerentes:** Para cada grupo do ambiente é permitida a promoção de um usuário comum (membro do grupo) à gerente do grupo. Aos gerentes de um grupo é permitida inclusão e exclusão de membros, assim como a criação de novos gerentes e controle dos usuários que compõem um grupo;
  - **Usuários comuns:** São os usuários que não são gerentes ou superusuários podendo pertencer aos grupos do ambiente.
- Todo usuário possui um identificador associado. O identificador é a informação utilizada pelas operações definidas por todo o ambiente para individualizar um usuário

específico. Portanto, não é permitida a existência de usuários ou grupos com identificadores iguais;

- Todo usuário é associado a um nome e uma senha. Aos super-usuários do ambiente é permitida a alteração de uma senha para um usuário;
- Os usuários podem ser organizados em grupos e papéis. Nas organizações de desenvolvimento de *software* os grupos de desenvolvedores associam um conjunto de usuários que possuem características comuns. Os papéis distintos no ambiente definem distintos níveis de acesso.

Dentro deste *framework*, seguindo o trabalho de (REIS, 1998) e (ENAMI, 2006), os usuários serão divididos entre Gerentes de Local, Gerentes e Desenvolvedor. De uma maneira genérica, estes usuários são chamados de participantes do ambiente e seus casos de uso variam em relação a sua participação em um projeto ou em relação a configuração do ambiente. Diante disto, um mesmo participante pode ser Desenvolvedor em um projeto e Gerente de projeto em outro. Uma melhor definição destes níveis de acesso, segundo a proposta do FRADE segue abaixo.

Desenvolvedor

Desenvolvedor é a denominação genérica para os papéis desempenhados pelos profissionais que atuam no desenvolvimento de *software* realizando atividades que contribuem diretamente com a sua produção. Esta denominação comum indica que estes usuários interagem com o ambiente para obter orientação sobre o que fazer e prover feedback sobre o que foi feito. Os desenvolvedores não precisam ser profundos conhecedores do modelo de processo que está sendo seguido na organização, ou seja, não necessitam entender sobre modelagem de processo, gerência de processo ou como a arquitetura do ADS funciona. São usuários que terão suas tarefas monitoradas de alguma forma e que devem utilizar uma interface que facilite essa situação (SOUSA et al., 2001).

Para o desenvolvedor do ambiente, o acesso é limitado a sua agenda no projeto do qual é participante, sua tarefa, o cronograma do projeto, o workflow das atividades e tarefas, os artefatos já gerados e necessários para o cumprimento de suas atividades e as informações a respeito aos demais usuários do projeto para o trabalho cooperativo.

Gerente de projeto

O termo gerente é utilizado aqui para definir diversos papéis dentro do processo de desenvolvimento de *software* incluindo as denominações gerente de qualidade, de projetos, gerente de processos, gerente de configuração de sistemas, entre outras. Para os nossos propósitos, gerentes têm como responsabilidade a monitoração, coordenação e manutenção do modelo de processo sendo executado (SOUSA et al., 2001).

Para o gerente de projeto, o ambiente deverá prover os processos já instanciados, recursos disponíveis e demais dados necessários para o desenvolvimento de um projeto no ambiente. Além disto, todo gerente é um usuário e as funcionalidades previstas para este deverão estar disponíveis aos gerentes.

O gerente poderá alocar recursos, usuários, distribuir tarefas e aprovar artefatos.

Gerente Local

O gerente local é um superusuário definido quando da criação de um ambiente cooperativo, e este usuário tem acesso irrestrito à todas as informações do ambiente (REIS, 1998).

Ao gerente local, caberá a tarefa de instanciar processos, cadastrar os recursos e instanciar projetos e gerentes para que os mesmos utilizem estas instâncias no decorrer do desenvolvimento de seu projeto.

Partindo do princípio de que cada instância do ambiente pode possuir uma configuração distinta, é função do gerente local definir as políticas de sua instância do ambiente. Esta diferença de configuração de uma instância para outra pode ocorrer pois cada gerenciador pode possuir uma política própria.

Cabe ao gerente Local disponibilizar a Infra-estrutura e os suportes necessários para o funcionamento de seu ambiente.

Tipos de acesso

Além de possuir estes três tipos de usuários pré-definidos no ambiente, há ainda o acesso entre máquinas para a replicação/distribuição de dados. Por esta razão, podemos dividir o acesso entre:

- **Acesso ao ambiente para usuários:** validará o nível de acesso e suas permissões, dependendo do usuário e projeto.
- **Acesso ao ambiente entre servidores:** para permitir a distribuição dos serviços do ambiente é necessário que os servidores possam trocar informações. A validação dos servidores da rede é necessária para evitar que usuários tentem se identificar como servidores e terem acesso a informações privilegiadas do ambiente. Esta validação depende, diretamente, do Gerenciador de Local. Um serviço é associado a um usuário em um IP. A autorização para executar um serviço é dada pelo Gerente de Locais. Os dados sobre quais estações podem ser servidores do ambiente serão armazenados junto ao serviço de persistência.

A interface do suporte a acesso do ambiente é apresentada na Figura 80. Os métodos para acesso são os tradicionais login e logout. Além destes, há um método getChave que permite armazenar a chave de validação deste usuário e, permite, que o ambiente grave esta senha localmente, evitando assim que o usuário necessite logar toda vez que utilizar o ambiente.

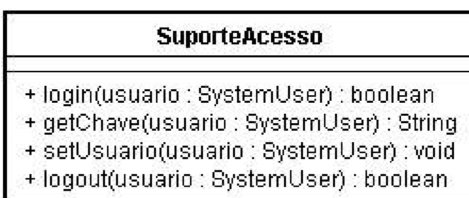

Figura 80 – Interface do Suporte a Acesso

Os métodos desta interface recebem como parâmetro o usuário local do sistema, o SystemUser. O SystemUser é uma abstração simples de usuário e possui apenas login e senha como atributos. Esta classe não é uma entidade do sistema e por isto não é persistida. Através do *login* e senha, é possível que o serviço de acesso mapeie este usuário diretamente para um Usuário do Gerenciador de Usuários. O Usuário do Gerenciador de Usuários é uma Entidade e possui papel, projeto, perfil, atividades e demais relacionamentos. A Classe SystemUser é apresentada na Figura 81. A classe SystemUser não possui método get para senha. Ela possui um getChave que é chave de acesso já criptografada. Para o serviço de acesso validar o Usuário é criptografada a chave do Usuário e ambas as chaves são comparadas.

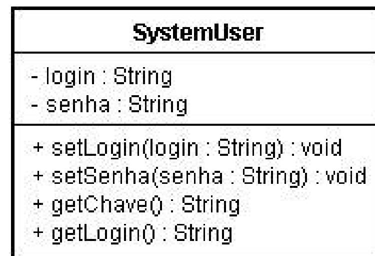

Figura 81 – Classe que representa o usuário do sistema: SystemUser

3.5.8 Serviço de Acesso

A visão geral do Serviço de Acesso é a de uma porta para a entrada no ambiente. Ele deverá ser consultado por todos os demais serviços para verificar se o usuário que requisitou um determinado serviço pode ou não ter o acesso a este. Por esta razão, um usuário que quiser ser registrado no ambiente e acessar seus serviços deverá, primeiramente, ser validado junto ao serviço de acesso.

A validação de um usuário junto ao Serviço de Acesso servirá para que o usuário do ambiente não precise efetuar login ao acessar cada novo serviço. Os usuários se registram neste serviço e, no caso de acessar outro serviço em outro servidor basta se identificar com sua chave. Este outro servidor pode verificar com o serviço de acesso se a chave apresentada pelo usuário é ou não válida.

O Serviço de Acesso deve possuir as seguintes características (REIS, 1998):

- **Alteração dinâmica de direitos:** durante a execução do ambiente deve ser permitida a alteração dos direitos de acesso dos usuários em relação aos objetos;
- **Alteração de acesso dependente do estado de uma atividade:** dependendo do estado de um objeto (grau de maturidade no processo de desenvolvimento de *software*) as permissões de acesso são automaticamente alteradas;
- **Permissões de acesso para grupos e papéis:** devem estar disponíveis facilidades para permitir alterações em permissões de acesso para grupos e papéis de usuários;
- **Prover mecanismos para controle de acesso de granularidade fina:** cada um dos componentes ou atributos do Gerenciador de Objetos deve possuir permissões de acesso específicas;

- **Identificação explícita dos gerentes e moderadores:** devem existir gerentes para cada um dos grupos e moderadores para cada um dos objetos que possam editar as permissões de acesso para os membros do grupo;
- **Controle do acesso dos usuários aos objetos:** identificar quem manipulou o objeto e quem concedeu/modificou uma permissão de acesso.

Vale lembrar que, para apoiar o compartilhamento cooperativo de informações, é necessário que estejam definidos no ambiente os papéis (funções) desempenhados pelos usuários e seus grupos. Os papéis dos usuários determinarão as permissões de acesso para cada objeto em relação aos usuários do ambiente (REIS, 1998). Desta maneira, mais que definir se um usuário pode ou não acessar uma informação, o Serviço de Acesso tem de definir se um usuário, exercendo determinado papel, pode acessar determinada informação.

A infra-estrutura necessária para a validação de um usuário no sistema é apenas um algoritmo de criptografia e um gerador de chaves de validação. A interface do Serviço de Acesso é apresentada na Figura 82.

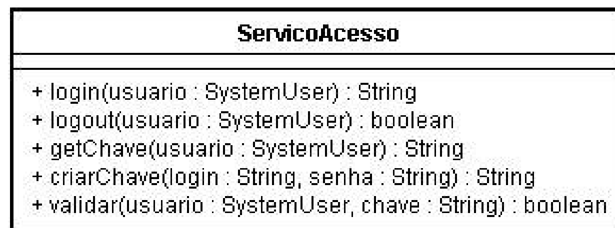

Figura 82 – Interface do serviço de acesso

Esta classe possui dois métodos similares: **login** e **validar**. A diferença entre estes métodos é que o validar retorna verdadeiro ou falso quanto à validação do usuário. O login retorna a chave caso o *login* for efetuado com sucesso. Estes métodos são utilizados de maneira diferente dentro do ambiente, pois enquanto login deve ser usado pelo usuário para obter sua chave de acesso, validar é usado por serviços para garantir que esta chave pertence ao usuário identificado.

Além de validar o acesso a usuários, o serviço de acesso possui a funcionalidade de criar chaves de acesso. Estas chaves podem ser criadas por outros serviços para a validação dos mesmos. Ao criar uma chave o serviço não valida usuário e senha.

Localmente, o serviço de acesso será uma lista de usuários que utilizam a estação de trabalho e suas chaves. Este arquivo local, mantido no *workspace*, deverá possuir uma estrutura similar ao *passwd* do Linux e manter estes dados criptografados localmente, evitando assim a possibilidade de cópia de senhas. A gravação local destas chaves permite duas abordagens de acesso ao ambiente:

- **Abertura:** Guardando a senha localmente, o ambiente pode ler o arquivo de senha local e conectar-se sem a interação do usuário. A chave pode ser lida do arquivo e informada ao ambiente. Esta abordagem necessita que o acesso físico à máquina seja restrito;
- **Restrição:** guardando localmente a lista de usuários e senhas o serviço de acesso local pode permitir que o ambiente não seja iniciado caso não lhe seja passado um usuário/senha válido, independente do mesmo estar conectado à rede ou não.

3.5.9 Suporte à Comunicação

A comunicação é, freqüentemente, o aspecto de maior dificuldade para equipes de desenvolvimento. Esta é a razão de sua importância para a engenharia de *software*. As técnicas de CSCW e engenharia devem ser combinadas para resolver conflitos que aparecem nas atividades de cooperação, de maneira a alcançar o desenvolvimento cooperativo (LIMA REIS, REIS, NUNES, 1998-b).

A camada de aplicação inclui as aplicações desktop que podem utilizar os serviços da camada de comunicação para gerenciar seus artefatos. Os serviços da camada de infra-estrutura podem definir e gerenciar os atores, os artefatos e as políticas de cooperação. Exemplos de serviços básicos para a cooperação são: serviços de sincronização, controle a acesso e versionamento. Para controlar a cooperação entre usuários, o serviço de cooperação controla a interação de acordo com políticas específicas (TATA, GODART, WIIL, 2002).

A idéia principal do suporte à comunicação é garantir a cooperação e a colaboração no ambiente através da comunicação. Uma das funções do Suporte à Comunicação é informar os usuários que estão utilizando o ambiente naquele momento, como se fosse um serviço de páginas brancas em uma lista telefônica. Além disto, o Suporte à Comunicação é utilizado pelas ferramentas para a interação e a comunicação entre usuários. Desta maneira, este suporte traz métodos para a comunicação síncrona ou assíncrona entre os usuários e suas ferramentas em um projeto.

As ferramentas que pretendem utilizar CSCW deverão utilizar os mecanismos fornecidos pelo suporte a comunicação para garantir a interação entre os usuários do ambiente.

O Suporte à Comunicação permite, ainda, o registro de Agentes de *software* para o monitoramento de atividades cooperativas.

A interface do Suporte à Comunicação é apresentada na Figura 83. Ela possui um método para listar os usuários online. Além disto, esta interface possui métodos para registrar uma ferramenta. Uma ferramenta registrada no suporte à comunicação pode receber mensagens de outras estações ou ser instanciada por outras estações dependendo da sua classe. Estes dois tipos de ferramentas cooperativas foram apresentados no Gerenciador de ferramentas e suas interfaces estão ilustradas na Figura 54 deste trabalho. Para isto, é armazenada uma lista das ferramentas locais registradas. No caso de receber uma mensagem, o suporte procura, entre as ferramentas registradas, qual(is) dela(s) sabe tratar este tipo de mensagem e repassa a mensagem à ferramenta responsável.

```
SuporteComunicacao
------------------
+ getUsuariosOnline() : List<Usuario>
+ registrarFerramenta(ferramenta : Ferramenta) : boolean
+ instanciarFerramentaRemota(usuario : Usuario, ferramenta : Ferramenta) : Ferramenta
+ enviarMensagem(mensagem : Mensagem, usuario : Usuario) : boolean
+ receberMensagem() : Mensagem
+ atualizar(usuarios : List<Usuario>) : void
+ registrarAgente(agente : Agente) : void
+ listarAgentes() : List<Agente>
```

Figura 83 – Interface do Suporte à Comunicação

Este suporte possui um método atualizar para que o serviço possa atualizar a lista de usuários online do suporte quando um usuário novo é conectado ao ambiente. Isto é possível graças à utilização do Padrão de projeto Observadores Remotos.

O Suporte à Comunicação pode ainda enviar mensagens ou instanciar ferramentas em outras estações. Para enviar mensagem às ferramentas são informados a mensagem e o usuário destinatário da mesma. O mesmo para a instanciação de ferramentas. Exemplos de ferramentas que estarão operando sobre o suporte a comunicação são ferramentas de bate-papo, com comunicação assíncrona por troca de mensagens, e editores UML cooperativos, com comunicação síncrona baseada em instanciação de objetos remotos.

3.5.10 Serviço de Comunicação

Segundo BANDINELLI, FUGGETTA e GHEZZI (1993), a arquitetura ideal para gerenciamento de processos deve ser heterogênea, concorrente, distribuída, suportar cooperação e interação entre vários agentes humanos.

O Serviço de Comunicação do FRADE permitirá que um usuário encontre outro usuário, alocado em um mesmo projeto ou local, para que os mesmos possam se comunicar diretamente utilizando, por exemplo, uma ferramenta de bate-papo interna ao ambiente. Além disto, este serviço será utilizado para resolver nomes dentro do ambiente por ferramentas cooperativas / colaborativas. Este serviço deverá fazer uma distinção dos usuários conectados por projeto ou mesmo por tarefas, para que as ferramentas cooperativas possam se utilizar deste dado para exercer uma determinada função de forma distribuída. Para esta distinção, será utilizado o modelo de negócios do Gerenciador de Usuários (seção 3.4.6) que conta com papel, habilidades e conhecimentos.

A existência de um Serviço de Comunicação se justifica pela necessidade de esconder o IP dos usuários e permitir mesmo assim a comunicação entre os usuários. Para que isto ocorra, ao se conectar na rede do ambiente, o usuário deverá se identificar para o Serviço de Comunicação do ambiente.

O Serviço de Comunicação é, ainda, responsável por enviar e receber mensagens entre usuários e por instanciar objetos remotos (comunicação síncrona) entre usuários do ambiente. Além disto, o Serviço de Comunicação funciona como uma região de Agentes para as ferramentas cooperativas / colaborativas de usuários.

Para isto, sua interface possui as características apresentadas na Figura 84. Seus métodos são bastante similares aos do suporte à comunicação. Este serviço possui um método registrarUsuario, que adiciona um usuário à lista de usuários online. Diferente do suporte a comunicação que atende a apenas um usuário, o serviço pode atender a vários usuários simultaneamente. Por esta razão, seu método registrarFerramenta conta também com a identificação do Usuário que está utilizando a ferramenta.

Figura 84 – Interface do serviço de comunicação

Quando um usuário registra uma ferramenta em seu suporte, o suporte registra esta ferramenta no serviço. O usuário que quiser instanciá-la remotamente irá acessar a ferramenta que se encontra no serviço remoto. Assim, o suporte e serviço de comunicação funcionam no ambiente como um proxy para ferramentas. Quando o usuário remoto executa um método em sua ferramenta local, o seu suporte executa o mesmo método no serviço e o serviço executa o mesmo método no suporte local e o suporte local executa o mesmo método na ferramenta local. Isto garante a sincronização entre dois usuários sem a necessidade que os mesmos estejam conectados entre si. Esta comunicação é apresentada na Figura 85.

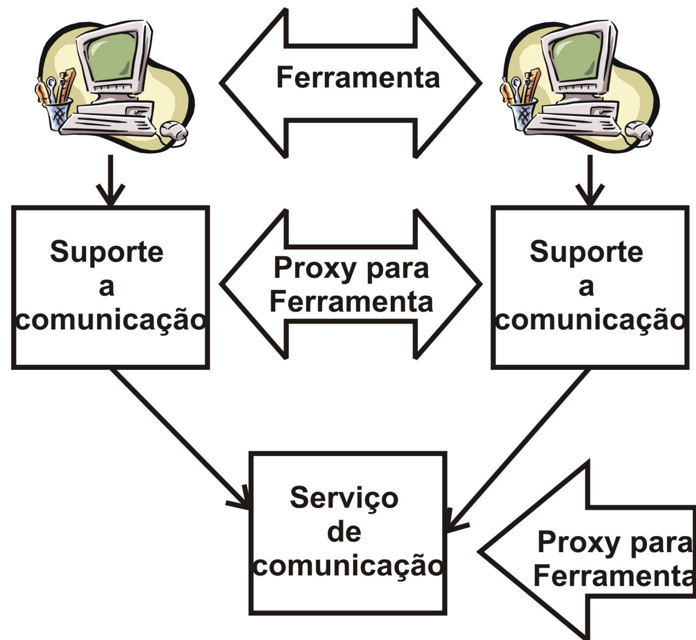

Figura 85 – Funcionamento da comunicação com objetos remotos

3.5.11 Suporte à Distribuição

O Suporte à Distribuição é o que garante ao ambiente transparência, tolerância a falhas e escalabilidade, transformando-o de um sistema cliente-servidor em um sistema distribuído. Ao contrário de um ambiente cliente-servidor clássico aonde há uma dependência da disponibilidade do serviço, a distribuição deve garantir autonomia operacional para todos os locais que participam ativamente do ambiente (LAVVA, HOLDER, BEN-SHAUL, 1997).

A autonomia operacional significa que cada participante é gerenciado por uma entidade administrativa local independente, que concorda em cooperar com outras entidades em um determinado período de tempo. Isto significa que, o recurso e o serviço do colaborador pode mudar durante o tempo. Além disto, a autonomia para colaborar implica em questões de privacidade e segurança.

Um segundo requisito é a dispersão física. Isto implica que, locais que cooperam podem ter diferentes soluções de conectividade, com diferentes velocidades de conexão ou infra-estrutura física de rede.

Outro requisito básico é a escalabilidade quanto ao número de objetos que participam no processo de computação cooperativo distribuído (LAVVA, HOLDER, BEN-SHAUL, 1997). Um ambiente deve prover mecanismos que possibilitem a diminuição ou expansão da quantidade de estações envolvidas em um processo de *software*.

O Suporte à Distribuição irá operar de maneira que a quebra de um serviço na rede não influencie nos demais serviços e que um serviço possa ser encontrado em mais de uma máquina (BEN-SHAUL, KAISER, 1994) (POHL et al., 1999).

Em contraste à integração dos serviços que levam em consideração as interfaces dos serviços, um mecanismo de controle de integração é necessário para transmitir requisições de um serviço em particular e retornar a informação entre os componentes de integração de processo do ambiente.

O suporte à distribuição tem função semelhante ao de comunicação. Enquanto o suporte à comunicação se encarrega dos usuários conectados e da comunicação entre os mesmos, o suporte à distribuição se encarrega dos serviços habilitados no ambiente e da comunicação entre eles.

O suporte à distribuição também possui a possibilidade de registrar Agentes junto ao ambiente. Diferentes dos agentes de comunicação, os agentes aqui registrados atuarão não sobre o controle e monitoramento de ferramentas, mas sobre os serviços do ambiente.

Este suporte possui uma lista dos serviços autorizados que se encontram disponíveis no ambiente e informará à camada de comunicação aonde a mesma deve se conectar para exercer determinada atividade. Mais que manter a lista de locais, este suporte é responsável, diretamente, pela comunicação entre os usuários e os servidores fornecendo métodos para a comunicação síncrona ou assíncrona.

Para tal, o suporte a comunicação é implementado seguindo a interface proposta na Figura 86. Esta classe possui métodos para a comunicação com os serviços do ambiente e também é responsável por saber aonde cada serviço do ambiente se encontra.

```
SuporteDistribuicao
+ getLocais() : List<Servico>
+ atualizar(servicos : List<Servico>) : boolean
+ enviarMensagem(local : Local, mensagem : Mensagem) : boolean
+ receberMensagem() : Mensagem
+ registrarServico(servico : Servico) : boolean
+ instanciarServicoRemoto(servico : Servico) : Servico
+ getTipoServico(servico : Servico) : int
+ getServicoInformacoes(servico : Servico) : int
+ salvarConfiguracoes(servicos : List<Servico>) : void
+ carregarConfiguracoes() : List<Servico>
+ registrarAgente(agente : Agente) : void
+ listarAgentes() : List<Agente>
```

Figura 86 – Interface do Suporte a Distribuição

A comunicação entre usuários/ferramentas e servidores pode ser feita através da instanciação de serviços remotos ou através da troca de mensagens.

Quando a camada de comunicação vai conectar-se ao ambiente, é requisitado ao suporte à distribuição quais os locais possuem os serviços do ambiente. Por esta razão, o suporte à comunicação salva localmente a lista dos locais com autorização para executar serviços no ambiente. Uma vez conectado, a comunicação deve requisitar ao serviço a lista de locais para a distribuição e atualizar sua lista local.

Além disto, o suporte à distribuição é o responsável por registrar os serviços locais que compõe o *workspace* de uma estação. Assim, tanto serviço local quanto remoto podem ser encontrados nesta lista dinâmica. Caso uma mensagem chegue ao suporte, o mesmo irá enviá-la ao serviço responsável por tratar esta mensagem. Os serviços locais que não são reconhecidos como serviços do ambiente não deverão receber mensagens pois os mesmos não serão reconhecidos por outros locais como um serviço para a distribuição.

3.5.12 Serviço de Distribuição

Segundo HOLDER, BEN-SHAUL e GAZIT (1999), a abordagem natural para alcançar a distribuição é fornecer uma camada dinâmica que permita localizar componentes a qualquer momento. O aspecto de distribuição deve ser explícito se necessário, assim como deve ser explícita a especificação de possibilidades de redundância ou políticas de realocação de serviços.

Seguindo este conceito, e, por se tratar de um ambiente distribuído, haverá a possibilidade de dispôr os vários serviços fornecidos ao ambiente em várias máquinas geograficamente distribuídas. Por esta razão, há a necessidade de identificar quais locais possuem os serviços do ambiente. Esta localização de servidores é feita através do serviço de distribuição. Ao ser validado no ambiente, um usuário pode requisitar a lista de locais responsáveis por manter cada um dos serviços. Graças a esta funcionalidade a configuração dos serviços do ambiente poderá ser dinâmica.

Isto também permitirá que cada serviço seja alocado em uma máquina e que estas máquinas forneçam apenas um serviço. Independente de possuir ou não todos os serviços é

obrigação de um servidor possuir a lista de serviços de todo o ambiente. Desta forma, os clientes poderão confiar a qualquer servidor sua tarefa e o mesmo irá se encarregar de resolvê-la, independente de esta solução ser local ou não. Isto é feito de uma maneira similar ao da camada de comunicação. Uma estação registra um serviço em seu suporte e o mesmo se torna um *proxy* para o ambiente. Outro serviço pode instanciar este *proxy* e atender requisições para o serviço e encaminhá-la à estação responsável de maneira transparente ao usuário requisitante.

Por esta razão, o único serviço que todo servidor do ambiente deve oferecer, obrigatoriamente, é o serviço de distribuição. A interface deste serviço é apresentada na Figura 87.

```
ServicoDistribuicao
---
+ getLocais() : List<Servico>
+ atualizar(servicos : List<Servico>) : boolean
+ enviarMensagem(local : Local, mensagem : Mensagem) : boolean
+ receberMensagem() : Mensagem
+ registrarServico(servico : Servico) : boolean
+ instanciarServicoRemoto(servico : Servico) : Servico
+ getTipoServico(servico : Servico) : int
+ getServicoInformacoes(servico : Servico) : int
+ salvarConfiguracoes(servicos : List<Servico>) : void
+ carregarConfiguracoes() : List<Servico>
+ registrarAgente(agente : Agente) : void
+ listarAgentes() : List<Agente>
```

Figura 87 – Interface do serviço de distribuição

3.6 Camada de comunicação

A camada de comunicação é a mais baixa do *framework*, como ilustra a Figura 88. Esta camada possui a função da comunicação de cada estação do ADS com o meio físico e, por meio desta, com outras estações.

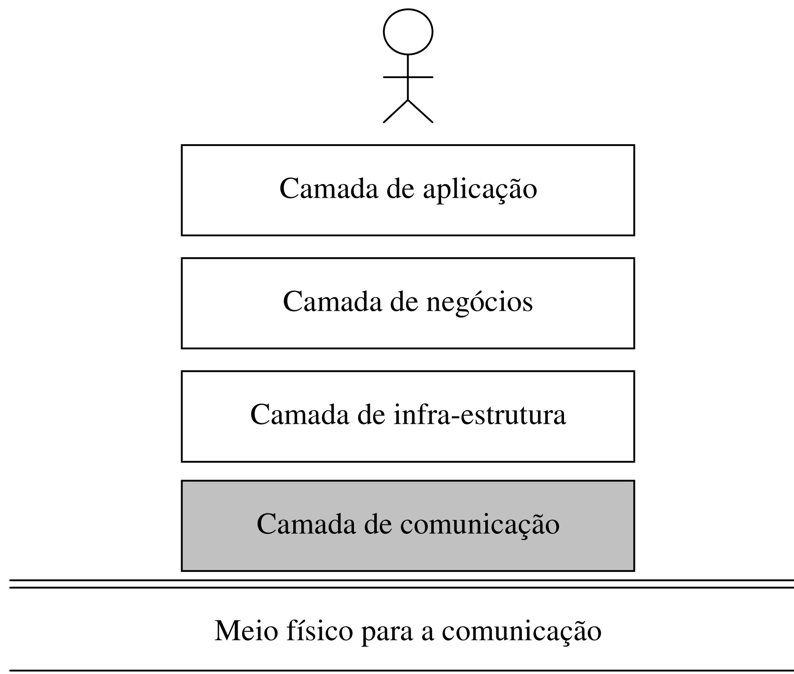

Figura 88 – Camada de comunicação

PAASIVAARA (2003) identificou quatro tipos principais de necessidades para o estabelecimanento da comunicação:

1. A solução de problemas;
2. Informação e monitoramento;
3. Criação de relacionamentos;
4. Tomada de decisão e cordenação.

O monitoramento é necessário para dar transparência ao progresso do projeto. A criação de relacionamentos incluem todo tipo de comunicação social. A solução de problemas é uma comunicação presente de maneira mais constante em projetos em que há mais incertezas e tomadas de decisões.

Segundo BARGOUTHI et al. (1996), tipicamente os usuários de um ADS estão em diferentes estações de trabalho conectados por uma LAN ou WAN. Normalmente os usuários compartilham dados via um servidor central que controla a concorrência e a transação sobre estes dados. Isto porque, segundo CUGOLA e GHEZZI (1999) e AMBRIOLA, CONRADI e FUGGETTA (1997), a maioria dos ADSs, desenvolvido na década de 90, adotam o padrão arquitetural cliente-servidor. A definição de processo é centralizada e as ferramentas se comunicam através de conexões ponto-a-ponto. Exemplos destes ADSs são Adele, EPOS, JIL, Marvel, Merlin, Oikos, ProcessWeaver, Provence, SENTINEL e SPADE.

Abordagens comuns de ADSs utilizam, por exemplo, CORBA, RMI e RPC. O resultado desta arquitetura é um alto acoplamento entre o objeto requisitado e o serviço fornecido. Estas abordagens reduzem a possibilidade de reconfigurar a arquitetura da aplicação em tempo de execução e resultam em uma escalabilidade limitada (CUGOLA, GHEZZI, 1999).

Um estilo que recebeu uma atenção maior é baseado na noção de eventos. Os componentes de uma arquitetura baseada em eventos cooperam enviando e recebendo eventos, uma forma particular de mensagens. O remetente envia um evento para o *event dispatcher* que é o responsável por distribuir o evento para todos os componentes que possuem interesse declarado em recebê-lo. Após isto, o *event dispatcher* permite que remetente e destinatários de um evento sejam totalmente desacoplados (CUGOLA, GHEZZI, 1999).

Este estilo utiliza, entre outras técnicas, o *framework* JEDI. Este *framework* é caracterizado por suas propriedades serem: assíncronas, baseadas em *multi-cast*, a não identificação do remetente ou destinatário nas mensagens e a garantia de recebimento do evento na mesma sequencia em que eles são produzidos (CUGOLA, GHEZZI, 1999).

BEN-SHAUL e KAISER (1994) lembram que a comunicação pode ser síncrona e enviar requisições de execução ao servidor ou assíncrona e trazer o dado localmente para a sua execução. A utilização destas técnicas deve prever a sincronização e a integridade dos dados. Por esta razão, o nível de importância destes dados deve ser avaliada. Manter dados referentes a projetos e processos local, muitas vezes, pode gerar instâncias distintas e isto pode implicar na não integridade do projeto. Por esta razão, as informações de processo e projeto devem ser mantidos sempre remotas. As informações consideradas no nível dos dados podem ser replicadas localmente e dispor de mecanismos de sincronização e, por isto, utilizar comunicação assíncrona.

O conceito de comunicação síncrona e assíncrona assume diferentes definições na literatura. RAMESH e PERROS (1998) chamam comunicação síncrona a comunicação aonde há uma espera pela resposta e comunicação assíncrona aonde não há espera pela resposta em uma requisição. Exemplos destas comunicações seriam respectivamente a utilização de objetos distribuídos ou comunicação por troca de mensagens.

Segundo AMBRIOLA, CONRADI e FUGGETTA (1997), a maioria dos ADSs possui suporte apenas cooperação assíncrona. Este trabalho propõe uma camada de comunicação que permita a comunicação síncrona e assíncrona. Isto seria alcançado por meio de um *middleware* que pemite troca de mensagens e/ou a instanciação de objetos remotos.

A camada do *middleware* tem o objetivo de mascarar a heterogeneidade de arquiteturas de computadores, sistemas operacionais, linguagens de programação e tecnologias de rede para facilitar o desenvolvimento de aplicações e gerenciamento (BIANCHI et al., 2003). Isto é conseguido através de serviços que intermediam a integração entre aplicações (RUH, MAGINNIS, BROWN, 2000).

De maneira mais simples, a adição de um *middleware* ao ADS lhe dá capacidade de executar código através da rede e capacidade de leitura e escrita de dados através da Internet (CHAMBERS, LANG, 1999).

Além disto, o *middleware* do FRADE permite quatro tipos de comunicação (BEN-SHAUL, KAISER, 1994):

- Comunicação cliente-servidor;
- Comunicação entre servidores;
- Comunicação servidor-cliente;
- Comunicação entre clientes.

A possibilidade da comunicação servidor-cliente distingue o *middleware* do FRADE dos demais protocolos que permitem apenas comunicação cliente-servidor. Protocolos como o

RMI, por exemplo, não permite que o servidor envie uma informação ao cliente caso o mesmo não a requisite.

O modelo da camada de comunicação do FRADE é apresentado na Figura 89. A última camada da comunicação é a camada de transporte. Esta camada baseia-se em um *socket* para a comunicação. O *socket* foi escolhido para a comunicação física por permitir comunicação *full-duplex*. Desta maneira, os servidores podem ativar a comunicação com os clientes, lhes enviar mensagens e instanciar objetos remotos nos clientes.

A utilização de *sockets* é recomendada por BELOGLAVEC et al. (2005) devido aos fatores de desempenho na rede. Segundo estes autores, a velocidade de recebimento de decodificação de mensagens não depende apenas da largura de banda da rede. A infra-estrutura do servidor utilizado atua de maneira diferente com os vários tamanhos de mensagens. Servidores baseados em *threads* podem ler diretamente de *sockets* com o auxílio de *streams* de dados, enquanto servidores orientados a eventos, lêem mensagens longas que lhes são transmitidas e isto implica em uma diminuição significativa de desempenho para o servidor. Por esta razão, o tamanho das mensagens a serem trocadas é um ponto importante no planejamento da aplicação.

A camada de aplicação do FRADE possui uma divisão em camadas. Esta divisão foi feita para facilitar uma reimplementação da camada de comunicação com outra tecnologia de transporte, distinta de *socket*. Similar às camadas do modelo ISO/OSI ou TCP/IP, as camadas da camada de comunicação são apresentadas na Figura 89.

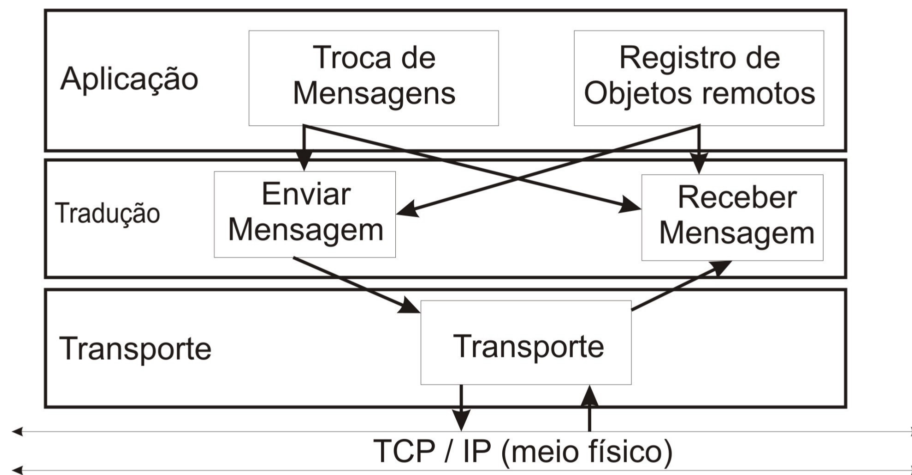

Figura 89 – Estrutura da camada de comunicação

Indo ao encontro da preocupação quanto ao custo da comunicação, as mensagens trocadas pelo *middleware* são objetos serializáveis compactados para garantir um melhor aproveitamento da infra-estrutura física da rede. A camada de tradução possui a função de serializar as mensagens e compatacá-las para o envio e descompacta-las e desserializá-las no recebimento. A serialização/compactação auxilia não apenas no fator desempenho, mas, também, no fator segurança. A compactação das mensagens transforma-as em código ilegível a uma tentativa de interceptação da comunicação do ambiente.

A camada superior é a de aplicação que trocará informações com os suportes do ambiente. É função desta camada consultar o Suporte a Comunicação e Distribuição para encontrar o destinatário das mensagens do ambiente. Também, é função desta camada repassar as mensagens recebidas dos suportes à comunicação e distribuição para o destinatário.

A interface da camada de comunicação possui acesso a seus objetos e é apresentada na Figura 90. O método conectar desta interface não possui parâmetros. Isto porque o usuário não escolhe aonde ele será conectado. Esta informação será requisitada ao suporte a distribuição.

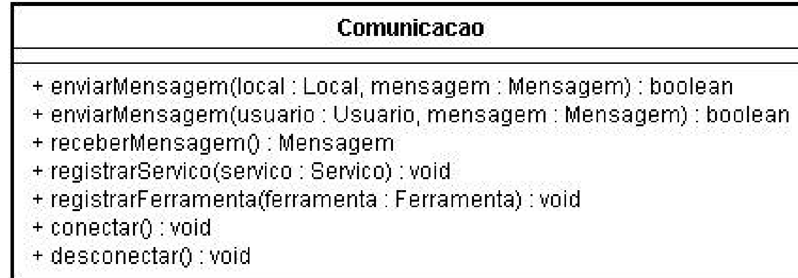

Figura 90 – Interface da camada de comunicação

A interface da camada de comunicação possui os métodos para que os serviços e ferramentas possam comunicar-se com o ambiente. Para a troca de Mensagem é utilizada uma interface padrão, que deve ser implementada por toda mensagem do ambiente. A comunicação baseada em troca de mensagens permite que cada serviço ou ferramenta possua uma especialização de mensagem de maneira que a mesma tenha os atributos necessários para a comunicação. A interface da mensagem é apresentada na Figura 91.

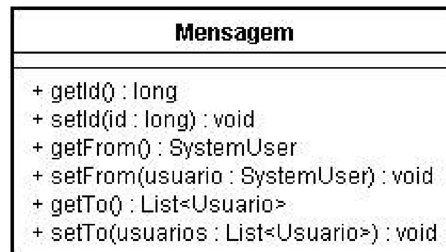

Figura 91 – Classe mensagem

O registro de objetos remotos é um tipo de comunicação definida pelo ambiente. Apesar de funcionar com troca de mensagens, esta comunicação é bloqueante. Para a comunicação entre objetos remotos são utilizadas quatro tipos de mensagens padrões, como ilustra a Figura 92.

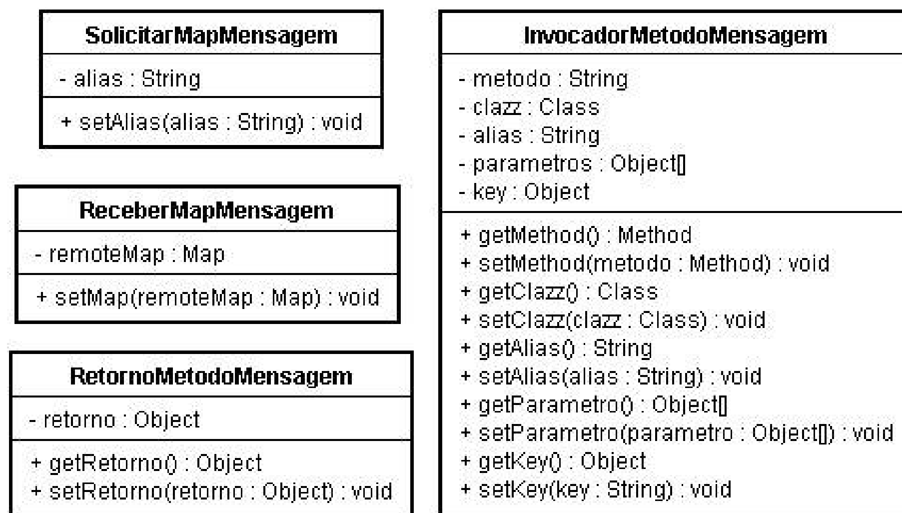

Figura 92 – Classes mensagens padrão para a comunicação síncrona

Estas classes permitem que o ambiente envie e receba mensagens para os objetos remotos registrados. Através destas quatro mensagens padrão, a comunicação consegue instanciar objetos registrados no suporte de comunicação ou distribuição e fornecer um proxy deste objeto localmente. Para acessar um objeto remoto é necessário ainda conhecer a sua interface.

Na camada de tradução é usada uma classe para a compressão e descompressão das mensagens, e seus métodos são apresentados na Figura 93.

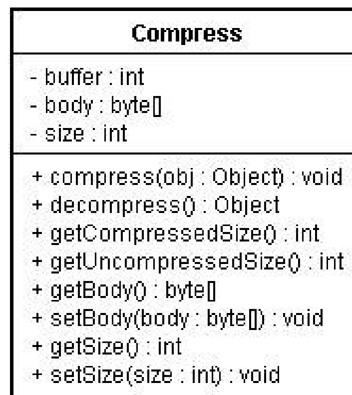

Figura 93 – Classe compressão usada para compactar as mensagens da comunicação

A camada de transporte da comunicação é a responsável pela conexão/desconexão com o ambiente e também por enviar as mensagens pelo socket e ouvir o socket aguardando respostas. Esta classe é apresentada na Figura 94.

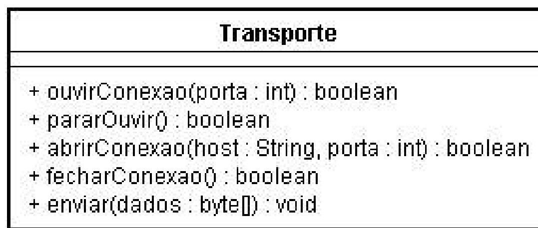

Figura 94 – Diagrama de Classe do Componente de Transporte

3.7 Conclusão

Neste capítulo foram apresentadas as camadas do *framework*, suas interfaces, seus gerenciadores, serviços e suportes.

Na camada de aplicação foram apresentadas as possibilidades de aplicações principais, em modo texto ou gráfico, tratamento de exceção através de log ou mensagens para o usuário, a configuração da aplicação e, outros componentes para o apoio ao desenvolvimento de ferramentas gráficas, conforme ilustra a Figura 95.

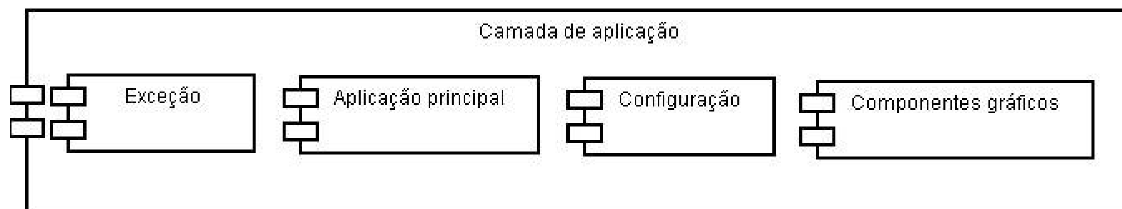

Figura 95 – Componentes da camada de aplicação

Na camada de negócios foram apresentados os gerenciadores do ambiente e o conteúdo de suas classes. Os gerenciadores desta camada compõem o Gerenciador de Objetos e estão apresentados na Figura 96.

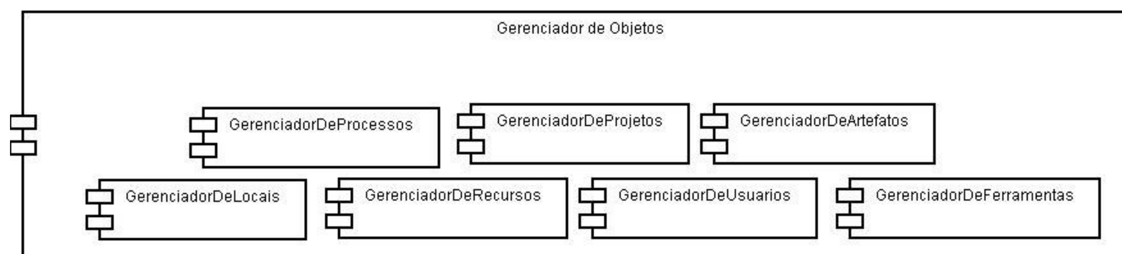

Figura 96 – Componentes da camada de negócios

A Figura 97 apresenta a interdependência dos gerenciadores da camada de negócios a partir de um diagrama de componentes. A componentização dos gerenciadores permite que as classes de negócios dos gerenciadores possam ser alteradas sem que a mudança altere todas as classes do *framework*.

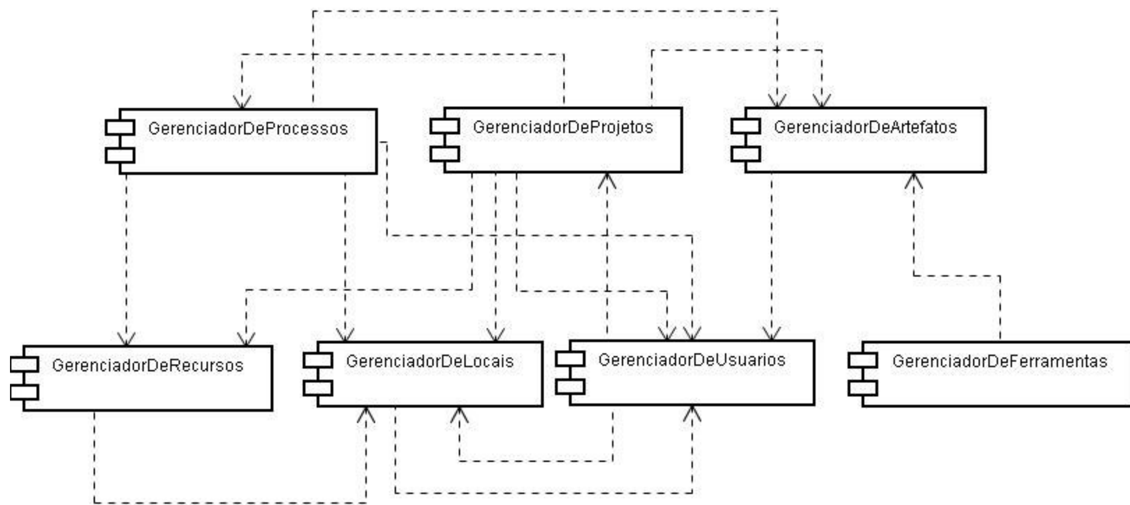

Figura 97 – Dependência dos gerenciadores da camada de negócio

Na camada de infra-estrutura foram apresentados os suportes e serviços do ambiente, conforme ilustrado na

Figura 98.

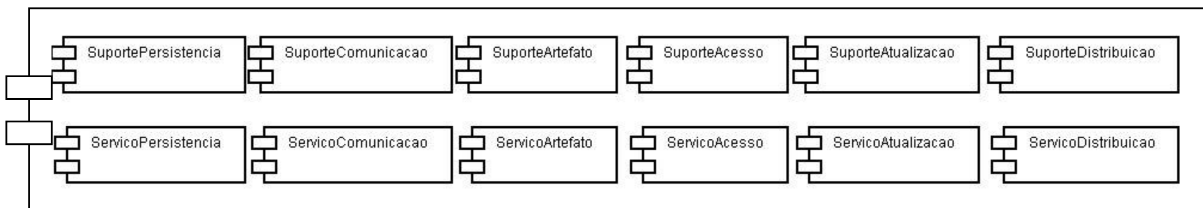

Figura 98 – Componentes da camada de infra-estrutura

Os suportes e serviços foram mapeados a partir das necessidades apresentadas pela camada de negócios e a dependência de cada gerenciador com os suportes, como ilustra a Figura 99.

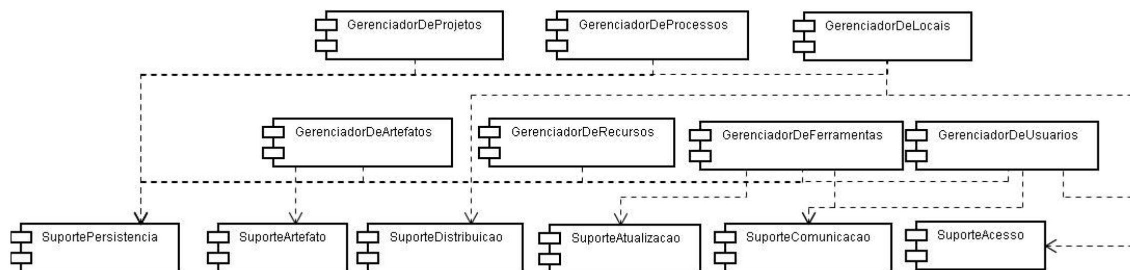

Figura 99 – Dependência dos gerenciadores e os suportes

As dependências foram descritas em função dos suportes e não dos serviços, pois os suportes deverão estar presentes em todas as instâncias do ambiente enquanto os serviços podem estar distribuídos.

Na camada de infra-estrutura há uma dependência implícita entre os suportes. Para que as estações do ambiente consigam encontrar os serviços, é necessário que o suporte à distribuição esteja em execução. Além disto, para acessar o ambiente é necessário validar o usuário junto ao serviço de acesso. Por esta razão, todos os demais suportes dependem do suporte ao acesso e à distribuição. A Figura 100 ilustra esta dependência.

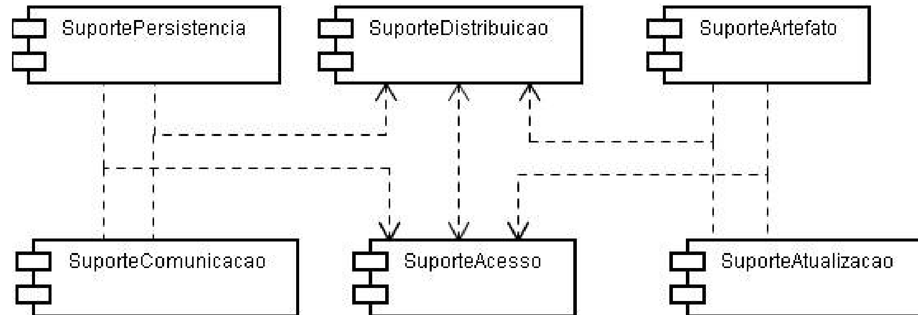

Figura 100 – Dependência entre os suportes

Os serviços do ambiente podem ser locais ou remotos. Os serviços locais são utilizados para garantir as funcionalidades do ambiente no caso de o usuário estar desconectado. Os serviços locais compõe o *workspace* do ambiente. Como foi dito definição do serviço/suporte a artefatos (seção 3.5), o conceito de *workspace* encontrado na literatura é associado a uma cópia local do repositório de artefatos. Neste trabalho, o *workspace* é composto não apenas pelo repositório local de artefatos (chamado de serviço local) mas também por cópias locais dos demais serviços que compõe o ambiente.

Localmente, o serviço de distribuição mantém uma lista de locais para a conexão do usuário. O serviço de atualização local é uma cópia dos módulos requisitados pelo usuário para futura instalação no ambiente. O serviço de persistência local pode ser visto como a persistência de dados que estão sendo utilizados pelo usuário em um determinado momento, como, por exemplo, sua agenda. O serviço local de acesso pode ser implementado como um arquivo que possui usuário e senha criptografados e que obriga o usuário seja validado no sistema mesmo quando este está desconectado. O serviço local de artefatos, como foi dito anteriormente, é uma cópia local do repositório global que pode conter apenas os artefatos que estão sendo utilizados em um determinado momento. O *workspace* do FRADE é representado pela Figura 101.

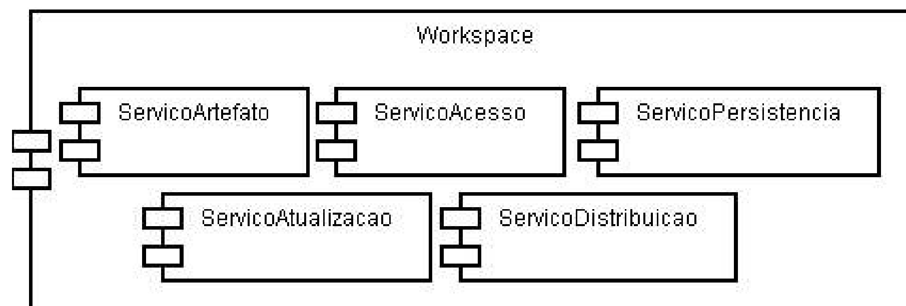

Figura 101 – Componentes do *Workspace*

Entre os serviços encontrados no FRADE, o único que não possui cópia local é o de comunicação. O serviço de comunicação mantém a lista de usuários online, Agentes de monitoramento para ferramentas e possibilita a cooperação e colaboração dentro do ambiente através de ferramentas que possuem este fim. Por esta razão, este serviço não possui

mapeamento dentro do *workspace* pois não há funcionalidade para o mesmo quando desconectado.

Na camada de comunicação foram apresentados os tipos de comunicação do ambiente e as classes responsáveis pela comunicação. O FRADE possui comunicação síncrona e assíncrona. Apesar de a camada de comunicação também possuir uma divisão lógica em camadas, ela pode ser representada, de maneira simplificada por dois componentes: comunicação síncrona e assíncrona, conforme ilustra a Figura 102.

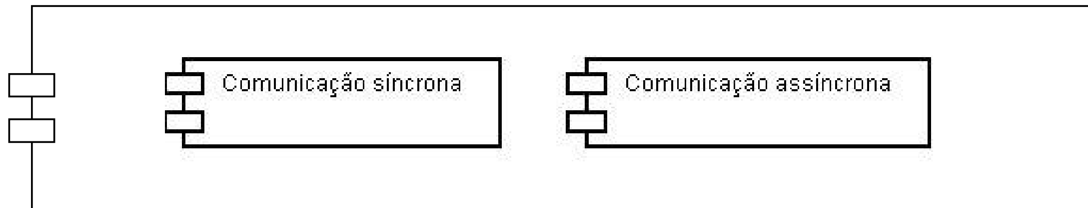

Figura 102 – Componentes da camada de comunicação

A camada de comunicação também possui dependência com o Suporte à Distribuição e ao Acesso para iniciar a comunicação com os serviços do ambiente. A camada de comunicação depende também do Suporte à Comunicação para a colaboração entre usuários. Esta dependência é ilustrada na Figura 103.

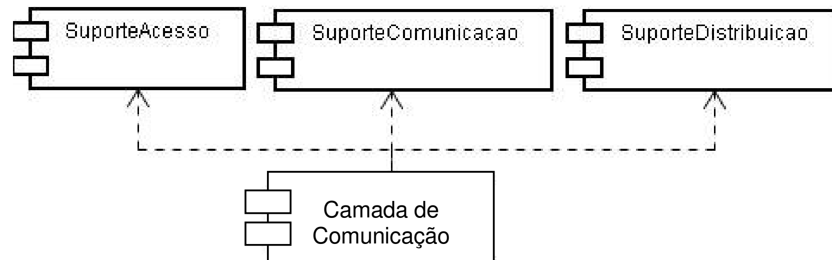

Figura 103 – Dependências da Camada de Comunicação

No próximo capítulo serão vistas a instanciação, experimentação e avaliação do *framework* proposto.

4 Avaliação do *Framework*

"A experiência com ADSs indica que uma arquitetura aberta é crucial"

(CONRADI et al., 1991)

No final do capítulo 2 foi elaborada uma tabela (Tabela 5 – Requisitos desejáveis para um ADDS) listando os requisitos e características desejáveis para um ADDS. Partindo destes requisitos, a Tabela 8 a seguir apresenta o posicionamento do FRADE em relação a estes requisitos mapeados.

Tabela 8 – Avaliação dos requisitos desejáveis para ADDS no FRADE

Infra-estrutura de comunicação	Suprida pelos **Serviço e suporte à comunicação**. A infra-estrutura de comunicação do FRADE permite troca de mensagens entre ferramentas e trabalho cooperativo entre usuários.
Definição de processo	A definição de processo no FRADE ocorre na sua camada de negócios por meio **Gerenciadores de Objetos**. Cada Projeto instancia um Processo previamente definido. No modelo de negócios do FRADE é possível ainda alterar um processo ou projeto sem que esta alteração cause impacto aos demais projetos definidos pelo processo modificado.
Integração e interoperabilidade	A integração de novas ferramenta ao ambiente é feita por meio da **infra-estrutura de atualização.** A interoperabilidade de ferramentas é feita no FRADE por meio da **infra-estrutura de Artefatos** que possibilita repositório local (*workspace*).
Autonomia local	A autonomia local de uma estação no FRADE é possível devido à existência de repositórios locais distintos no *Workspace* para os **serviços de Artefato, Persistência, Distribuição, Atualização e Acesso**.
Percepção	A **infra-estrutura de Comunicação e Distribuição** definida no FRADE oferecem apoio a percepção em um ambiente. A infra-estrutura de comunicação garante a percepção dos usuários no ambiente e a infra-estrutura de distribuição garante a percepção dos serviços presentes no ambiente. Apesar de garantir a percepção quanto à alteração dinâmica do ambiente, o FRADE garante, também, a transparência caso o usuário não queira verificar a configuração do ambiente como um todo.
Configuração dinâmica	A configuração dinâmica do ambiente é garantida em níveis distintos: configuração dinâmica de serviços por meio da **infra-estrutura de distribuição**, configuração dinâmica do ambiente por meio da **infra-estrutura de atualização** e configuração dinâmica do projeto e processo por meio da **infra-estrutura de Persistência**.

Persistência	Estão definidos no FRADE o apoio necessário a persistência de diferentes tipos de dados: meta-dados, ICS e módulos. A persistência de meta-dados é feita pela **infra-estrutura de persistência**, de ICS pela **infra-estrutura de Artefatos** e de módulos pela **infra-estrutura de atualização**.
Semântica transacional	Apesar de possuir três tipos distintos de repositórios, apenas no serviço de persistência (meta-dados) é garantida a semântica transacional. O serviço de persistência utiliza banco de dados relacionais e, permite, transações longas por meio de seções no banco de dados.
Gerência de Acesso	A gerência de acesso no FRADE é apoiada pela infra-estrutura de acesso que trata, não apenas o acesso de usuários ao ambiente, mas garante a validação dos serviços oferecidos no ambiente.
Gerência de Visão	A gerência de visão no FRADE é apoiada pela **camada de aplicação**. Esta camada possui a responsabilidade de tornar o ambiente homogêneo quanto às suas ferramentas.
Gerenciamento de Recursos	Foi definido na camada de negócios do FRADE o apoio necessário ao gerenciamento de recursos. Foram definidos os diferentes tipos de recursos que podem existir em um ambiente. Para cada tipo de recurso há um gerenciador na camada de negócios que irá permitir o gerenciamento deste recurso. São eles: **Gerenciador de recursos**, **Gerenciador de usuários**, **Gerenciador de ferramentas** e **Gerenciador de locais**, respectivamente.

4.1 Implementação

Conforme é demonstrado na comparação dos ambientes estudados na seção 2.2 deste trabalho, a plataforma de desenvolvimento do Projeto ADS está centrada na orientação a objetos, tendo Java como a linguagem de programação escolhida. Esta escolha deve-se ao fato dela suportar a Orientação a Objetos e, por ser independente de plataforma, característica desejável a um ambiente que se propõe a ser de uso geral. Segundo CHAMBERS e LANG (1999), Java suporta em sua biblioteca padrão GUI, rede, utilização de métodos nativos, orientação a objetos, componentização, *Multi-thread* e portabilidade entre diversas arquiteturas. FREITAS (2005) lembra que a portabilidade deve-se ao fato de que o sistema é executável nas plataformas para as quais está disponível o *Java Runtime Environment*.

Como modelo de definição do ambiente foram utilizados os trabalhos publicados anteriormente pelo grupo de pesquisa. A adequação neste modelo ocorreu basicamente no Gerenciador de objetos, conforme foi ilustrado no capítulo 3 desta dissertação. Os efeitos desta alteração foram apresentados neste trabalho nas definições dos serviços e suportes.

A primeira preocupação quanto à alteração do gerenciador de objetos proposto nos trabalhos anteriores foi quanto a separação clara do que é modelo de negócios e o que é funcionalidade no ambiente. Esta separação dividiu entre as camadas de infra-estrutura e camada de negócios, os requisitos funcionais dos não funcionais, ficando os requisitos

funcionais de ADDS na camada de negócios e os requisitos não funcionais na camada de infra-estrutura.

A definição explícita da camada de comunicação, como é apresentado neste trabalho, ocorreu devido à necessidade de minimizar os recursos necessários para a implantação de um ADDS baseado no FRADE. Primeiramente, os serviços do ambiente utilizam protocolos de comunicação próprios baseando-se na comunicação independente de seus componentes, como, por exemplo, um repositório CVS e um banco de dados com suporte a conexão de rede. Porém, o controle de acessos e a configuração do ambiente utilizando tal modelo de comunicação se tornaram mais complexos. Seria necessário, por exemplo, utilizar várias portas, uma para cada serviço e cada serviço precisaria implementar seu serviço de validação. Ao adicionar ao gerenciador de usuários níveis de acesso com diferentes granularidades, a validação de usuários a partir de cada serviço se tornou uma implementação que implicaria na alteração de componentes externos como bancos de dados e VCSs.

Devido ao escopo deste trabalho o *framework* aqui proposto foi, parcialmente, implementado como será apresentado a seguir.

Segundo FREGONESE, ZORER e CORTESE (1999) o uso de padrões de projeto no processo de criação de um *framework* é feito pois:

1. fornece um vocabulário comum para o projeto;
2. reduz a complexidade do sistema, uma vez que as abstrações são nomeadas e definidas;
3. fornece peças que se conectam aonde projetos mais complexos podem ser construídos;
4. fornece metas para a reestruturação da hierarquia de classes.
5. padrões de projeto são utilizados como uma parte natural da documentação de um *framework*.

Para a implementação do FRADE foram utilizados padrões de projeto. Segundo BARGHOUTI e KAISER (1999), a integração do *framework* através de padrões de projetos tem por objetivo permitir a evolução do sistema. Já a evolução do sistema deve permitir que o mesmo altere suas filosofias de gerenciamento e, também, a reorganização de seus componentes de acordo com a evolução do sistema.

Entre os padrões de projeto utilizados na implementação deste *framework* podem ser destacados (BANASZEWSKI, 2006):

- ***Look Up*** para o serviço de comunicação e distribuição, guardando a referência aonde cada usuário está localizado em determinado instante;
- ***Partial aquisition*** para o suporte a atualização, recuperando primeiro os meta-dados dos pacotes de atualização para depois recuperar os pacotes propriamente dito;
- ***Eager aquisition*** para o carregamento do *middleware* no momento de inicialização do ambiente;
- ***Evictor*** para os serviços do ambiente, desconectando os cliente inativos;
- ***Coordinator*** para o gerenciamento de distribuição;
- ***Caching*** para a trava da infra-estrutura de artefatos;
- ***Pooling*** para o serviço de persistência, permitindo uma melhor utilização das conexões com o banco de dados;

- ***DAO/BO*** para a instanciação dos objetos da camada de negócios;
- ***Abstract Factory*** para o instanciação do ambiente e a atualização de seus componentes e ferramentas;
- ***Broker*** para a camada de comunicação.

Além destes padrões aqui destacados há classes *Singletons*, *Factorys* e outros padrões mais comuns que foram implementadas internamente no desenvolvimento de cada componente em individual.

4.2 Extensibilidade

A componentização do desenvolvimento permite criar, primeiramente, uma infra-estrutura a ser utilizada no desenvolvimento de sistemas para um domínio de aplicação. Isto torna possível uma implementação diferente de cada um dos componentes de infra-estrutura do ambiente, garantindo que os mesmos interajam entre si. Por distintas instanciações do *framework* é possível desenvolver distintos Ambientes Distribuídos de Desenvolvimento de *Software* (ADDS) e aplicações para estes ADDS, sem que haja a necessidade de reimplementar a infra-estrutura do ambiente a cada nova aplicação desenvolvida.

A extensibilidade do ambiente pode ser vista como a reimplementação de cada um dos seus gerenciadores, suportes, serviços ou classes de apoio para aplicações e comunicação.

A necessidade de alterar as classes da camada de aplicação pode ocorrer para facilitar a integração do ambiente desenvolvido sobre este *framework* com ferramentas ou *frameworks* existentes no mercado. Pode-se utilizar, por exemplo, a API Log4J para a implementação do tratamento de exceções. Uma outra implementação poderá unir a geração de logs com *awareness* para usuários. Na camada de aplicação pode-se ainda estender a interface texto do ambiente de maneira a utilizar parâmetros para a sua execução, facilitando assim, a inicialização do ambiente com a inicialização da estação de trabalho.

A camada de negócios pode ser extendida para atender novas necessidades quanto à representação dos dados em um ADS ou para adicionar políticas e regras de acesso aos objetos desta camada. Esta extensão pode ser feita em apenas um gerenciador ou em vários, gerando uma nova versão do gerenciador de objetos. A validação de um usuário para a criação de um novo processo seria um exemplo da extensão sem a alteração dos objetos da camada de infra-estrutura. A adição de métricas no gerenciador de projetos seria um exemplo desta extensão alterando a estrutura deste gerenciador. Caso haja a necessidade de alterar a interface de um dos gerenciadores, é necessário verificar quais gerenciadores utilizam esta interface e propagar a alteração entre os demais gerenciadores. A relação de dependência entre os gerenciadores é apresentada na Figura 97 deste trabalho.

A camada de infra-estrutura pode ser extendida para alterar a comunicação entre suportes e serviços, para novas implementações de serviços ou para a adição de regras e políticas de acesso, tanto a suportes quanto a serviços. Uma nova implementação do serviço de artefato que opera sobre um repositório ClearCase da Rational seria um exemplo de nova implementação de serviço. A checagem de permissão e papel de usuário para a adição de nova ferramenta ao repositório de atualização seria um exemplo de extensão para adição de regras de acesso.

A camada de comunicação pode ser extendida para a reimplementação de seus componentes ou para a alteração de seu funcionamento. A adição de criptografia nas

mensagens seria um exemplo de reimplementação desta camada que não altera seu relacionamento com a camada de infra-estrutura. A mudança no protocolo de comunicação e a comunicação através do RMI seria um exemplo de extensão desta camada.

A extensão deste *framework* pode ser feita com a geração de novos componentes. Por se tratar de um ambiente distribuído, a alteração de um componente pode necessitar sua atualização em todas as estações do ambiente. A atualização do ambiente é feita através do serviço de atualização. Para garantir o funcionamento do ambiente após a extensão de um componente, é utilizado o descritor do módulo a ser instalado e as dependências que o mesmo descreve. O descritor do módulo traz a descrição explícita das dependências, por exemplo, de uma ferramenta. Ele trará descrito que uma ferramenta só pode operar caso o gerenciador de objetos esteja na versão 2.0, por exemplo.

A adição de uma ferramenta de métricas ao ambiente, por exemplo, requer a alteração do gerenciador de projetos. Esta ferramenta precisa que este gerenciador seja atualizado. Para a instalação desta ferramenta é definida sua dependência com o gerenciador de projetos na versão 2.0 que inclui o pacote de métricas. Pode ser que seja necessário que esta alteração repercuta em todas as estações do ambiente. Quando atualizar o gerenciador de projetos para a nova versão, o serviço de persistência irá depender de uma atualização no banco de dados de maneira a adiconar as tabelas para a persistência das métricas. Isto pode implicar na versão 2.0 do serviço de persistência. Assim, mesmo que a alteração em um gerenciador implique na alteração de mais componentes, a descrição de dependência entre os pacotes do *framework* garantirá a extensibilidade e a integridade do ambiente como um todo.

A incorporação da infra-estrutura de atualização ao *framework* garante que a extensabilidade do mesmo seja garantida internamente ao *framework*. Além de instalar novas ferramentas, remover e atualizar ferramentas já instaladas em uma estação de trabalho, a infra-estrutura de atualização do ambiente garante que a mesma pode ser feita sem a necessidade de reiniciar o ambiente ou parar suas atividades. Isto é garantido pelo uso dos arquivos descritores do ambiente e pela alteração dinâmica no CLASSPATH da aplicação.

As atualizações do ambiente podem ser críticas e obrigatórias, recomendadas ou facultativas. Desta maneira, mesmo o ambiente estando geograficamente distribuído é possível que o mesmo esteja configurado de maneira uniforme em todas as suas estações de trabalho.

Além da extensibilidade de cada estação de trabalho, o FRADE garante a extensibilidade do ambiente aonde ele se encontra por meio da escalabilidade de sua distribuição. A criação do Gerenciador de Locais e dos Serviços de Comunicação e Distribuição, permitem que a infra-estrutura do ambiente seja escalonada de maneira a incorporar novos usuários ou serviços garantido a reconfiguração automática do ambiente cada vez que isto ocorre.

Unindo a escalabilidade do ambiente com a extensibilidade do *framework* em cada estação de trabalho, chega-se a um nível de flexibilidade do *framework* que permite uma a transparência do ambiente tanto em nível de desenvolvimento quanto em nível de utilização.

4.3 Instanciação do *framework*

A instanciação do *framework* FRADE implica na instanciação dos componentes que fazem parte do mesmo, a partir do descritor XML de cada camada. A combinação de

diferentes componentes em cada camada poderá criar diferentes instâncias do *framework* para diferentes aplicações.

A instanciação dos componentes a partir de seu descritor XML é feita através de uma fábrica abstrata, ilustrada na Figura 104. Esta fábrica se encarregará de procurar nos arquivos XML a definição das interfaces a serem instanciadas e a classe que irá implementar esta interface. Isto permite que uma alteração seja feita em tempo de execução garantindo a integridade do ambiente e seu correto funcionamento após cada alteração.

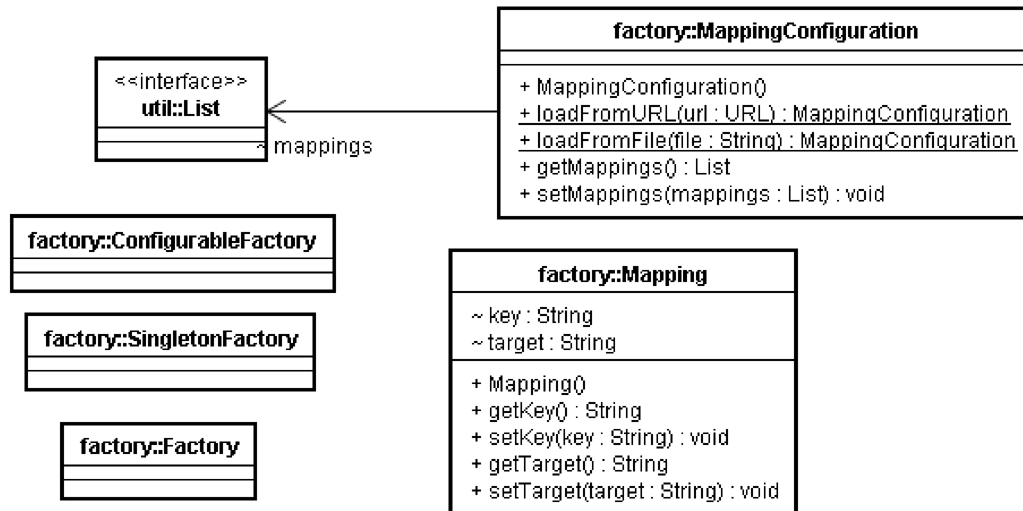

Figura 104 – Fábrica abstrata para a instanciação do *Framework*

Para que seja mais fácil às aplicações ou camadas encontrarem estas instâncias, uma classe será responsável por guardar a instância de cada camada do *framework*. A esta classe será dado o nome de Global ilustrada na Figura 105. A Classe Global possui métodos estáticos que permitirá as demais classes acessar cada camada do *framework* diretamente.

Figura 105 – Classe Global: referência a todas as camadas instanciadas

Além do acesso às camadas do *framework*, a classe Global guardará também o caminho de instalação do ambiente. O caminho de instalação do ambiente facilita o gerenciamento do *workspace* devido ao fato de os serviços locais utilizarem o sistema de arquivo para armazenar seus dados. O Global também armazena os argumentos que o ambiente poderá receber quando instanciado por meio de interface textual.

A utilização da classe Global é uma opção de projeto e a mesma se encontra fora do *framework* por ser uma opção para a instanciação do mesmo. Esta classe representa uma visão externa ao *framework*.

Por manter uma ligação com todas as camadas e a instanciação destas, a instanciação da classe Global representa a instanciação do FRADE. A Figura 106 apresenta o diagrama de atividades que devem ser executadas na instanciação desta classe e cada uma destas será explicada na seqüência.

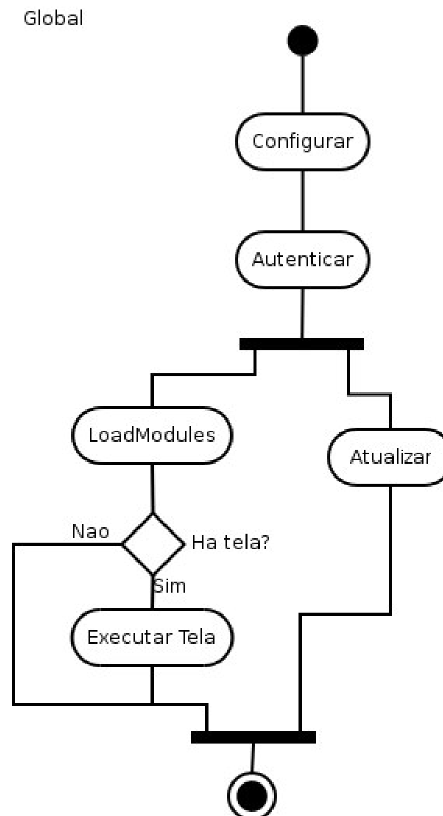

Figura 106 – Diagrama de atividades: Global

A atividade configurar pode ser dividida em várias partes: carregar o manipulador de Exceção, os argumentos para a aplicação, o caminho da aplicação, os locais para o suporte a distribuição e a aplicação principal. A Figura 107 apresenta a seqüência de atividades para a configuração do ambiente. A ordem destas atividades pode influenciar no comportamento do ambiente instanciado. O manipulador de exceções, por exemplo, é o primeiro a ser carregado, pois, caso haja algum problema na carga dos argumentos que gere uma exceção, a mesma deverá ser tratada por este manipulador.

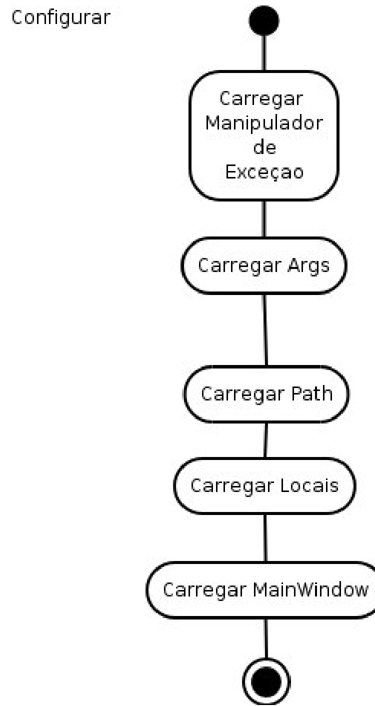

Figura 107 – Diagrama de atividades: configurar

A segunda atividade para a instanciação do *framework* é a autenticação de usuário. Esta atividade também pode ser decomposta em vários passos, como ilustra o diagrama de atividades da Figura 108. Um usuário é autenticado com usuário e senha em um determinado local. Caso o usuário não queira conectar-se, o canal de comunicação não será iniciado.

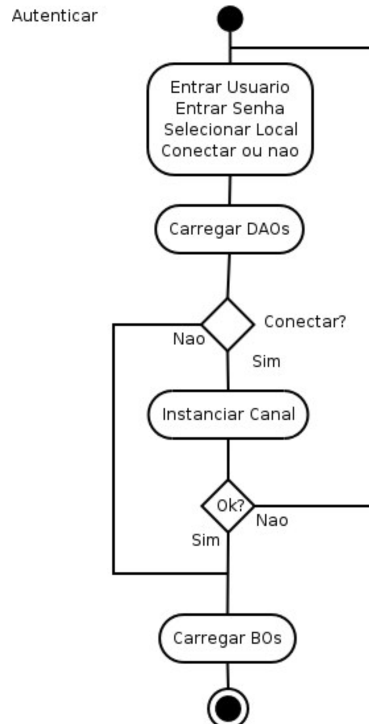

Figura 108 – Diagrama de atividades: Autenticar

A escolha do local para a conexão do usuário faz-se necessário pois um mesmo usuário pode participar de projetos distintos de organizações distintas. Neste caso, se ambas as organizações utilizarem o mesmo ambiente, o usuário poderá utilizar a mesma ferramenta para suas atividades. O local para a conexão é apresentado através de um ou mais nomes de locais.

O próximo passo para a instanciação é a carga dos módulos. Este passo é descrito pelo diagrama de atividades da Figura 109. Os módulos, como foi visto na infra-estrutura de atualização são os serviços, suportes, gerenciadores e ferramentas instaladas no ambiente. Um módulo pode ser auto-executável. Neste caso a seqüência de instanciação irá executá-lo.

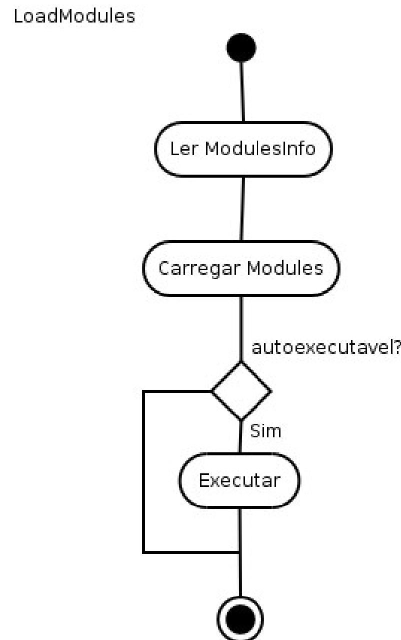

Figura 109 – Diagrama de atividades: LoadModules

Um módulo auto-executável pode ser uma ferramenta ou um serviço que é iniciado toda vez que o ambiente é iniciado. A possibilidade de haver módulos auto-executáveis permite que haja, no ambiente, servidores que não são utilizados como estações de trabalho.

No caso de um módulo carregado possuir interface gráfica, como as ferramentas, as GUIs do mesmo devem ser executadas. Uma ferramenta pode possuir ícones, botões ou menus para que o usuário possa iniciá-la no ambiente. A integração destes componentes de interface gráfica à aplicação principal preza por garantir uma homogeneidade do ambiente quanto à execução das ferramentas que estão integradas ao mesmo. O diagrama de atividades da Figura 110 ilustra a seqüência de atividades para a execução destas telas.

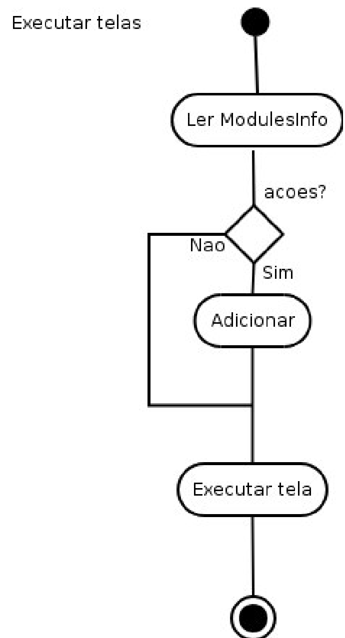

Figura 110 – Diagrama de atividades: Executar telas

Cada instância do FRADE pode alterar esta seqüência de inicialização modificando a implementação de sua classe Global. Isto permite que novas tarefas sejam consideradas no momento de instanciação do *framework* e que em determinadas estações de trabalho esta instanciação ocorra de maneira distinta. Um exemplo desta alteração seria a instanciação de uma estação sem interface gráfica. Neste caso a seqüência de instanciação não executará as telas para iniciar a aplicação principal.

4.4 Experimentos realizados

Finalizada a definição do *framework*, foram realizados experimentos no DiSEN utilizando-o. Para tanto, aproveitando a experiência adquirida, por parte dos membros do grupo, algumas ferramentas anteriormente prototipadas foram reimplementadas e outras foram criadas. Uma breve descrição destes experimentos estão a seguir.

Alguns trabalhos do mesmo grupo de pesquisa já tratavam de ferramentas para ADS. A validação do FRADE utilizou a reimplementação destas ferramentas com a utilização do *framework*. A ferramenta DiManager (PEDRAS, 2003) foi a primeira ferramenta a ser escolhida para ser reimplementada. A opção pela reimplementação desta ferramenta foi devido a sua ortogonalidade a todos os gerenciadores do ambiente. A DiManager é uma ferramenta para gerenciamento de projetos distribuídos, a ser utilizada por gerentes de projetos, para a definição de projetos de *software* incluindo a definição de processos. A DiManager inclui ainda a alocação de recursos para o projeto como os recursos humanos a serem utilizados e a divisão das atividades entre os usuários alocados em um projeto.

A DiManager foi integrada ao ambiente na forma de uma ferramenta/módulo. Seu funcionamento requer que todas as entidades do ambiente estejam corretamente relacionadas e, a ferramenta depende da persistência dos dados do projeto para seu funcionamento correto. Por esta razão, o serviço de persistência foi o primeiro a ser implementado.

Para a infraestrutura de persistência foi utilizado um banco de dados Potgresql e um banco MySQL com replicação de dados entre estes bancos. O serviço de persistência foi implementado em duas estações utilizando interfaces textuais para isto. Um protótipo do serviço de distribuição foi constuído para garantir a replicação deste serviço. Nesta validação não foi utilizada persistência local.

Para o desenvolvimento da DiManager foi utilizada uma ferramenta chamada GeCA (Gerador de Código Auxiliar). A ferramenta GeCA (STEINMACHER et al., 2006) foi implementada para auxiliar o desenvolvimento de aplicações que envolvem persistência sobre o FRADE. Ela baseia-se em modelos e cria aplicações que seguem o padrão MVC utilizando os DAOs e BOs para a comunicação com a persistência.

Além disto, a DiManager foi reimplementada utilizando os componentes gráficos da camada de aplicação para garantir uma homogeneidade de sua GUI com a do ambiente.

Estando integrada ao ambiente a ferramenta de gerência de projetos, surgiu a necessidade de prover as informações do projeto aos participantes do mesmo. Primeiramente um protótipo do serviço de acesso foi desenvolvido para a validação dos usuários.

Estando os usuários utilizando o ambiente surgiu a necessidade de desenvolver ferramentas para a comunicação entre os membros do projeto. Foi implementada uma ferramenta de Chat operando sobre a infra-estrutura de comunicação. Esta ferramenta exibe os usuários online e permite a troca de mensagens entre os mesmos.

Para garantir a cooperação e colaboração dentro do ambiente foi desenvolvida uma ferramenta de diagramação gráfica que permite o trabalho simultâneo de mais de um usuário sobre um mesmo desenho. Esta ferramenta foi chamada de DesenhadorRemoto e não é exatamente uma ferramenta específica para o domínio do *framework* aqui proposto mas cumpre o propósito de *awareness* em ferramentas de cooperação / colaboração a que o *framework* se propõe.

Uma outra ferramenta, a Requisite (BATISTA, 2003), teve também suas funcionalidades reimplementadas por WIESE (2006) e integrada ao ambiente. A Requisite é uma ferramenta de projeto para apoiar a fase de requisitos e gera diagramas de caso de uso. Sua integração permite que os artefatos gerados pela ferramenta sejam gravados apenas no *workspace* do usuário (repositório local).

Havendo a possibilidade de configurar ambientes distintos de maneira distinta, através da instalação das ferramentas aqui citadas, foi criada uma ferramenta para instalação/remoção de ferramentas que opera sobre o gerenciador de ferramentas. Por decisão de projeto esta ferramenta, chamada de Atualizador (SCHIAVONI, 2006). A partir da mesma, é possível configurar as demais ferramentas do ambiente.

Um resumo dos experimentos realizados pode ser visto na Tabela 9.

Tabela 9 – Resumo dos experimentos realizados

Itens	Implementação
Ferramentas desenvolvidas /Prototipadas	- DiManager para gerencia de projeto - Chat para comunicação - DesenhadorRemoto para *Awareness* - Requisite para elaboração de casos de uso - Atualizador para adição/remoção de ferramentas
Gerenciadores prototipados	- Gerenciador de acesso - Gerenciador de projetos - Gerenciador de processos - Gerenciador de recursos - Gerenciador de usuários
Interfaces de usuário implementadas	- GUI para integração de ferramentas - Textual para os servidores de persistência
Suportes/serviços implementados	- Persistência - Acesso (protótipo ainda não efetivo) - Distribuição (protótipo ainda não efetivo) - Comunicação - Atualização (apenas instalação de ferramentas) - Artefatos (apenas local)

4.5 Avaliação

A avaliação do FRADE baseou-se no estudo empírico das qualidades consideradas importantes a um *framework* tais como: usabilidade, reusabilidade, manutenibilidade e extesibilidade. A avaliação foi feita a partir da implementação do protótipo e ferramentas descritos na experimentação. A realização da experimentação e a avaliação sobre a mesma pretendem reforçar a usabilidade do modelo conceitual criado e verificar se os níveis de abstrações propostos atendem as expectativas do *framework*.

A primeira avaliação do FRADE foi quanto a usabilidade, para o desenvolvimento de ferramentas. A componentização do *framework* e a divisão do mesmo em camadas de abstração, simplificaram o desenvolvimento de ferramentas para ADDSs. Um desenvolvedor é capaz de implementar uma ferramenta para o ambiente utilizando para isto apenas os objetos de negócios. Isto torna o desenvolvimento distribuído mais simples pois as demais camadas do *framework* se encarregarão da conexão da ferramenta desenvolvida com os serviços necessários para o seu funcionamento.

Uma segunda avaliação tratou da extensibilidade do *framework*. A extensibilidade é garantida devido aos mecanismos de instanciação que garantem que componentes que implementam as interfaces propostas conseguem ser integrados ao ambiente, independentemente, de suas implementações. Apesar de não terem sido implementado todos os serviços propostos, a avaliação mostra que haverá pouco problema em integrá-los ao ambiente garantindo assim a sua extensibilidade. A implementação da infra-estrutura de atualização garante ainda que a extensibilidade pode ser alcançada em tempo de execução e, até mesmo, sem a intervenção do usuário para as atualizações críticas.

Outra facilidade constatada foi a integração dos módulos desenvolvidos ao ambiente através do suporte a atualização. Cada módulo proposto por interface (serviços, suportes e gerenciadores) pode ser implementado em um projeto separado e a integração e distribuição dos mesmos serão mantidos.

A manutenibilidade do *framework* é alcançada por meio da divisão de responsabilidades. A possibilidade de adicionar e remover serviços, por exemplo, garante a manutenção de uma das estações do ambiente instanciado sem interromper o funcionamento do ambiente como um todo. A manutenabilidade é ainda facilitada devido à implementação baseada em componentes e a utilização da arquitetura em camadas. Um componente ou toda uma camada pode ser alterada no ambiente sem que esta alteração repercuta no ambiente como um todo, desde que esta não necessite alterar a interface deste componente / camada. Além disto, dada a interdependência entre os componentes e camadas apresentada neste trabalho, caso uma interface necessite de manutenção é possível verificar quais são os demais componentes / camadas aonde esta alteração será repercutida.

Outra avaliação feita foi em relação à camada de comunicação. A abordagem adotada pretendia conseguir a conexão de estações que estivessem protegidas por NAT ou firewall ao ambiente. Mais que conectar ao ambiente, a interação com ferramentas de awareness pretendia ser alcançado. A ferramenta de Chat e o DesenhadorRemoto mostram que é possível a integração de máquinas com configurações distintas de acesso ao ambiente.

4.6 Contribuição

Dentro do contexto apresentado, as contribuições desta dissertação foram:

- O modelo de arquitetura em camadas e a separação em níveis lógicos das abstrações encontradas no domínio do *framework*;
- A definição dos gerenciadores e a componentização dos mesmos de maneira a atender o modelo de negócio para este domínio;
- A explicitação dos suportes e serviços, a componentização, a definição dos mesmos e a integração da infra-estrutura para o provimento das funcionalidades necessárias em ambientes de *software*;
- A definição da comunicação síncrona e assíncrona e a implementação de uma camada de comunicação que permite a integração de usuários em diversas configurações de rede, garantindo assim o aproveitamento da largura de banda através de mecanismos de compactação e redução das mensagens de comunicação;

- A integração, manutenitibilidade e extensibilidade do *framework* através de uma infra-estrutura integrada ao mesmo capaz de instanciar uma estação de trabalho com diversas configurações;
- A configuração dinâmica do ambiente distribuído através da infra-estrutura de comunicação e distribuição;
- A extensibilidade, homogeneidade e manutenibilidade dinâmica do ambiente por meio da utilização da infra-estrutura de atualização.

4.7 Conclusão

Neste capítulo foram apresentadas a instanciação do *framework* proposto, a seqüência de instanciação das classes e as classes de apoio para a instanciação. Foram apresentados também os resultados da validação com a construção de algumas ferramentas e a implantação de um primeiro ambiente implementado sobre o FRADE.

Retomando as características presentes nas ferramentas relacionadas do capítulo 2 notamos que o FRADE atende aos requisitos comuns a ADDS e, graças à facilidade de extensibilidade, pretende-se que o mesmo possa vir a atender os requisitos que surgirão para este domínio de aplicação.

A avaliação foi feita sobre a validação empírica baseando-se na construção do protótipo por meio de uma análise das características desejáveis a um *framework*: manutenibilidade, usabilidade, reusabilidade e extensibilidade. A avaliação do *framework* demonstrou que o mesmo atende às necessidades propostas.

5 CONCLUSÕES

Durante a idade média, o conhecimento na igreja católica resumia-se aos padres e cléricos. Isto se justificava pelo fato do alto clero achar que o conhecimento de Deus ser muito complexo para boa parte da população. Neste contexto surgiram algumas ordens religiosas que tinham por objetivo popularizar a informação. Por se dirigir uns aos outros por irmão (frater em latin), estes cléricos ficaram conhecido por frades. Para popularizar a informação sobre Deus, os frades católicos tiveram muitas vezes que trabalhar níveis de abstrações distintos para que um conceito complexo pudesse ser mais facilmente entendido pela população católica.

O desenvolvimento distribuído de *software* é um processo bastante complexo que envolve vários mecanismos, políticas e pessoas. O objetivo do FRADE é simplificar este processo criando níveis de abstrações distintas de modo que mesmo pessoas com níveis de conhecimento distintos consigam interagir com o ambiente.

Primeiramente, foram criados quatro níveis de abstração lógica: Aplicação, Negócios, Infra-estrutura e Comunicação. Cada nível de abstração possui uma responsabilidade distinta e um papel dentro do *framework*. A organização em camadas a partir destes níveis lógicos foi feita com intuito de permitir um aprofundamento dentro do *framework* quando necessário e, também, permitir uma visão de alto nível que apresente um conteúdo coeso.

Dentro de cada camada foram definidos vários componentes sendo que, dependendo da camada, estes componentes possuem nomeações distintas. O termo gerenciador foi utilizado para a camada de negócios e os termos suporte e serviço foram utilizados para a camada de infra-estrutura. A definição de componentes permite que cada funcionalidade específica do sistema possa ser isolada e tratada de maneira diferente, com políticas distintas e com implementações, parcialmente, independentes.

Além da definição do conteúdo de cada componente e de sua interface e dependências foi feita uma definição conceitual da sua funcionalidade. Isto porque estamos certos de que este *framework* em sua primeira versão não atende completamente todos os requisitos e cenários possíveis para ADDS.

Para a definição das camadas e componentes foi necessário um aprofundamento no estudo de ambientes existentes apresentados na seção de trabalhos relacionados. Com este aprofundamento foram identificados requisitos comuns a ADSs e também as nomeclaturas comumente utilizadas neste domínio de aplicação. O estudo de trabalhos relacionados também apontou a evolução de alguns ADSs e as decisões tomadas por grupos de projetos. Estas decisões foram analisadas e incorporadas ao *framework*.

Alguns termos comumente utilizados na informática mas que possuem uma certa amplitude de significado na literatura também foram elucidados. Uma definição *framework*, componentes, Desenvolvimento Global de *software*, sistemas em camadas, sistemas distribuídos e Ambientes de desenvolvimento de *software* foi apresentada para posicionar o entendimento destes conceitos no contexto deste trabalho.

Findadas as definições e a conceitualização do *framework* foi feita uma avaliação e uma validação através da prototipação de alguns dos componentes propostos neste trabalho. A avaliação mostrou que o objetivo de simplificar o desenvolvimento por meio de abstrações lógicas foi conseguido e que o desenvolvimento de ADDSs e de ferramentas para ADDS se tornou mais simples e rápido.

Entre os objetivos alcançados com este trabalho, a configuração dinâmica de ambientes e a capacidade de awareness foram focadas e alcançadas. Para permitir *awareness* entre participantes de um projeto foi criado um protocolo de comunicação próprio e foi definida toda a camada de comunicação, no intuito de simplificar o desenvolvimento de ferramentas colaborativas / cooperativas.

Para a configuração dinâmica, alguns mecanismos extras foram criados tais como a infra-estrutura de distribuição, comunicação e atualização. Estes mecanismos, que foram definidos como suportes e serviços na camada de Infra-Estrutura, permitem que a configuração do ambiente possa ser feita em tempo de execução e de maneira transparente ao seu usuário.

5.1 Limitações

O *framework* proposto por este trabalho possui as seguintes limitações:

- A implementação do FRADE limitou-se a resolver os problemas de comunicação, persistência, cooperação, colaboração e configuração dinâmica. A persistência de artefatos em um repositório remoto, por exemplo, não foi solucionada nesta primeira versão;
- As políticas de acesso às informações do *framework* não foram definidas. Neste estágio de implementação o *framework* possui mecanismo que garantem o acesso às informações do ambiente, porém não trata a autorização deste acesso para os diferentes níveis de usuário que o ambiente pode ter;
- A definição dos gerenciadores não inclui todos os cenários possíveis dentro de gerenciamento de projeto. Características desejáveis como métricas, análise de riscos e desenvolvimento colaborativo de artefatos não foram consideradas, mas encontram-se em desenvolvimento por membros da equipe;
- O modelo de negócios proposto pelo FRADE inclui a distribuição através da replicação dos dados persistidos mas não prevê a fragmentação da base de dados;
- A criação de federações de ADSs não foi prevista por este trabalho.

Entre os inúmeros fatores que levaram à aceitação destas limitações encontra-se o fator tempo de desenvolvimento e pesquisa para que a superação das limitações fosse possível. Por esta razão, serão apresentados a seguir os trabalhos futuros que pretendem alcançar o fim destas limitações.

5.2 Trabalhos futuros

Entre os trabalhos futuros a esta dissertação encontram-se:

- Definição e implementação da infra-estrutura de acesso incluindo técnicas de criptografia, algoritmos de chaves públicas/privadas e assinatura digital;

- Definição e implementação da infra-estrutura de artefato incluindo o desenvolvimento cooperativo, repositórios locais, sincronização e suporte às várias ferramentas de SCM existentes no mercado;
- Definição de implementação da infra-estrutura de atualização por meio de um repositório de atualização ativo aonde mais que atualizar sua estação o usuário possa interagir com o repositório;
- Definição e implementação de um repositório de dados para trabalho cooperativo;
- Definição das políticas de acesso em suas várias granularidades e a criação de mecanismos para a alteração destas políticas em tempo de execuação;
- Definição e implementação de um modelo de ADS que suporte federação de ADDSs;

Referências

AMBRIOLA, V., CIANCARINI, P., MONTANGERO, C. **Software process enactment in Oikos**. In Proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environments (Irvine, California, United States, December 03 - 05, 1990). R. N. Taylor, Ed. SDE 4. ACM Press, New York, NY, 183-192. 1990. DOI= http://doi.acm.org/10.1145/99277.99294

AMBRIOLA, V., CONRADI, R., FUGGETTA, A.. **Assessing process-centered software engineering environments.** ACM Transactions on Software Engineering and Methodology. 6(3):283—328. July 1997. http://citeseer.ist.psu.edu/article/ambriola97assessing.html

ANEROUSIS, N. **An Architecture for Building Scalable**, Web-Based Management Services. J. Netw. Syst. Manage. 7, 1 (Mar. 1999), 73-104. 1999. DOI= http://dx.doi.org/10.1023/A:1018717900333

ARAÚJO, Marco Antônio Pereira. **Automatização do Processo de Desenvolvimento de Software nos Ambientes Instanciados pela Estação TABA.** Dissertação de Mestrado COPPE/UFRJ. Rio de Janeiro, 1998.

AVERSANO, L., CIMITILE, A., DE LUCIA, A., **A Communication Protocol for Distributed Process Management**, Workshop on Global software development, collocated to the International Conference on Software Engineering, ICSE 2003, p. 16-20.

AVERSANO, L., CIMITILE, A., LUCIA, A., STEFANUCCI, S., VILANN, M. L., **Workflow Management in the GENESIS Environment.** Research Center on Software Technology, Department of Engineering, University of Sannio, 2003.

AVERSANO, Lerina, LUCIA, Andrea De, GAETA, Matteo, RITROVATO, Pierluigi, VILLANI, Maria-Luisa. **Managing Distributed Projects in GENESIS.** SPT, Software Process Technology. UP GRADE - the European Journal for the Informatics Professional. Vol V. N.5. Outubro 2004.

BALLARINI, D., CADOLI, M., GAETA, M., MANCINI, T., MECELLA, M., RITROVATO, P., SANTUCCI, G.. **Modeling Real Requirements for Cooperative Software Development: A Case Study**. In Proc. 2nd Work. on Cooperative Supports for Distributed Software Engineering Processes. March 2003. Benevento, Italy.

BANASZEWSKI, R.F. **Gerenciamento de Padrões para aplicação em Ambientes de desenvolvimento Distribuídos**. Monografia (Conclusão de Curso Bacharelado em Análise de Sistemas) 140 pg. Departamento de Ciência da Computação. Universidade Estadual do Centro Oeste. 2006.

BANDINELLI, S. C., FUGGETTA, A., and GHEZZI, C. **Software Process Model Evolution in the SPADE Environment**. IEEE Trans. Softw. Eng. 19, 12 (Dec. 1993), 1128-1144. 1993. DOI= http://dx.doi.org/10.1109/32.249659

BARGHOUTI, N. S. **Supporting cooperation in the Marvel process-centered SDE**. In Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development

Environments (Tyson's Corner, Virginia, United States, December 09 - 11, 1992). SDE 5. ACM Press, New York, NY, 21-31. 1992. DOI= http://doi.acm.org/10.1145/142868.142907

BARGOUTHI, Naser S., EMMERICH, Wolfgang, SCHÄFER, Wilhelm, SKARRA, Andrea. **Information management in process-centered software engineering environments**. In Alfonso Fuggetta and Alexander Wolf, editors, Software Process. John Wiley & Sons. 1996.

BARGHOUTI, N.S., KAISER, G.E., **Scaling Up Rule-Based Software Development Environments.** In A. van Lamsweerde, A. Fugetta (eds.) Proc. of the 3 rd European Software Engineering Conference, Milan. 1991. http://citeseer.ist.psu.edu/barghouti91scaling.html

BARTHELMESS, P., **Collaboration and Coordination in Process-Centered Software Development Environments: A Review of the Literature**, Information and Software Technology, 45(13). 2003. 911--928.

BASILE, C., CALANNA, S., NITTO, E. D., FUGGETTA, A., and GEMO, M. **Mechanisms and Policies for Federated PSEEs: Basic Concepts and Open Issues**. In Proceedings of the 5th European Workshop on Software Process Technology (October 09 - 11, 1996). C. Montangero, Ed. Lecture Notes In Computer Science, vol. 1149. Springer-Verlag, London, 86-91. 1996.

BATISTA, Sueleni Mendez. **Uma ferramenta de apoio à fase de requisitos da mdsodi no contexto do ambiente DiSEN**. Dissertação (Mestrado) – Programa de Pós-Graduação em Informática, Universidade Federal do Paraná. Curitiba, 2003.

BECKER, S., JÄGER, D., SCHLEICHER, A., and WESTFECHTEL, B. **A Delegation Based Model for Distributed Software Process Management**. In Proceedings of the 8th European Workshop on Software Process Technology (June 19 - 21, 2001). V. Ambriola, Ed. Lecture Notes In Computer Science, vol. 2077. Springer-Verlag, London, 130-144. 2001.

BELOGLAVEC, S., HERIČKO, M., JURIČ, M. B., and ROZMAN, I. **Analysis of the limitations of multiple client handling in a Java server environment**. *SIGPLAN Not.* 40, 4 (Apr. 2005), 20-28. 2005. DOI= http://doi.acm.org/10.1145/1064165.1064170

BELKHATIR, N., MELO, W.L., **Supporting software maintenance processes in TEMPO** in Proceedings of the Conference on Software Maintenance, IEEE Computer Society Press. September 1993. http://citeseer.ist.psu.edu/belkhatir93supporting.html

BELKHATIR, N., ESTUBLIER, J., MELO, W. L.. **Cooperative Work in Large Scale Software Systems**. Software Maintenance, Research and Practice, Vol. 6, 319-335, 1994. http://citeseer.ist.psu.edu/belkhatir95cooperative.html

BELKHATIR, N., MELO, W.L., **Evolving Software Processes by Tailoring the Behavior of Software Objects**. Proc. Intl. Conf. on Software Maintenance, Victoria, Canada, September 1994-b. http://citeseer.ist.psu.edu/belkhatir94evolving.htm

BEN-SHAUL, I., HEINEMAN, G. T. **A 3-level Atomicity Model for Decentralized Process-Centered Software Engineering Environments**. In Proceedings of the 5th European Workshop on Software Process Technology (October 09 - 11, 1996). C.

Montangero, Ed. Lecture Notes In Computer Science, vol. 1149. Springer-Verlag, London, 61-64. 1996.

BEN-SHAUL, I. Z., KAISER, G. E., HEINEMAN, G. T. **An architecture for multi-user software development environments**. In Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments (Tyson's Corner, Virginia, United States, December 09 - 11, 1992). SDE 5. ACM Press, New York, NY, 149-158. 1992. DOI= http://doi.acm.org/10.1145/142868.143765

BEN-SHAUL, I.Z., KAISER, G.E., **A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment**, 16 th International Conference on Software Engineering, IEEE CS Press, 179-188. 1994. http://citeseer.ist.psu.edu/article/ben-shaul94paradigm.html

BEN-SHAUL, Israel, KAISER, Gail E.. **A Paradigm for Decentralized Process Modeling**. Kluwer Academic Publishers, Boston/Dordrecht/London, 1st edition, 1995. ISBN 0-7923-9631-6.

BEN-SHAUL, I., KAISER, G., **A paradigm for decentralized process modelling and its realization in the Oz environment**, in Proc. 16th International Conference on Software Engineering, pp. 179-188. 1996.

BEN-SHAUL, I. Z. and KAISER, G. E. **Federating Process-Centered Environments: The Oz Experience**. *Automated Software Engg.* 5, 1 (Jan. 1998), 97-132. 1998. DOI= http://dx.doi.org/10.1023/A:1008610426207

BIANCHI, A., CAIVANO, D., LANUBILE, F., VISAGGIO, G., **Defect Detection in a Distributed Software Maintenance Project**, Proc. of the International Workshop on Global Software Development (GSD 2003), Portland, Oregon, USA, May 2003.

BIANCHI, Silvia Cristina Sardela, SALES, André Barros De, WESTPHALL, Carlos Becker, SIBILLA, Michelle, WESTPHALL, Carla Merkle. **Gerenciamento dos Recursos do Sistema em Ambientes distribuídos usando WBEM/CIM**. Dans : II Encontro de Ciência e Tecnologia (II ECTec), Lages (Brésil), 29/09/03-03/10/03, septembre 2003.

BLOIS, A. P., BECKER, K., WERNER, C. M. L. . **Um Processo de Engenharia de Domínio com foco no Projeto Arquitetural Baseado em Componentes**. In: Quarto Workshop de Desenvolvimento Baseado em Componentes, 2004, João Pessoa. Quarto Workshop de Desenvolvimento Baseado em Componentes. João Pessoa : UFPB, 2004. v. 1. p. 15-20.

BOLDYREFF, Cornelia, DEWAR, Rick, SMITH, Mike, WEISS, Dawid, NUTTER, David, WILCOX, Pauline, RANK, Stephen. **Environments to Support Collaborative Software Engineering**. 2nd Workshop on Cooperative Supports for Distributed Software Engineering Processes, Benevento, Italy, March 25, 2003.

BRAUN, Torsten and DIOT, Christophe. **Protocol Implementation Using Integrated Layer Processnig**, in Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM) . ACM, September 1995. http://citeseer.ist.psu.edu/braun95protocol.html

BRAUN, Andreas, DUTOIT, Allen H., BRUEGGE, Bernd, **A Software Architecture for Knowledge Acquisition and Retrieval for Global Distributed Teams**, International Workshop on Global Software Development, International Conference on Software 191 Engineering. Portland, Oregon, May 9, 2003. http://citeseer.ist.psu.edu/braun03software.html

BROWN, A. L. **Persistent Object Stores**, Ph.D thesis, Computational Science, University of St. Andrews, 1988. http://citeseer.ist.psu.edu/brown88persistent.html

BROWN, AL, DEARLE, A, MORRISON, R, MUNRO, D, ROSENBERG, J. **A Layered Persistent Architecture for Napier88**. In: Security and Persistence*, Rosenberg, J, Keedy, JL (eds),* Proc. International Workshop on Security and Persistence, Bremen, 1990*,* pp 155-172*.* Springer-Verlag. 1990.

CAVANNES, Chuck. **Programming Jakarta Struts** . Greenwich: O ' Reilly Publications, 2003.

CESETTI, O., AKAMATU, D. M., KIRNER, C. . **Paradigmas para a Construção de Sistemas Distribuidos**. Revista Tema, Brasília - DF, v. 24, n. 114, p. 1-4, 1993.

CHAMBERS, John M., LANG, Duncan Temple , **Omega- A Component-based Statistical Computing Environment**. Bulletin of the International Statistical Institute. Finland 1999. Disponível em <http://www.stat.fi/isi99/proceedings/arkisto/varasto/temp0984.pdf>. Acessado em Novembro/2006.

CLEMENTS, P. C. 2001. **From subroutines to subsystems: component-based software development**. In Component-Based Software Engineering: Putting the Pieces Together, G. T. Heineman and W. T. Councill, Eds. Addison-Wesley Longman Publishing Co., Boston, MA, 189-198.

COLS, D. R. 1993. **Business-structured client/server: an architecture for distributed applications**. In Proceedings of the 1993 Conference of the Centre For Advanced Studies on Collaborative Research: Software Engineering - Volume 1 (Toronto, Ontario, Canada, October 24 - 28, 1993). A. Gawman, E. Kidd, and P. Larson, Eds. IBM Centre for Advanced Studies Conference. IBM Press, 41-53.

CONRADI, R., LIU, Ch.: **Process Modelling Languages: One or Many?**, in Software Process Technology - Fourth European Workshop EWSPT'95, LNCS 913, Springer, Berlin, 1995; pages 98-118. http://citeseer.ist.psu.edu/conradi94process.html

CONRADI, R., LIU, C., HAGASETH, M. **Planning support for cooperating transactions in EPOS**. Inf. Syst. 20, 4 (Jun. 1995), 317-336. 1995. DOI= http://dx.doi.org/10.1016/0306-4379(95)00017-X

CONRADI, R., OSJORD, E., WESTBY, P. H., LIU, C. **Initial software process management in EPOS**. Softw. Eng. J. 6, 5 (Sep. 1991), 275-285. 1991.

COULOURIS, George, DOLLIMORE, Jean, KINDBERG, Ti. **Distributed Systems: Concepts and Design.** Pearson. 2000

CREIGHTON, O., DUTOIT, A.H., BRUEGGE, B.. **Supporting an Explicit Organizational Model in Global Software Engineering Projects**. In International Workshop on Global Software Development, International Conference on Software Engineering. Portland, Oregon, May 9, 2003.

CUGOLA, G., GHEZZI, C. **Software Processes: a Retrospective and a Path to the Future**, Software Process Improvement and Practice, 4, 101-123. 1998. http://citeseer.ist.psu.edu/cugola98software.html

CUGOLA, Gianpaolo, GHEZZI, Carlo. **Design and implementation of PROSYT: a distributed process support system**. In Proceedings of the 8th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, California, USA, June 1999. Stanford University. http://citeseer.ist.psu.edu/cugola99design.html

CUGOLA, G., CUNIN, P., DAMI, S., ESTUBLIER, J., FUGGETTA, A., PACULL, F., RIVIERE, M., VERJUS, H. **Support for Software Federations: The PIE Platform**. In Proceedings of the 7th European Workshop on Software Process Technology (February 21 - 25, 2000). R. Conradi, Ed. Lecture Notes In Computer Science, vol. 1780. Springer-Verlag, London, 38-53. 2000.

DAMIAN, D., CHISAN, J., ALLEN, P., CORRIE, B, **Awareness meets requirements management: awareness needs in global software development**, Proc. of the Int'l Workshop on Global Software Development, International Conference on Software Engineering (ICSE 2003), Portland, Oregon, May 2003

DAMIAN, D., LANUBILE, F., OPPENHEIMER, H. L. **Addressing the challenges of software industry globalization: the workshop on Global Software Development**. In Proceedings of the 25th international Conference on Software Engineering (Portland, Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 793-794. 2003.

DEARLE, A. **On the Construction of Persistent Programming Environments**. Ph.D. Thesis, University of St Andrews. 1988. http://citeseer.ist.psu.edu/dearle88construction.html

DERNIAME, J., KABA, B., WASTELL, D. (Eds.). **Software Process: Principles, Mehodology and Technology**. Lecture Notes in Computer Science, 1500. Springer, 1999.

DIAS, Viviane C., PIETROBOM, Carlos A. M., ALVES, M. F. **Definição de um Ambiente para Apoiar a Gerência de Conhecimento para Projeto de Banco de Dados Utilizando um Framework Conceitual e Padrões de Análise**.. In: III Simposio Brasileiro de Qualidade de Software - II Worwshop de Tecnologia da Informação e Gerência do Conhecimento - WGC - 2004, 2004, Brasilia. Anais do III Simposio Brasileiro de Qualidade de Software, 2004.

DOGAC, A., DENGI, C., SZU, M.T.. **Distributed Object Computing Platforms**. Communications of the ACM. September 1998, Vol. 41. No. 9. http://citeseer.ist.psu.edu/dogac98distributed.html

DOSSICK, S. E., KAISER, G. E. **CHIME: a metadata-based distributed software development environment**. SIGSOFT Softw. Eng. Notes 24, 6 (Nov. 1999), 464-475. 1999. DOI= http://doi.acm.org/10.1145/318774.319264

EMMERICH, Wolfgang. **Engineering of Distributed Objects.** Editora John Wiley & Sons. 2000.

ESTUBLIER, Jacky, CASALLAS, Rubby. **The Adele configuration manager**. In Walter F. Tichy, editor, Configuration Management. Trends in Software. John Wiley & Sons, 1994. http://citeseer.ist.psu.edu/estublier94adele.html

ESTUBLIER, J.. **Software configuration management: a roadmap**. In A. Finkelstein, editor, "The Future of Software Engineering", Special Volume published in conjunction with ICSE 2000, 2000. http://citeseer.ist.psu.edu/estublier00software.html

FEINSTEIN, W. P. **A study of technologies for Client/Server applications**. In Proceedings of the 38th Annual on Southeast Regional Conference (Clemson, South Carolina, April 07 - 08, 2000). ACM-SE 38. ACM Press, New York, NY, 184-193. 2000. DOI= http://doi.acm.org/10.1145/1127716.1127757

FELDMAN, Stuart. **Software Practice and Experience**, 9(3):255--265, March 1979. http://citeseer.ist.psu.edu/feldman79make.html

FERREIRA, Fábio Martins Gonçalves. **Desenvolvimento e Aplicações de um Framework Orientado a Objetos para Análise Dinâmica de Linhas de Ancoragem e de Risers**. 2005. Dissertação (Mestrado em Engenharia Civil) - Universidade Federal de Alagoas

FRANCH, Xavier, RIBO, Josep M. **Some reflexions in the modeling of software process**. procs França: International Process Technology Workshop,1999.

FREGONESE, G., ZORER, A., CORTESE, G. **Architectural framework modeling in telecommunication domain**. In *Proceedings of the 21st international Conference on Software Engineering* (Los Angeles, California, United States, May 16 - 22, 1999). International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos, CA, 526-534. 1999.

FREITAS, A., MAIA, A., NUNES, D. **Um Modelo para Interação entre Processos de Software**. In: CONGRESSO BRASILEIRO DE COMPUTAÇÃO, CBCOMP, 4., 2004, Itajaí. Anais. Itajaí: Univali, 2004. p. 149 p 154

FREITAS, Ana Vitoria Piaggio de, MAIA, Anderson Baia, NUNES, D. J. . **Gerenciamento da Integração entre Processos de Software no APSEE-Integrate**. In: 30a CLEI - Conferência Latino Americana de Informática, 2004-b, Arequipa. Anais da 30a Conferência Latino Americana de Informática, 2004. v. 1. p. 620-631.

FREITAS, A. **APSEE-Global: um Modelo de Gerência de Processos Distribuídos de Software**. 2005. 255 f. Dissertação (Mestrado em Ciência da Computação). Instituto de Informática, UFRGS, Porto Alegre.

FUJIEDA, K., OCHIMIZU, K. **Investigation of Repository Reprecation Models in Globally Distributed Configuration Management**. In Proc. of the Workshop on Global Software Development at ICSE. 2003.

GARLAN, David, SHAW, Mary, **An Introduction to Software Architecture**, Advances in Software Engineering and Knowledge Engineering, Volume 1, World Scientific Publishing Co., 1993.

GARLAN, D., SHAW, M. **An Introduction to Software Architecture**. Technical Report. UMI Order Number: CS-94-166., Carnegie Mellon University. 1994.

GERMAN, D.M. **GNOME, a case of open source global software development**. In International Workshop on Global Software Development, p. 39-43. 2003. <gsd2003.cs.uvic.ca/gsd2003proceedings.pdf>

GIMENES, I., BARROCA, L., HUZITA, E., CARNIELLO, A. **O Processo de Desenvolvimento Baseado em Componentes Através de Exemplos** *VIII Escola de Informatica da SBC Sul*, pp. 147-178 Raul Ceretta Nunes (ed) (ed.) Editora da UFRGS. 2000.

GIMENES, Itana M de S., HUZITA, Elisa H. M.. **Desenvolvimento baseado em componentes.** Editora Ciência Moderna. 2005.

GOMAA, Hassan, MENASCÉ, Daniel A., SHIN, Michael E. **Reusable component interconnection patterns for distributed software architectures.** Journal Title: ACM SIGSOFT Symposium on Software Reusability. 2001.

GRUNDY, J.C. **Visual specification and monitoring of software agents in decentralised process-centred environments**, International Journal on Software Engineering and Knowledge Engineering: Special Issue on Visual Programming for Parallel and Distributed Computing, Vol. 9, No. 4, World Scientific Publishing Company, August 1999, pp. 425-444. http://citeseer.ist.psu.edu/grundy99visual.html

GRUNDY, J. C., APPERLEY, M. D., HOSKING, J. G., and MUGRIDGE, W. B. **A Decentralized Architecture for Software Process Modeling and Enactment**. IEEE Internet Computing 2, 5 (Sep. 1998), 53-62. 1998. DOI= http://dx.doi.org/10.1109/4236.722231

GULLA, B., KARLSSON, E., YEH, D. **Change-oriented version descriptions in EPOS**. Softw. Eng. J. 6, 6 (Nov. 1991), 378-386. 1991.

HEIMANN, P., JOERIS, G., KRAPP, C.-A., WESTFECHTEL, B.: **DYNAMITE: Dynamic Task Nets for Software Process Management**, in Proc. of the 18 th Int. Conf. on Software Engineering, IEEE Computer Society Press, 1996, pages 331-341.

HEIMANN, P., KRAPP, C.-A., WESTFECHTEL, B.. **An environment for managing software development processes**. In Proc. 8th Conf. on Software Engineering Environments, pages 101--109, Cottbus, Germany, Apr. 1997. http://citeseer.ist.psu.edu/heimann97environment.html

HERBSLEB, J. D., MOCKUS, A., FINHOLT, T. A., GRINTER, R. E. **An empirical study of global software development: distance and speed**. In Proceedings of the 23rd

international Conference on Software Engineering (Toronto, Ontario, Canada, May 12 - 19, 2001). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 81-90. 2001.

HERBSLEB, James D., MOITRA, Deependra, **Guest Editors' Introduction: Global Software Development**, IEEE Software ,vol. 18, no. 2, pp. 16-20, March/April, 2001.

HEUSER, L. **Development of distributed and client/server object-oriented applications (panel): industrial solutions**. In Addendum To the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum) (Portland, Oregon, United States, October 23 - 28, 1994). M. C. Wilkes, Ed. OOPSLA '94. ACM Press, New York, NY, 59-62. 1994. DOI= http://doi.acm.org/10.1145/260028.260116

HITOMI, A. S., BOLCER, G. A., TAYLOR, R. N. **Endeavors: a process system infrastructure**. In Proceedings of the 19th international Conference on Software Engineering (Boston, Massachusetts, United States, May 17 - 23, 1997). ICSE '97. ACM Press, New York, NY, 598-599. 1997. DOI= http://doi.acm.org/10.1145/253228.253491

HITOMI, A. S., LEE, D. **Endeavors and Component Reuse in Web-Driven Process Workflow**, California Software Symposium, 1998. http://citeseer.ist.psu.edu/335440.html

HITOMI, A. S., KAMMER, P. J., BOLCER, G. A., TAYLOR, R. N.. **Distributed Workflow using HTTP: Example using Software Pre-requirements**. Proceedings of the 20 th International Conference in Software Engineering, April 1998-b http://citeseer.ist.psu.edu/hitomi98distributed.html

HOLDER, Ophir, BEN-SHAUL, Israel, GAZIT, Hovav. **System Support for Dynamic Layout of Distributed Applications**. In Proceedings of the 19th IEEE international Conference on Distributed Computing Systems (May 31 - June 04, 1999). ICDCS. IEEE Computer Society, Washington, DC, 403. 1999.

HUZITA, Elisa. H. M. **Uma Metodologia para o Desenvolvimento Baseado em Objetos Distribuídos Inteligentes**. Projeto de pesquisa, Universidade Estadual de Maringá. Departamento de Informática. 1999.

HUZITA, Elisa. H. M. **Suporte a reutilização em ambientes distribuídos de desenvolvimento de Software.** Projeto de pesquisa em andamento, Universidade Estadual de Maringá. Departamento de Informática, 2005.

KAISER, G. E., BARGHOUTI, N. S., SOKOLSKY, M. H.. **Preliminary experience with process modeling in the Marvel software development environment kernel**. In Proc. of the 23th Annual Hawaii Int'l Conf. on System Sciences, pages 131--140, Kona, HI, January 1990.

KAISER, Gail E., BEN-SHAUL, Israel Z., BARGHOUTI, Naser S.. **Preliminary Design of an Object Management System for Multi-User MARVEL**. In Takuya Katayama (editor), 6th International Software Process Workshop: Support for the Software Process. IEEE Computer Society Press, Hakodate, Hokkaido, Japan, October, 1990. Position paper. In press.

KAISER, Gail E., BEN-SHAUL, Israel Z., HEINEMAN, George T., MARRERO, Wilfredo. **Process evolution for the marvel environment**. Technical Report CUCS-047-92, Columbia University Department of Computer Science, April 1993. Submitted for publication. http://citeseer.ist.psu.edu/kaiser92process.html

KAISER, Gail E., POPOVICH, Steven S., BEN-SHAUL, Israel Z.. **A Bi-Level Language for Software Process Modeling**. In 15th International Conference on Software Engineering, pages 132--143, Baltimore MD, May 1993. IEEE Computer Society Press. http://citeseer.ist.psu.edu/kaiser93bilevel.html

KIEL, L., **Experiences in Distributed Development: A Case Study**, Proceedings of the International Workshop on Global Software Development at the 25th International Conference on Software Engineering, Portland, Oregon (2003), pp. 44–47, http://gsd2003.cs.uvic.ca/gsd2003proceedings.pdf.

LANUBILE, F., **A P2P Toolset for Distributed Requirements Elicitation**, Proc. of the ICSE Workshop on Global Software Development (GSD 2003), Portland, Oregon, USA, May 2003.

LANUBILE, Filippo, DAMIAN, Daniela, OPPENHEIMER, Heather L. **Global Software Development: Technical, Organizational, and Social Challenges.** ACM SIGSOFT Software Engineering Notes. Volume 28 Number 6. November 2003.

LAVVA, B., HOLDER, O., BEN-SHAUL, I. **Object management for network-centric systems with mobile objects**. In Proceedings of the Third IEEE international Conference on Engineering of Complex Computer Systems (ICECCS '97) (September 08 - 12, 1997). ICECCS. IEEE Computer Society, Washington, DC, 186. 1997.

LIMA, Fabiana. **Mecanismos de Apoio ao Gerenciador de Recursos Humanos no Contexto de um Ambiente Distribuído de Software**. 2004, 86 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, Universidade Estadual de Maringá, Maringá, 2004.

LIMA, Edson Pinheiro de; LEZANA, Álvaro Guilhermo Rojas. **Desenvolvendo um framework para estudar a ação organizacional: das competências ao modelo organizacional**. Gestão & Produção; São Paulo, 2005, 12, 2, 177-90, maio/ago. 2005.

LIMA REIS, Carla Alessandra. **Um gerenciador de processos de software para o ambiente PROSOFT**. Porto Alegre, 1998. 197 f.. Dissertação (Mestrado em Informática) - PPGC, Universidade Federal do Rio Grande do Sul

LIMA REIS, C. A., REIS, Rodrigo Quites, NUNES, Daltro José . **A Synchronous Cooperative Architecture For The Prosoft Software Engineering Environment**. In: IV Congresso Argentino de Ciencias de la Computacion - CACIC'98, 1998-b, Neuquen, Argentina. ANAIS.

LIMA REIS, Carla Alessandra, REIS, Rodrigo Quites, NUNES, Daltro José. **Gerenciamento do Processo de Desenvolvimento Cooperativo de Software no Ambiente PROSOFT**. In: XII SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (SBES'98), 12, 1998, Maringá, Paraná. Anais.... Maringá: SBC/UEM, 1998. p. 221 – 236

LEE, Dongman, LIM, Mingyu, HAN, Seunghyun. **ATLAS: a scalable network framework for distributed virtual environments.** September 2002 Proceedings of the 4th international conference on Collaborative virtual environments.

LEWANDOWSKI, S.M., **Frameworks for Component-Based Client/Server Computing**, ACM Computing Surveys, Vol. 30, No. 1, Mar. 1998. http://citeseer.ist.psu.edu/article/lewandowski98frameworks.html

LERNER, B.S., NINAN, A.G., OSTERWEIL, L.J., PODOROZHNY, R.M.. **Modeling and Managing Resource Utilization in Process, Workflow, and Activity Coordination**. Technical Report UM-CS-2000-058, Department of Computer Science, University of Massachusetts, August 2000. Disponível em http://laser.cs.umass.edu/publications/?category=PROC

LOPES, L. G., MURTA, L. G. P., WERNER, C. M. L.. **Controle de modificações em software no desenvolvimento baseado em componentes**. In: 2o. Workshop de Manutenção de Software Moderna, 2005, Manaus. 2o. Workshop de Manutenção de Software Moderna, 2005. p. 82-97.

MAIA, N., BLOIS, A. P., WERNER, C. M. L.. **Odyssey-MDA: Uma Ferramenta para Transformação de Modelos UML**. In: XIX Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas, 2005, Uberlândia. XIX Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas, 2005. p. 51-56.

MANGAN, M., WERNER, C. M. L., BORGES, M.. **Especificação de Componentes de Software em Sistemas Colaborativos**. In: II Workshop em Desenvolvimento Baseado em Componentes, 2002, Itaipava. II Workshop em Desenvolvimento Baseado em Componentes, 2002.

MAURER, F., SUCCI, G., HOLZ, H., KOTTING, B., GOLDMAN, S., DELLEN, B.. **Software process support over the internet**. In Proceedings 21st International Conference on Software Engineering, Los Angeles, May 1999.

MIAN, P.G., NATALI, A.C.C., FALBO, R.A., **Ambientes de Desenvolvimento de Software e o Projeto ADS** Revista Engenharia, Ciência e Tecnologia, Volume 04, Número 04, ISSN 1414-8692, pp 3 - 10, Vitória, Espírito Santo, Brasil, Julho/Agosto 2001.

MONTRESOR, A., DAVOLI, R., BABAOĞLU, Ö. **Middleware for dependable network services in partitionable distributed systems**. *SIGOPS Oper. Syst. Rev.* 35, 1 (Jan. 2001), 73-96. 2001. DOI= http://doi.acm.org/10.1145/371455.371463

MORO, César F. **Suporte à persistência de artefatos para o ambiente distribuído de desenvolvimento de software DiSEN**. 2003, 98 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Informática, Universidade Federal do Paraná, Curitiba, 2003.

MOURA, L. M. V., ROCHA, A. R. C. **Ambientes de Desenvolvimento de Software**, Publicações Técnicas COPPE/UFRJ, ES-271/92, Rio de Janeiro.

MUNRO, D.S. **On the Integration of Concurrency, Distribution and Persistence**. Ph.D. Thesis, University of St Andrews (1993).

NARENDRA, N. C. **Adaptive workflow management—an integrated approach and system architecture**. In Proceedings of the 2000 ACM Symposium on Applied Computing - Volume 2 (Como, Italy). J. Carroll, E. Daminani, H. Haddad, and D. Oppenheim, Eds. SAC '00. ACM Press, New York, NY, 858-865. 2000. DOI= http://doi.acm.org/10.1145/338407.338578

NUNES D. J. **PROSOFT - Projeto de ambiente de desenvolvimento de software**. Disponível em: <http://www.inf.ufrgs.br/~prosoft/>. Acesso em: jun. 2006.

NUNES, Daltro José, REIS, Rodrigo Quites, REIS, Carla Alessandra Lima. **A Synchronous Cooperative Architecture for the PROSOFT Software Engineering Environment.** Disponível em: <http://www.inf.ufrgs.br/~prosoft/>. Acesso em: jun. 2006. 1999.

OLIVEIRA, K., **Modelo para Construção de Ambientes de Desenvolvimento de Software Orientados a Domínio**, Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro, Brasil, 1999.

OLIVEIRA, H., MURTA, L. G. P., WERNER, C. M. L.. **Odyssey-VCS: Um Sistema de Controle de Versões para Modelos baseados no MOF**. In: XVIII Simpósio Brasileiro de Engenharia de Software, Ferramentas, 2004, Brasília. Anais do XVIII Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas. Brasília : Universidade de Brasília, 2004. v. 1. p. 85-80.

OLIVEIRA, H., MURTA, L. G. P., WERNER, C. M. L. . **Odyssey-VCS: a Flexible Version Control System for UML Model Elements**. In: ACM 12th International Workshop on Software Configuration Management, 2005, Lisboa. 12th International Workshop on Software Configuration Management, 2005. p. 1-16.

OLIVEIRA, K. M, SANTOS, G., ZLOT, F. et al. **Construção de Ambientes de Desenvolvimento de Software Orientados a Domínio na Estação TABA.** III Workshop Ibero-Americano de Engenharia de Requisitos e Ambientes de Software (IDEAS'00). Cancun, México, 2000.

OLIVEIRA, K. M., SANTOS, Gleison, ZLOT, Fabio, GUEDES, G., CERQUEIRA, A., SILVA, Cathia Galota M, MACHADO, L. F., VILLELA, Karina Lima, RAPCHAM, F., FALBO, R. A., TRAVASSOS, G. H., ROCHA, Ana Regina C da . **Construção de Ambientes de Desenvolvimento de Software Orientados a Domínio na Estação TABA**. In: IDEAS'00 - Terceiro Workshop IberoAmericano de Engenharia de Requisitos e Ambientes de Software, 2000, Cancun, 2000.

OSSHER, H., HARRISON, W., and TARR, P. **Software engineering tools and environments: a roadmap**. In Proceedings of the Conference on the Future of Software Engineering (Limerick, Ireland, June 04 - 11, 2000). ICSE '00. ACM Press, New York, NY, 261-277. 2000. DOI= http://doi.acm.org/10.1145/336512.336569.

PAASIVAARA, M., **Communication Needs, Practices, and Supporting Structures in Global Inter-Organizational Software Development Projects**, Proceedings of the International Workshop on Global Software Development at the 25th International Conference on Software Engineering, Portland, Oregon (2003), p. 59–63, http://gsd2003.cs.uvic.ca/gsd2003proceedings.pdf.

PASCUTTI, Márcia C. D. **Uma proposta de arquitetura de um ambiente de desenvolvimento de software distribuído baseado em agentes**. 2002. 102 f. Dissertação (Mestrado) - Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre.

PEDRAS, Maria Edith V. **Uma Ferramenta de Apoio ao Gerenciamento de Desenvolvimento de Software Distribuído.** 2003, 113 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Informática, Universidade Federal do Paraná, Curitiba. 2003.

PETERS, James F., PEDRYCS, Witold. **Engenharia de Software - Teoria e Prática.** Campus. 2001.

POHL, K., WEIDENHAUPT, K., DMGES, R., HAUMER, P., JARKE, M., KLAMMA, R.. **PRIME: Towards Process-Integrated Environments**. ACM Transactions on Software Engineering and Methodology, 8(4):343—410. 1999.

POPOVICH, S. S. **Rule-based process servers for software development environments**. In Proceedings of the 1992 Conference of the Centre For Advanced Studies on Collaborative Research - Volume 1 (Toronto, Ontario, Canada, November 09 - 12, 1992). J. Botsford, A. Ryman, J. Slonim, and D. Taylor, Eds. IBM Centre for Advanced Studies Conference. IBM Press, 477-497. 1992.

POZZA, Rogério dos Santos. **Proposta de um Framework para Aspectos do Gerenciador de Workspace no Ambiente DiSEN**. Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, Universidade Estadual de Maringá, Maringá. 2004.

PRESSMAN, Roger S. **Software Engineering, a Parctitioner´s approach**. Mc Graw Hill. 2001.

PRIKLADNICKI, R. **MuNDDoS - Um Modelo de Referência para Desenvolvimento Distribuído de Software**. PPGCC - FACIN - PUCRS, 2003. (Dissertação de Mestrado), Porto Alegre, RS, Brasil.

PRIKLADNICKI, R., AUDY, Jorge Luis Nicolas, EVARISTO, R. **Requirements management in global software development: preliminary findings from a case study in a SW-CMM context** In: International workshop on global software development, 2003, Portland, Oregon.Proceedings of IWGSD. Portland: IWGSD Press, v.1. p.1 – 10. 2003.

PYYSIAINEN, J., **Building Trust in Global Inter-Organizational Software Development Projects: Problems and Practices**, Proceedings of the International Workshop on Global Software Development at the 25th International Conference on Software Engineering, Portland, Oregon (2003), p. 69–75, http://gsd2003.cs.uvic.ca/gsd2003proceedings.pdf.

RAMAMPIARO, Heri, BRASETHVIK, Terje, WANG, Alf Inge. **Supporting Distributed Cooperative Work in CAGIS**. In 4th IASTED International Conference on Software Engineering and Applications (SEA'2000), Las Vegas, Nevada, USA, 6-9 November 2000. http://citeseer.ist.psu.edu/ramampiaro00supporting.html.

RAMESH, S., PERROS, H. G. **A multi-layer client-server queueing network model with synchronous and asynchronous messages**. In *Proceedings of the 1st international Workshop on Software and Performance* (Santa Fe, New Mexico, United States, October 12 - 16, 1998). WOSP '98. ACM Press, New York, NY, 107-119. 1998. DOI= http://doi.acm.org/10.1145/287318.287343.

REIS, Rodrigo Quites. **Uma proposta de suporte ao desenvolvimento cooperativo de software no ambiente PROSOFT**. Dissertação de Mestrado. Mestrado em Ciências da Computação. Universidade Federal do Rio Grande do Sul, UFRGS, Brasil. Ano de Obtenção: 1998.

REIS, Rodrigo Quites, LIMA REIS, Carla Alessandra, SILVA, Fábio Augusto das Dores, NUNES, Daltro José. **Um Modelo de Simulação de Processos de Software Baseado em Agentes Cooperativos**. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (SBES'99), 13, 1999, Florianópolis. Anais.... Florianópolis: SBC/UFSC, 1999. p. 163 - 178..

REIS, Rodrigo Quites, LIMA REIS, Carla Alessandra, NUNES, Daltro José. **Automated Support for Software Process Reuse Requirements and Early Experiences with the APSEE model**. In: INTERNATIONAL WORKSHOP ON GROUPWARE (CRIWG'2001), 17, 2001, Darmstadt, Germany. Proceedings. Washington: IEEE Computer Society Press, 2001. p. 50 – 57.

REIS, Rodrigo Quites. **APSEE-StaticPolicy: Sintaxe, semântica algébrica e exemplos de uma linguagem para verificação automática de políticas estáticas em modelos de processos de software**. Porto Alegre: PPGC, Universidade Federal do Rio Grande do Sul, 2001.

REIS, Carla Alessandra Lima.. **Uma abordagem flexível para execução de processos de software evolutivos.** Tese de doutorado. Universidade Federal do Rio Grande do Sul. Porto Alegre, BR-RS. 2003

RICHARDSON, D. J., O'MALLEY, T. O., MOORE, C. T., and AHA, S. L. **Developing and integrating ProDAG in the Arcadia environment**. In Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments (Tyson's Corner, Virginia, United States, December 09 - 11, 1992). SDE 5. ACM Press, New York, NY, 109-119. 1992. DOI= http://doi.acm.org/10.1145/142868.143759.

RONI, Luis Carlos de. **Padrões de projeto para o ambiente DiSEN**. Trabalho de conclusão de curso. Universidade Estadual de Guarapuava. 2006.

ROYCE, Walker. **Software Project Management: A Unified Framework.** Pearson. 1998.

RUH, W., MAGINNIS, F., BROWN, W. **Enterprise application integration**. New York, USA: John Wiley & Sons, Inc.. 2001.

SCHIAVONI, Flávio Luiz. **EDS - Distribuição Eletrônica de Software**. Relatório técnico. Universidade Estadual de Maringá. 2006.

SILVA, I. A., MANGAN, M. A. S., WERNER, C. M. L., **CVS-Watch: A Group Awareness Tool Applied to Collaborative Software Development**, In: LAWEB/WebMidia 2004, Ribeirão Preto, SP, Brasil, outubro 2004.

SILVA, L. F., PAULA, V. C. C. **Um Meta-Modelo para Especificação de Arquiteturas de Software em Camadas**. In: Simpósio Brasileiro de Engenharia de Software, 2001, Rio de Janeiro. 15 Simpósio Brasileiro de Engenharia de Software, v. 1. 2001.

SOKOLSKY, Michael H., KAISER, Gail E. **A framework for immigrating existing software into new software development environments**. Software Engineering Journal, 6(6):435--453, November 1991. http://citeseer.ist.psu.edu/sokolsky91framework.html.

SOUSA, A.L. R., LIMA REIS, C. A., REIS, R. Q., PIMENTA, M. S., NUNES, D. J. **Analisando a Interação de Gerentes e Desenvolvedores em Ambientes de Processo de Software: Classificação e Exemplos**. In: Jornadas Chilenas De Computación (I Workshop De Ingeniería De Software), 2001, Punta Arenas. Proceedings.... Santiago: Sociedad Chilena de Ciencia de la Computación, 2001.

SOUSA, A. L. R., LIMA REIS, C. A., REIS, R. Q., NUNES, D. J., PIMENTA, M. S. **Analisando a interação de gerentes e desenvolvedores em ambientes de processos de software**. In: I Workshop Chileno de Engenharia de Software, 2001, Punta Arenas, 2001.

STEINMACHER, I., AMORIM, E., SCHIAVONI, F. L., HUZITA, E. H. M. **GeCA: Uma Ferramenta de Engenharia Reversa e Geração Automática de Código**. In: Simpósio Brasileiro de Sistemas de Informação (SBSI), 2006, Curitiba. III Simpósio Brasileiro de Sistemas de Informação, 2006.

STEINMACHER, I., GIMENES, I. M. S., OLIVEIRA JUNIOR, E. A., TAKANO, E.T. **ExPSEE: Um Ambiente Experimental de Engenharia de Software Orientado a Processos** In: II Congresso Brasileiro de Computação, 2002, Itajaí..

TATA, S., GODART, C., WIIL, U. K. **Policies for cooperative hypermedia systems: concepts and prototype implementation**. In Proceedings of the Thirteenth ACM Conference on Hypertext and Hypermedia (College Park, Maryland, USA, June 11 - 15, 2002). HYPERTEXT '02. ACM Press, New York, NY, 140-141. 2002. DOI= http://doi.acm.org/10.1145/513338.513374.

TANENBAUM, Andrew S. **Distributed Operating Systems.** Pearson. 1994.

TANENBAUM, Andrew S., STEEN, Maarten Van. **Distibuted Systems Principles and paradigms.** Prentice Hall. 2002.

TAYLOR, R. N., BELZ, F. C., CLARKE, L. A., OSTERWEIL, L., SELBY, R. W., WILEDEN, J. C., WOLF, A. L., YOUNG, M. **Foundations for the Arcadia environment architecture**. SIGSOFT Softw. Eng. Notes 13, 5 (Nov. 1988), 1-13. 1988. DOI= http://doi.acm.org/10.1145/64137.65004

TEIXEIRA, H., MURTA, L. G. P., WERNER, C. M. L.. **LockED: Uma Abordagem para o Controle de Alterações de Artefatos de Software**. In: IV Workshop Ibero-americano de

Engenharia de Requisitos e Ambientes de Software (IDEAS'01), 2001, Costa Rica. Memorias Ideas 2001, 2001-a. v. 1. p. 348-359

TEIXEIRA, H., MURTA, L. G. P., WERNER, C. M. L. . **LockED: Uma Ferramenta para o Controle de Alterações no Desenvolvimento Distribuído de Artefatos de Software**. In: XV Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas, 2001, Rio de Janeiro. Anais do XV Simpósio Brasileiro de Engenharia de Software, 2001-b. p. 380-385.

TRAVASSOS, G. H. **O Modelo de Integração de Ferramentas da Estação TABA**. Tese de D. Sc., COPPE/UFRJ, Rio de Janeiro. 1994.

TSAI, W. T., SONG, W., PAUL, R., CAO, Z., HUANG, H. **Services-Oriented Dynamic Reconfiguration Framework for Dependable Distributed Computing**. In Proceedings of the 28th Annual international Computer Software and Applications Conference (Compsac'04) – v.1. (September 28 - 30, 2004). COMPSAC. IEEE Computer Society, Washington, DC, 554-559. 2004.

TURNER, Mark, BUDGEN, David, BRERETON, Pearl. **Turning Software into a Service**, Computer ,vol. 36, no. 10, pp. 38-44, October . 2003.

VILLELA, K., OLIVEIRA, K. M., SANTOS, G., ROCHA, A. R. C., TRAVASSOS, G. H. **Cordis-FBC: an Enterprise Oriented Software Development Environment** In: Workshop Learning Software Organization, Luzern. 2003.

VILLELA, K. L.,ROCHA, A. R. C.,,TRAVASSOS, G. H. **Definição e Construção de Ambientes de Desenvolvimento de Software Orientados a Organização** . In: III Simposio Brasileiro de Qualidade de Software, 2004, Brasilia. Anais do SBQS. Porto Alegre: Sociedade Brasileira de Computação, 2004. v. 1. p. 2-17.

VILLELA, K. L., SANTOS, G., MONTONI, M. A, BERGER, P., FIGUEIREDO, S. M. de, M., NOGUEIRA, S., ROCHA, A R. C., TRAVASSOS, G. H.. **Definição de Processos em Ambientes de Desenvolvimento de Software Orientados a Organização** . In: III Simposio Brasileiro de Qualidade de Software, 2004, Brasilia. Anais do SBQS. Porto Alegre: Sociedade Brasileira de Computação, 2004. v. 1. p. 22-37.

WANG, A. I. **A process centred environment for cooperative software engineering**. In Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering (Ischia, Italy, July 15 - 19, 2002). SEKE '02, vol. 27. ACM Press, New York, NY, 469-472. 2002.DOI= http://doi.acm.org/10.1145/568760.568841.

WERNER, C. M. L., BORGES, M., MATTOSO, M., BRAGA, R., CAMPOS, F., MANGAN, M., VIEIRA, V. **OdysseyShare: Desenvolvimento Colaborativo de Componentes**. In: Webmídia - CSCW Track. Salvador. Webmídia - CSCW Track. v. 1. p. 469-476. 2003.

WERNER, C. M. L., MANGAN, M., MURTA, L. G. P., PINHEIRO, R., MATTOSO, M., BRAGA, R., BORGES, M. **OdysseyShare: an Environment for Collaborative Component-based Development**. In: IEEE International Conference on Information Reuse and Integration, 2003, Las Vegas. IEEE International Conference on Information Reuse and Integration, 2003. v. 1. p. 61-68.

WERNER, C. M. L., MANGAN, M., MURTA, L. G. P., PINHEIRO, R., OLIVEIRA, A., MATTOSO, M., BRAGA, R., BORGES, M. **OdysseyShare: Um ambiente para o Desenvolvimento Cooperativo de Componentes**. In: XVI Simpósio Brasileiro de Engenharia de Software, Ferramentas, 2002, Gramado. XVI Simpósio Brasileiro de Engenharia de Software, Ferramentas, p. 444-449. 2002.

WIESE, Igor Scaliante. **Um modelo de interoperabilidade para ambientes de desenvolvimento distribuído de software**. Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, Universidade Estadual de Maringá, Maringá. 2006.

WILLIAMS, L. G. **Software process modeling: a behavioral approach**. In Proceedings of the 10th international Conference on Software Engineering (Singapore, April 11 - 15, 1988). International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos, CA, 174-186. 1988.

WIJNSTRA, Jan Gerben. **Supporting diversity with component frameworks as architectural elements**. Journal Title: International Conference on Software Engineering. 2000.

YOUNG, P. S. **Customizable Process Specification and Enactment for Technical and Non-Technical Users**. Doctoral Thesis. UMI Order Number: UMI Order No. GAX94-22217., University of California at Irvine. 1994.

ANEXO I – Outros ADSs encontrados na literatura

Além dos ambiente apresentados no segundo capítulo desta dissertação, outros ambientes foram encontrados na literatura. Os dados encontrados sobre estes ADS foram poucos e não permitiu que os mesmos fossem utilizados na comparação entre ambientes.

Neste Anexo são apresentados estes ambientes. Abaixo segue uma lista de ambientes que não foram encontrados dados significativos para este trabalho.

- ADDD
- GOSIP
- LEU
- MELMAC
- METAVIEW
- PROCESS JILL
- PROVENCE
- PYDONE
- SENTINEL

AHEAD

O PSEE AHEAD (Adaptable Human-centered Environment for the Administration of Development processes) (BECKER et al., 2001), desenvolvido na Aachen University of Technology, Alemanha, utiliza o conceito de delegação de processos para representação de processos distribuídos. Uma abordagem top-down é empregada, onde uma organização atua como supervisora do processo distribuído. A organização supervisora é vista como cliente no processo, delegando a execução de partes do processo para as demais organizações, que atuam como contratadas. Uma contratada pode, por sua vez, delegar partes do seu processo, atuando nesse caso como cliente das organizações sub-contratadas.

Cada organização contratada possui autonomia para executar a parte do processo delegada, podendo refinar as atividades nela definidas, desde que não altere a estrutura original. As informações sobre refinamentos realizados podem ser ocultadas do cliente. Alterações na estrutura não podem ser realizadas pelo fato de que uma delegação é vista como um contrato: uma vez que uma parte de processo tenha sido delegada, ela só pode ser alterada em comum acordo por ambas as organizações envolvidas na delegação.

ALF

No ALF os artefatos são armazenados no PCTE que fornece uma camada de transparência do sistema de arquivo UNIX (AMBRIOLA, CONRADI, FUGGETTA, 1997).

O ALF define seu modelo em uma linguagem chamada MASP/DL (*Model for Assisted Software Process Data Language*). Os modelos de processo consistem de uma hierarquia de fragmentos modelos chamados MASP. Cada MASP é definido por (BARTHELMESS, 2003):

- Um modelo de objeto que descreve os dados utilizados no fragmento;

- Um conjunto de operadores de tipo que são as ferramentas ou tipos de ferramentas que devem ser empregadas;
- Um conjunto de regras que reagem a eventos, como, por exemplo, o sucesso de uma compilação;
- Um conjunto de regras que definem as operações e sua seqüência.
- Uma definição de características invariantes que devem ser mantidas por todo o tempo do fragmento específico.

O ALF utiliza um sistema de arquivo virtual que mapeia o sistema de arquivo do Unix de maneira transparente para que as ferramentas possam interagir diretamente com os artefatos (BARTHELMESS, 2003).

O ALF define o contexto de trabalho para cada função de usuário no sistema. Dentro destas funções há os artefatos que podem ser compartilhados por vários usuários que exercem uma mesma função (BARTHELMESS, 2003).

APEL

Desenvolvido no Laboratoire Logiciels, Systèmes, Rèseaux, na França, APEL (CUGOLA, GHEZZI, 1998) (ESTUBLIER, CUNIN, BELKHATIR, 1998) (ESTUBLIER, AMIOUR, DAMI, 1999) é um PSEE com o objetivo de suportar interoperabilidade entre PSEEs heterogêneos, permitindo a construção de uma federação de processos capaz de gerenciar processos distribuídos complexos (FREITAS, MAIA, NUNES, 2004). Seus objetivos pricipais são (CUGOLA, GHEZZI, 1998):

1. Prover suporte para a interoperabilidade entre PSEEs heterogêneos permitindo que o desenvolvedor do processo construa uma federação de PSEEs para gerenciar processos complexos e distribuídos.
2. Prover suporte para a evolução do processo, de maneira a atender situações não previstas durante o andamento do projeto.

A arquitetura do APEL tem sido explicitamente trabalhada para suportar processos grandes e distribuídos. Ela tira benefícios de um paradigma composto de comunicação entre componentes que inclui tanto conexões ponto-a-ponto como baseada em eventos (CUGOLA, GHEZZI, 1999).

O APEL têm foco na integração de ferramentas distintas, não necessariamente PSEEs. Ele baseia-se na idéia de um modelo de processo comum, que define a interação entre as diferentes ferramentas. O modelo de processo comum é executado, e coordena a execução das demais ferramentas (FREITAS, MAIA, NUNES, 2004-b). Este controle é realizado de forma transparente, e cada PSEE executa de forma autônoma (FREITAS, MAIA, NUNES, 2004).

APSEE

O APSEE, um ambiente de engenharia de *software* centrado no processo desenvolvido no contexto do grupo de pesquisa Prosoft, provendo um conjunto de funcionalidades para suporte a processos distribuídos. Ele é uma extensão do modelo de definição de processos originalmente proposto para o PROSOFT (FREITAS, 2005).

Um resumo das características principais do meta-modelo APSEE é apresentado abaixo (FREITAS, 2005):

- É baseado em um paradigma baseado em atividades, e sua PML descreve um processo de *software* como um conjunto de atividades orientadas para um objetivo específico;
- Provê uma taxonomia hierárquica para os tipos de processos em um estilo orientado a objetos (O-O) para todos os componentes dos processos;
- Utiliza-se dos serviços de gerência de trabalho cooperativo e controle de versões fornecidos pelo repositório do ambiente;
- Permite a definição tardia de estruturas de processos, adiando a definição de estruturas concretas até o momento de execução do processo;
- Provê um conjunto dinâmico de mecanismos de controle para conexão de atividades em um processo.

Arcadia

O Arcadia´s ProcessWall não é um ambiente completo mas um servidor de estado de processo com a representação predefinida de passos, chamadas a tarefas, controle hierárquico das tarefas e controle do fluxo das tarefas (KAISER, POPOVICH, BEN-SHAUL, 1993).

O ambiente Arcadia possui uma abordagem de programação de processos que derivou da linguagem Ada e, portanto, executa seus processos de forma procedimental (LIMA REIS, REIS, NUNES, 1998).

O projeto Arcádia tem estressado a área de prototipação de componentes e ferramentas para ADS e a integração destas diversas capacidades em um ambiente protótipo (RICHARDSON et al., 1992). Ele parte de uma visão ampla de produtos de *software* e uma perspectiva similarmente ampla na necessidade de qualidade de cada artefato de *software*. A necessidade em assegurar a alta qualidade de produtos de *software* requerem flexibilidade e um conjunto de ferramentas poderoso para análise e testes (RICHARDSON et al., 1992).

Esta prototipação de ferramentas baseia-se no fato de que um ambiente deve ser flexível e extensível e isto gira em torno de várias fundamentações. Primeiramente, os usuários são diferentes e percebem suas necessidades de maneiras diferentes. Além disto, os projetos são diferentes e tem diferentes necessidades de suporte. Os projetos evoluem e suas necessidades mudam e a percepção dos usuários do sistema também muda. Estes fatores são o suficiente para afirmar que os ambientes devem ser flexíveis o bastante para alterar a natureza do suporte que o mesmo oferece aos seus usuários da maneira menos dolorosa possível. Há ainda o fator do surgimento de novas tecnologias e ferramentas que os ambientes devem absorver e incorporar para não se tornarem obsoletos e ineficientes (TAYLOR et al., 1988).

Seus principais componentes incluem o suporte para processo, gerenciamento de objetos, desenvolvimento de interface para usuário, definição e execução de processos, medição e validação (RICHARDSON et al., 1992).

Dynamite

O Dynamite é um ambiente desenvolvido na Universidade de Aachen, Alemanha, que utiliza uma representação gráfica baseada em redes de tarefas hierárquicas para apoiar o gerenciamento de processo de *software*. Neste ambiente, o gerente pode planejar o projeto,

alterar o modelo durante a execução, assinalar tarefas aos desenvolvedores, e organizar a documentação dos processos (HEIMANN, KRAPP, WESTFECHTEL, 1997) (SOUSA et al., 2001).

O Dynamite fornece para os desenvolvedores uma agenda e um working context que oferece diferentes graus de orientação ao desenvolvedor no mesmo processo. Os desenvolvedores têm acesso às redes de atividades, mas não podem modificá-las. A orientação do processo pode ser configurada dependendo do desenvolvedor e da tarefa. Após seleção de uma tarefa na agenda, o working context mostra versões de documentos de entrada; documentos de saída; guidelines (normas e regras gerais); e documentos auxiliares (para anotações do desenvolvedor). O desenvolvedor informa explicitamente o término de uma tarefa, porém é necessário que pelo menos uma versão de cada documento de saída tenha sido produzida e que as versões mais recentes de documentos de entrada tenham sido consumidas (HEIMANN, KRAPP, WESTFECHTEL, 1997) (SOUSA et al., 2001).

Os serviços fornecidos pelo ambiente Dynamite incluem (SOUSA et al., 2001):

- ***Feedback View*** - permite estimar as conseqüências da reativação de tarefas,
- ***Data Flow View*** - para cada atividade do processo, é visualizado a interdependência dos artefatos de *software* utilizados;
- ***Hierarchical View*** - permite a navegação nos diferentes níveis de detalhamento que constituem a definição hierárquica do processo,
- ***Control Flow View*** - apresenta a visão comportamental do processo, permitindo adaptação dinâmica do modelo através da criação de novas versões do modelo

Estação TABA

A Estação TABA foi definida e construída na COPPE/UFRJ para auxiliar na definição, implementação e execução de Ambientes de Desenvolvimento de *Software*. Para a construção de ADSOrg, a infra-estrutura fornecida pela Estação TABA foi novamente estendida e utilizada. A Estação TABA teve a sua infra-estrutura estendida para permitir a configuração de ambientes organizacionais (Ambiente Configurado) capazes de instanciar ADSOrg para projetos específicos e novas ferramentas foram desenvolvidas (VILLELA et al., 2003).

A estrutura da Estação TABA (TRAVASSOS, 1994) foi definida como um conjunto de componentes integrados que possuem controle sobre a sua existência, suas informações, estados e funcionalidades básicas associadas. A utilização, em conjunto, destes componentes permite a integração de ferramentas ao ambiente e provê recursos para a incorporação de ferramentas externas. A utilização desses componentes define, ainda, a filosofia de integração da Estação TABA e dos seus ambientes instanciados através de quatro tipos de integração (OLIVEIRA et al., 2000):

- Apresentação e interação, possibilitando a homogeneização das formas de apresentação das informações e das técnicas de interação com o usuário;
- Integração de gerenciamento e controle, provendo serviços e funcionalidades que permitam o funcionamento de forma organizada ao longo do processo de desenvolvimento;

- Integração de dados, estabelecendo a forma como as ferramentas da Estação realizarão o tratamento das informações;
- Integração do conhecimento, que torna possível os serviços básicos de armazenamento, gerenciamento e utilização do conhecimento descrito e adquirido ao longo do processo de desenvolvimento. A integração de conhecimento se dá através da utilização de servidores do conhecimento, no qual o conhecimento é modelado para reuso através de ontologias, sendo disponibilizado um conjunto de componentes que podem ser usados por várias ferramentas a serem desenvolvidas.

Desta forma, a Estação TABA possui facilidades para a definição de processos, métodos e ferramentas CASE a serem utilizadas no processo de desenvolvimento. Estes elementos estão organizados em um modelo de integração do ambiente, permitindo que diferentes ambientes sejam definidos e instanciados (OLIVEIRA et al., 2000).

A instanciação de um ADS na Estação TABA é feita a partir da definição de um processo de desenvolvimento que se caracteriza pela descrição de uma seqüência de atividades, suas ferramentas de apoio, produtos de *software* gerados e recursos consumidos. Este processo considera a definição de um Processo Padrão para a organização para a qual o ambiente será construído, que, por sua vez, serve de base para a definição dos processos de *software* adequados aos diferentes tipos de *software* desenvolvidos na organização e, a partir dos quais, processos para projetos específicos podem ser definidos (OLIVEIRA, 1999).

O Repositório da Organização e os Serviços/Ferramentas de Gerência do Conhecimento compõem a Infra-estrutura de Gerência do Conhecimento. Os Serviços/Ferramentas de Gerência do Conhecimento têm como objetivo facilitar o armazenamento de dados, conhecimentos e experiências no Repositório da Organização, além de apoiar a constante disseminação e atualização dos mesmos (VILLELA et al., 2003).

Merlin

O Merlin foi desenvolvido na University of Dortmund como parte de um projeto com o mesmo nome (BARTHELMESS, 2003) e está sendo continuado na Universidade de Paderborn em colaboração com a indústria local (BARGOUTHI et al., 1996).

O Merlin é construído sobre a base de dados GEMSTONE e oferece transações longas. Ele pode utilizar estratégias pessimistas ou otimistas. Os usuários podem ajudar a solução de conflitos fornecendo os dados transacionais dos objetos (BARTHELMESS, 2003).

O Merlin suporta uma especificação baseada em regras mesclando artefatos com funções em suas interações. Ele apresenta os documentos relevantes disponíveis de uma maneira gráfica e um documento é transferido para seu próximo contexto quando uma interação é encerrada (BARTHELMESS, 2003).

Seus elementos modelados incluem atividades, papéis, gerenciamento, artefatos e recursos. Os artefatos são associados a atividades que transformam estes objetos. Grupos de artefatos podem ser associados a papéis que possuam usuários que irão trabalhar sobre estes artefatos (BARTHELMESS, 2003). O desenvolvedor visualiza um espaço de trabalho (working context) que contém documentos, suas dependências e as atividades que devem ser realizadas nos documentos. A diferença do Merlin para outros PSEEs, como o Marvel, é que o desenvolvedor recebe todas as informações necessárias sobre a tarefa no working context. O feedback fornecido pelo desenvolvedor constitui a informação do estado do documento manipulado (SOUSA et al., 2001).

A cooperação é suportada de uma maneira diferente por meio de ambientes que transferem documentos entre contextos de trabalho de papéis distintos (BARTHELMESS, 2003). Seus usuários escolhem executar suas atividades na ordem que eles preferirem (BARTHELMESS, 2003).

O Merlin utiliza descrições como as do PROLOG para descrever processos de *software* e a seção concorrente de sincronização implementada por políticas de sincronização que podem ser sobrescritas em PROLOG. Os usuários do Merlin possuem uma agenda pessoal para manter as informações de projeto atualizadas de maneira coletiva. O Merlin também suporta engenharia de processo com um modelo de processo para definição e validação baseado em uma ferramenta gráfica com descrições similares a OMT que são mapeadas em PROLOG (BARGOUTHI et al., 1996).

A arquitetura do Merlin é baseada em um modelo de distribuição que assume vários contextos de trabalho associados a um processo. A comunicação entre os contextos de trabalho e a engine de processo é feita através de mensagens sobre o protocolo TCP/IP.

Os dados do processo e organização no Merlin são armazenados em um repositório central - ODBS O2. O acesso a estes dados são gerenciados por um gerenciador de Trava e de transação (BARGOUTHI et al., 1996).

Ophelia

O projeto Ophelia é uma plataforma open source que provê um ambiente distribuído de engenharia de *software*. Seu produto primário é um conjunto de interfaces que provêm captura de requisitos, modelagem e desenvolvimento de *software*, geração de código e acompanhamento de erros (bugs) acompanhados por uma metodologia apropriada para trabalhos distribuídos (BOLDYREFF et al., 2003).

O objetivo principal do Ophelia é unificar vários tipos de ferramentas de desenvolvimento de *software* em uma plataforma abstrata e transparente. Esta integração é feita utilizando um conjunto de interfaces CORBA que fornecem uma visão uniforme dos elementos e serviços disponíveis em uma determinada área do processo de desenvolvimento de *software*. O Ophelia inclui (BOLDYREFF et al., 2003):

- gerenciamento de requisitos;
- Modelagem (UML);
- gerenciamento de projetos;
- gerenciamento de documentação;
- *bug tracking*;
- repositórios para os elementos do projeto.

Assumindo esta condição, não é intenção do Ophelia desenvolver ferramentas mas criar um módulo de especificação de Interfaces (MIS – Module Interfaces Specification) (BOLDYREFF et al., 2003).

Não há necessidade de um repositório de artefatos para as ferramentas que trabalham a parte na plataforma Ophelia. Cada ferramenta pode possuir seu próprio repositório, algumas ferramentas podem compartilhar repositório e outras ferramentas (como métricas geradas dinamicamente) podem não possuir repositórios. O que integra todas estas ferramentas é a implementação das interfaces CORBA especificadas na plataforma. Os elementos do projeto

são alcançados não através de seus repositórios mas através de suas ferramentas. Os repositórios são integrados no OPHELIA como mais uma ferramenta (BOLDYREFF et al., 2003).

Oz

Desenvolvido na Universidade de Columbia, como um sucessor do MARVEL, Oz foi o primeiro PSEE descentralizado. Vários PSEEs desenvolvidos depois do OZ suportam distribuição física de desenvolvedores que cooperam em um processo de *software* utilizando uma arquitetura cliente/servidor para implementar sua infra-estrutura. As pessoas podem acessar os serviços fornecidos pelo PSEE executando ferramentas nos cliente e conectando o servidor e um repositório centralizado. Esta solução é viável para equipes médias e pequenas que sejam fortemente conectadas e que cooperam no mesmo processo através de uma LAN (CUGOLA, GHEZZI, 1998) (KAISER, POPOVICH, BEN-SHAUL, 1993).

O Oz utiliza a abordagem de fornecer uma federação de PSEE através da composição de diferentes instâncias de PSEEs locais. As instâncias não necessitam prover suporte para o processo de desenvolvimento executado por uma das organizações. Cada PSEE diferente executa de maneira autônoma de acordo com seu próprio processo. A interação entre os PSEEs ocorre com um local assumindo a coordenação e enviando e recebendo os dados necessários para o cumprimento de cada tarefa. Para tal, várias políticas organizacionais são suportadas. Estas políticas podem ser utilizadas, implementadas ou combinadas e são compostas de um conjunto de operações básicas (BEM-SHAUL, KAISER, 1995) (WANG, 2000).

Uma federação no PSEE Oz é composta de dois ou mais sub-ambientes. Cada sub-ambiente tem o controle completo do seu processo, ferramentas e dados enquanto permite acesso restrito que é determinado pelo próprio ambiente. Para suportar cooperação de sub ambientes, um sub processo comum deve ser definido. Isto é obtido através da definição de um ou mais tratamentos. Um tratamento define um sub-processo comum, um sub esquema comum de acesso aos dados, um conjunto de regras de acesso, para permitir a cooperação entre os sub ambientes (CUGOLA, GHEZZI, 1998) (FREITAS, MAIA, NUNES, 2004).

A essência do modelo de interoperabilidade do Oz reside em dois mecanismos para abstração da definição e da execução interprocessos: treaties – que permitem a definição de sub-processos compartilhados; e summits – que provêem a execução dos treaties definidos (FREITAS, MAIA, NUNES, 2004-b).

Oz utiliza a metáfora de "aliança internacional" (international alliance) para definir processos distribuídos. Segundo essa metáfora, países independentes (processos) operam de forma autônoma. Colaboração entre países (interoperabilidade entre processos) ocorre apenas com base em acordos prédefinidos (CUGOLA, GHEZZI, 1998) (KAISER, POPOVICH, BEN-SHAUL, 1993).

A colaboração é modelada como um encontro de ápice, onde os preparativos e as consequências do encontro são realizados de forma independente (por meio de sub-processos privados). Apenas o encontro de ápice em si é realizado de forma cooperativa (com base em um sub-processo compartilhado) (CUGOLA, GHEZZI, 1998) (KAISER, POPOVICH, BEN-SHAUL, 1993).

Um acordo (treaty) define um sub-processo comum, um sub-esquema comum e um conjunto de regras para acesso a dados. A definição de acordos envolve os seguintes aspectos (CUGOLA, GHEZZI, 1998) (KAISER, POPOVICH, BEN-SHAUL, 1993):

Baseado no Oz, o OzWeb permite que um conjunto de usuários colaborem acessando e manipulando um conjunto de documentos de hipermídia de acordo com um modelo de fluxo de dados bem definido. Ele utiliza as tecnologias web como http e HTML para prover o acesso e a manipulação de documentos de hipermídia mas introduz um modelo de workflow e algumas facilidades de suporte. O OzWeb amplia o servidor Oz para operar como um servidor web exportando todos os serviços necessários para montar um ambiente de colaboração de hipermídia de maneira que as pessoas dispersas pela Internet possam colaborar para alcançar um objetivo comum (CUGOLA, GHEZZI, 1998).

PIE

O PSEE PIE (Process Instance Evolution) é parte do projeto europeu ESPRIT, desenvolvido com base no APEL. O ambiente PIE é uma federação de PSEEs heterogêneos, com o objetivo de suportar e gerenciar a evolução das instâncias de processo. Na definição da federação deve-se estabelecer um conjunto de conceitos comuns que abstrai e integra os conceitos dos PSEEs individuais. Estes conceitos são definidos através do Universo Comum, que contém instâncias dos conceitos comuns. Os aspectos funcionais do comportamento do Universo Comum estão contidos no Modelo do Universo Comum. Deve-se definir também um Modelo Operacional, com o objetivo de definir como os aspectos mencionados no Modelo do Universo Comum devem ser tratados. O Modelo Operacional descreve a reação a mudanças no Universo Comum, indicando explicitamente as regras de consistência para cada reação, como ordem de invocação de componentes para determinada mudança no Universo Comum, controle transacional, etc (CUGOLA et al., 2000) (FREITAS, MAIA, NUNES, 2004).

O PIE têm foco na integração de ferramentas distintas, não necessariamente PSEEs. Ele baseia-se na idéia de um modelo de processo comum, que define a integração entre as diferentes ferramentas. O modelo de processo comum é executado, e coordena a execução das demais ferramentas. Tal controle é realizado de forma transparente, e cada PSEE executa de forma autônoma (FREITAS, MAIA, NUNES, 2004) (FREITAS, MAIA, NUNES, 2004-b).

PRIME

PRIME (Process-Integrated Modeling Environments) propõe o paradigma de interação orientada a ferramentas. Além de definir o processo é necessário modelar como o desenvolvedor vai realizar suas tarefas, o que requer conhecimento das ferramentas e cuidado com o excesso de rigidez do processo. Os documentos são descritos com mais detalhes que em outros PSEEs. As ferramentas do PRIME ajustam seu comportamento ao processo e às definições de método de Engenharia de *Software* utilizado. Por exemplo, para o desenvolvedor alterar um diagrama Entidade-Relacionamento (ER), o Editor de ER é utilizado, porém com seus serviços restritos apenas à manipulação de algumas entidades, conforme modelado. O restante do diagrama e outras funções da ferramenta ficam indisponíveis. O ambiente ainda fornece alternativas ao desenvolvedor que não quiser seguir o processo prédefinido (SOUSA et al., 2001).

ProcessWeaver

ProcessWeaver iniciou como uma experiência acadêmica francesa e hoje é um produto da Cap Gemini sendo um PSEE comercial (BARTHELMESS, 2003).

Em ProcessWeaver, os modelos de processo são desenvolvidos através de uma notação gráfica que permite a decomposição de atividades de alto nível e de processos de baixo nível. Estas visões gráficas também são suportadas por informações textuais armazenadas em formulários. Além disso, existe a possibilidade de armazenar processos para reutilização (LIMA REIS, 1998).

A execução do processo de *software* em ProcessWeaver inicia a partir da instanciação do processo. Entretanto, o ambiente não provê mecanismos para permitir modificação dinâmica do processo (durante a execução), nem suporta o registro da história do processo e coleta de métricas (LIMA REIS, 1998).

O ambiente ProcessWeaver, provê ferramentas para gerenciar tarefas individuais e automação do processo com características úteis para a depuração do modelo de processos, verificação de consistência, e monitoração. Existem ainda outras ferramentas para o suporte à execução de processos, tais como (LIMA REIS, 1998) (SOUSA et al., 2001):

- ***Method Editor*** - editor gráfico para definição de atividades hierárquicas;
- ***Activity Editor*** - permite a definição detalhada das atividades do processo através da associação dos produtos de entrada e saída, agentes, e funções de cada usuário envolvido permite especificação das entradas, saídas e cargos para as atividades;
- ***Cooperative Procedure Editor*** – representa o encadeamento das atividades utilizando Redes de Petri;
- ***Work Context Editor*** - permite que sejam definidas as janelas onde os agentes executam suas atividades.

No Process Weaver não há base de dados. O PSEE age diretamente sobre o sistema de arquivos aonde os artefatos estão armazenados (AMBRIOLA, CONRADI, FUGGETTA, 1997).

O Process Weaver fornece aos usuários em sua agenda as ações a serem tomadas e os documentos necessários para isto (BARTHELMESS, 2003). As agendas são o principal mecanismo de comunicação entre o ambiente e o desenvolvedor (LIMA REIS, REIS, NUNES, 1998). Além da Agenda, o ambiente fornece para os desenvolvedores working contexts. Sua interação é "desacoplada" pois é possível defini-la através da ferramenta Work context Editor. O ambiente permite delegação de tarefas através da Agenda. Quando uma tarefa é selecionada, seus detalhes são mostrados no Work Context (SOUSA et al., 2001).

Serendipity-II

O ambiente Serendipity-II (GRUNDY, 1999) (GRUNDY et al., 1998) foi desenvolvido em conjunto pela University of Waikato e pela University of Auckland, Nova Zelândia pelo grupo do Prof. John Grundy. Esta versão do ambiente é uma extensão descentralizada do Serendipity, uma ferramenta centralizada de suporte a processos.

O ambiente é distribuído, permitindo a flexibilidade na definição e monitoração de processos usando um formalismo orientado a atividades. Este paradigma estimula a interação entre usuários distribuídos e "desconectados" e o próprio programa de processo em execução é totalmente distribuído (replicado) entre os membros envolvidos. Desenvolvedores podem assumir o papel de gerentes, manipulando e monitorando a definição de processos. O Serendipity-II possui um mecanismo adicional de monitoração de eventos chamado Agente Local. Um agente local permite que um desenvolvedor descreva um conjunto de regras ECA

(Evento-Condição-Ação) usando um formalismo gráfico chamado VEPL (Visual Event Processing Language), que é na realidade um subconjunto da PML usada para definir processos no ambiente (SOUSA et al., 2001).

Todas as informações de modelos de processo que devem ser compartilhadas são replicadas entre os ambientes locais dos usuários, e a consistência é obtida através da troca de eventos ou objetos de atualização. Desse modo, usuários podem trabalhar desconectados, sincronizando posteriormente os modelos (FREITAS, MAIA, NUNES, 2004).

A arquitetura do Serendipity-II implementa a comunicação entre usuários e agentes de *software* por meio de comunicação ponto a ponto através da Internet, dispensando a existência de um servidor central. A rede pode ser modificada dinamicamente, porém cada ambiente precisa conhecer a localização dos demais ambientes (FREITAS, MAIA, NUNES, 2004).

Shamus

O ambiente Shamus explora uma abordagem centrada em documentos para o suporte ao desenvolvimento de *software*. A abordagem consiste em adicionar propriedades ativas aos documentos. As propriedades ativas podem ser usadas para implementar o controle de acesso e fazer notificações de alteração as partes interessadas (BARTHELMESS, 2003).

A modificação de um objeto é avisada em tempo real de maneira a evitar conflitos quando o código deste objeto retorna ao repositório. O sistema pode apresentar quais métodos e classes estão sendo modificadas antes mesmo deste ser enviado ao repositório. Outra propriedade ativa permite a automatização do controle de tarefas comuns ao desenvolvimento como a geração de código ou a compilação. Desta maneira, o ambiente mantém os usuários informados sobre as ações de outros usuários com o intuito de reduzir conflitos (BARTHELMESS, 2003).

SPADE

O projeto SPADE foi iniciado em 1991 por Bandinelli no Politecnico di Milano como o desenvolvimento de um ambiente para analise, desenvolvimento e ordenação de processos de *software* (AMBRIOLA, CONRADI, FUGGETTA, 1997) (BARTHELMESS, 2003).

O objetivo em longo prazo do projeto SPADE é fornecer métodos e técnicas para prover suporte a todo o ciclo de vida do processo (AMBRIOLA, CONRADI, FUGGETTA, 1997). A experiência com o PSEE SPADE demonstra que o suporte a alteração do processo é fundamental para a agilidade no desenvolvimento de projetos (CUGOLA, GHEZZI, 1998).

O SPADE utiliza a base de dados comercial orientada a objetos, o OODBMS O2, para armazenar os modelos de processos e os artefatos. O ambiente utiliza também as facilidades de integração de ferramentas que utilizam DEC FUSE, Sun Tooltalk e Microsoft OLE. Sua versão SPADE-1 foi completada em 1995 (AMBRIOLA, CONRADI, FUGGETTA, 1997) (BARTHELMESS, 2003) (LIMA REIS, 1998).

No ambiente SPADE o interpretador SLANG é o responsável pela execução das atividades. Cada atividade é executada por uma máquina de processo. Quando ocorre um evento de início de uma atividade, uma nova máquina de processo é ligada dinamicamente à definição da atividade, e a atividade inicia sua execução. A máquina de processo avalia as restrições associadas com as transições e analisa restrições de tempo. Uma transição é habilitada para disparar quando a condição e a restrição de tempo estiverem satisfeitas. O interpretador seleciona ciclicamente uma transição habilitada (automaticamente ou com

intervenção do usuário) e a dispara. A ocorrência de um evento produz uma mudança de estado que pode habilitar novos disparos (LIMA REIS, 1998).

A cooperação no nível de interação dos usuários no SPADE pode ser modelada em SLANG através do controle e da influência das ferramentas utilizadas por agentes do processo no ambiente (AMBRIOLA, CONRADI, FUGGETTA, 1997).

O SPADE possui uma agenda onde é possível delegar e/ou receber tarefas de outros. Possui interação "desacoplada", o que influencia na arquitetura do ambiente (o mecanismo de interação é independente). A agenda utiliza um arquivo de configuração para definir seu comportamento e a estrutura das tarefas. A agenda do SPADE pode ser adaptada para mostrar documentos ao invés de atividades.

O ambiente SPADE inclui um interpretador de processo capaz de executar modelos de processo escritos em SLANG. O ambiente de interação com usuário gerencia a interação entre usuários e ferramentas no ambiente. As ferramentas do ambiente também podem acessar o banco de dados e compartilhar dados entre si ou com o processo. A arquitetura de SPADE é dividida em três componentes (LIMA REIS, 1998):

- **Ambiente de Interação com Usuário**: responsável pela interação entre os usuários e as ferramentas do ambiente;
- **Ambiente de Execução do Processo**: responsável por executar os modelos de processo. A execução é feita concorrentemente por um conjunto de máquinas de processo que são dinamicamente criadas durante a execução;
- **Filtro:** gerencia a comunicação entre os dois ambientes.

SPADE suporta múltiplos usuários. Cada usuário interage com o processo através de um conjunto de ferramentas integradas. Existem duas classes de ferramentas: caixa-preta e estruturadas. As ferramentas caixa-preta são vistas pelas máquinas de processo como funções sobre argumentos de entrada resultando em alguma saída. O processo não tem controle sobre a ferramenta enquanto ela está trabalhando. Uma ferramenta estruturada, por sua vez, é decomposta em fragmentos visíveis. Isto torna possível atingir uma integração de baixa granulosidade no modelo de processo (LIMA REIS, 1998).

TeamWare

O ambiente TeamWare foi desenvolvido como protótipo e apresentado em uma tese de doutorado. Tendo sido desenvolvido no ambiente HyperCard para Apple Macintosh, este primeiro protótipo serviu para descrever uma estrutura básica de validação para algumas das principais idéias de interação humana defendidas pelo autor. TeamWare foi o precursor do Endeavors, desenvolvido em Java, que permite a alteração dinâmica de processos e geração de documentação de processos em HTML (SOUSA et al., 2001).

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo